Interactive Level-of-Detail Selection Using Image-Based Quality Metric for Large Volume Visualization

Chaoli Wang* Antonio Garcia† Han-Wei Shen‡

The Ohio State University

ABSTRACT

For large volume visualization, an image-based quality metric is much difficult to incorporate for level-of-detail selection and rendering without sacrificing the interactivity. In this paper, we introduce an image-based level-of-detail selection algorithm for interactive visualization of large volumetric data. The design of our image-based quality metric is based on an efficient way to evaluate the contribution of multiresolution data blocks to the final image. To ensure real-time update of the quality metric and interactive level-of-detail decisions, we propose a summary table scheme in response to run-time transfer function changes, and a GPU-based solution for visibility estimation. Experimental results on large scientific and medical data sets demonstrate the promise of our image-based level-of-detail selection algorithm.

CR Categories: E.4 [Coding and Information Theory]: Data compaction and compression; I.3.3 [Computer Graphics]: Picture and Image Generation—Viewing algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: level-of-detail, image-based quality metric, visibility computation, multiresolution rendering, large volume visualization

1 INTRODUCTION

Direct volume rendering with hardware texture mapping has become a standard technique for visualizing three-dimensional scalar fields from scientific and medical applications. An increasing number of these applications are now producing large-scale data sets, ranging from gigabytes to terabytes. One example is the Visible Woman (VisWoman) data set with resolution of $512 \times 512 \times 1728$ from The National Library of Medicine, generated as part of the Visible Human Project. Another example is the Richtmyer-Meshkov Instability (RMI) simulation performed at Lawrence Livermore National Laboratory. The simulation was executed on a $2048 \times 2048 \times 1920$ rectilinear grid, and it produced 7.5 gigabytes of data at each time step.

While it is common for the domain scientists to generate enormous amount of data, the state-of-the-art high-end graphics hardware is limited to only several hundred megabytes. This disparity severely challenges brute-force conventional hardware-texturing based volume rendering approaches, since the rendering time is proportional to the size of the data. New visualization systems that can scale adequately and ensure the level of interactivity are needed. Among several alternatives, multiresolution volume rendering [31, 15, 12] is a solution that can reduce the rendering cost dramatically. To perform interactive rendering, a multiresolution data hierarchy composed of multiple spatially partitioned blocks is first created. At run time, as the user navigates through the hierarchy, various amounts of data at different levels of detail can be extracted and used for the rendering.

Often, such a level-of-detail (LOD) is determined by various user-specified parameters, such as the tolerance of errors based on certain data-dependent metrics [17, 1, 29], different view-dependent parameters [15, 19], or both [32, 12, 11, 20]. In general, these metrics can be classified as *data-based* and *image-based* metrics. Data-based metrics measure the distortion between low and high (or full) resolution coarse-grained data blocks in the volume. The most widely used data-based metrics are *mean square error* (MSE), L^2 *norm*, and *peak signal-to-noise ratio* (PSNR). Although these metrics have clear physical meanings and are simple to compute, they are usually not good at predicting the quality of the rendered images due to the lack of correlation between data and image [13, 7, 27, 30, 20]. Image-based metrics focus on the ultimate images the user perceives, and strive to capture the quality loss in the rendered images introduced by rendering lower resolution data. These metrics are intrinsically view-dependent and more difficult to develop in conjunction with interactive LOD selections for large volume visualization. The major challenge lies in designing a comprehensive image-based metric, and updating the metric fast enough as not to harm the interactivity.

In this paper, we present an interactive LOD selection and rendering algorithm using an image-based quality metric for visualizing large volumetric data. We advocate an image-space approach for the quality metric design, based on a novel way to evaluate the importance values of coarse-grained multiresolution data blocks on the final image. Crucial to this approach is the notion of real-time update of the quality metric when the viewing or transfer function changes at run time. Utilizing programmable graphics hardware, we propose a GPU reduction scheme that can efficiently perform the visibility estimation for multiresolution data blocks. Consequently, all the view-dependent information can be updated in real time, ensuring interactive LOD selections. Moreover, our method does not make any assumption of the transfer function the user may use. By storing summary tables of data blocks in the multiresolution hierarchy with very small storage overhead (around 1% of the original volume data), we can update the quality metric within seconds whenever the transfer function changes at run time. As demonstrated in Section 5, our LOD selection algorithm performs better than traditional algorithms using data-based metrics, such as the MSE [1].

The remainder of the paper is structured as follows. First, we review related work in Section 2. In Section 3, we briefly introduce the multiresolution data representation we use for large three-dimensional data sets. In Section 4, we describe our multiresolution LOD selection and rendering algorithm in detail. Experimental evidence showing the superiority of our image-based LOD selection to the MSE-based one is provided in Section 5. The paper is concluded in Section 6 with future work for our research.

*e-mail: wangcha@cse.ohio-state.edu

†e-mail: agarcia@cse.ohio-state.edu

‡e-mail: hwshen@cse.ohio-state.edu

[1] In order to better perceive the image differences, the reader should view the rendered images in this paper electronically. Additionally, some video clips (http://www.cse.ohio-state.edu/∼wangcha/research/lod.avi) have been made as the supplementary material to compare our LOD selection method with the traditional MSE-based one.

2 Related Work

This section presents a brief review of related work in the areas of multiresolution data representation for volumetric data, image-based quality measurement, and visibility computation.

Building a multiresolution data hierarchy allows the user to visualize data at different scales, and balance image quality and computation speed. A number of techniques have been developed to provide hierarchical data representation for volumetric data, such as the *Laplacian pyramid* [9], multi-dimensional trees [32], and octree-based hierarchies [15, 1]. Muraki first introduced the use of wavelet transforms for volumetric data [23]. Westermann [31] presented a framework for approximating the volume rendering integral using multiresolution spaces and wavelet projections. More recently, Guthe et al. [12] presented a wavelet-encoded hierarchical representation for large volume data sets that supports interactive walkthroughs on a single commodity PC. Vector quantization [24, 28] has also been utilized to compress large data sets for direct volume rendering. Although this approach achieves good compression, it is not as flexible as a hierarchical data representation.

The lack of correlation between the type of error in an image and the response of the *human visual system* (HVS) to different types of errors prompted researchers to develop image-based metrics. Jacobs et al. [13] introduced an image-query metric for searching in an image database using a query image similar to the intended target. The metric makes use of multiresolution wavelet decompositions of the query and database images, and operates on the coefficients of these decompositions. Gaddipati et al. [7] presented a wavelet-based metric which captures the change in images wrought by operators and the image synthesis algorithms. Sahasrabudhe et al. [27] proposed a quantitative technique which accentuates differences in images and data sets through a collection of partial metrics. A study of different image comparison metrics, categorized into spatial domain, spatial-frequency domain, and perceptually-based metrics, was presented in [33]. Alternatively, Wang et al. [30] proposed the use of structural similarity for the design of image quality measures. Experimental results show that their *Structural Similarity Index* simulates the response of the HVS with low computation cost.

In the context of large volume visualization, an image-based metric is hard to develop because of the following reasons: First, image-based metrics need to consider run-time information, such as the viewing, projection, and occlusion. Unlike most data-based metrics which can be easily computed in a preprocessing stage, to get view-dependent occlusion information for a large data set, one has to resort to either sophisticated precomputation with considerable overhead [8] or run-time calculation with rough approximation [11]. Next, since the user may adjust the transfer function during the rendering in order to reveal different features, image-based metrics should be adaptive to run-time transfer function changes. Previous work on large data visualization was either dependent on the input transfer function [20], or limited to a family of transfer functions consisting of a series of basis functions [8]. Last, the human observer plays a central role in perceiving the image quality. Therefore, image-based metrics should also reflect the human perception [25] involved in the volume rendering process. This includes a wide variety of spectrums, such as the perception of distance, coverage, shape, color, occlusion, texture, and lighting. In this paper, we integrate an image-based quality metric within the multiresolution LOD selection and rendering framework.

To accelerate the process of image generation, visibility computation has long been employed for occlusion culling [2] in rendering large polygonal models as well as volumetric data sets. Klosowski and Silva [14] introduced the time-critical *Prioritized-Layered Projection* (PLP) algorithm for fast rendering of high depth complexity scenes, using a solidity-based metric for visibility estimation. A similar approach that integrates occlusion culling with view-dependent rendering was given in [4]. Gao et al. [8] proposed a *Plenoptic Opacity Function* (POF) scheme, which encodes the view-dependent opacity of a volume block, for visibility culling of large volume data. Utilizing visibility information for multiresolution volume rendering has not been widely studied. A conservative way of visibility testing that assumes a uniform opacity for each data block was presented in [11]. In [20], a low resolution ray-casting renderer was used to estimate the average opacity of each block, followed by empirical tests for approximating the simplified discrete rendering equation with no emission factor.

3 Multiresolution Data Hierarchy

To build a multiresolution data hierarchy from a large three-dimensional data set, we use wavelet transforms to convert the data into a hierarchical multiresolution representation, called the *wavelet tree* [12]. The wavelet tree construction algorithm starts with subdividing the original 3D volume into a sequence of blocks with the same size (assuming each has N voxels). These raw volume blocks form the leaf nodes of the wavelet tree. After performing a 3D wavelet transform to each block, a low-pass filtered subblock of size $N/8$ and wavelet coefficients of size $7N/8$ are produced. The low-pass filtered subblocks from eight adjacent leaf nodes in the wavelet tree are grouped into a single block of N voxels, which becomes the low resolution data stored in the parent node. We recursively apply this 3D wavelet transform and subblock grouping process in a bottom-up manner till the root of the tree is reached, where a single block of size N is used to represent the entire volume. To reduce the size of the coefficients stored in the wavelet tree, the wavelet coefficients in each tree node are set to zero if they are smaller than a user-specified threshold. These wavelet coefficients are then compressed using run-length encoding combined with a fixed Huffman encoder. Note that in the wavelet tree, the multiresolution data blocks associated with all the tree nodes have data of the same size, which is N. However, the spatial resolutions they represent may vary, depending on which level of the tree the corresponding nodes lie on.

4 LOD Selection and Rendering Algorithm

Our multiresolution LOD selection and rendering algorithm optimizes the quality of rendered images through the use of an image-based quality metric. Our quality metric evaluates the importance values of multiresolution data blocks, by examining the contribution of data blocks to the final image, based on the *discretized volume rendering integral* (DVRI). The evaluation approximates the emission of each block, as well as taking into account the occlusion caused by the blocks in front of it. To capture the multiresolution error in the data hierarchy, we modulate the importance value with the distortion between low and high resolution data blocks, calculated in the roughly perceptually-uniform CIE $L^*u^*v^*$ (CIELUV) color space. To ensure real-time update of the quality metric, we propose a summary table scheme in response to changes of the transfer function, and a GPU reduction scheme for visibility estimation. At run time, for a given viewing direction, the LOD selection is made based on a priority queue scheme with the priority values of multiresolution blocks set as their importance values. The wavelet tree traversal maintains the LOD as a cut through the hierarchy, and the importance values dictate the sequence of LOD refinements. A certain number of blocks up to a user-specified budget are extracted and sent to the texture hardware for rendering.

4.1 Volume Rendering Integral

According to the emission-absorption optical model [21], the *volume rendering integral* (VRI) that calculates the amount of light I along a viewing ray r through the volume is given by:

$$I_r = \int_0^D \tilde{c}(s(\vec{x}(\lambda))) \exp(-\int_0^\lambda \tau(s(\vec{x}(\lambda'))) \, d\lambda') \, d\lambda \qquad (1)$$

where $s(\vec{x}(\lambda))$ is the scalar value at position $\vec{x}(\lambda)$ in the volume, parameterized by the distance λ to the viewpoint; $\tilde{c}(s)$ is the volume source term or intensity; $\tau(s)$ defines the attenuation function.

In general, the VRI cannot be evaluated analytically. Therefore, practical volume rendering algorithms discretize the VRI by numerical approximation. Using Riemann sum for n equal ray segments of length D/n, and further approximating the exponential function with the first two terms of the Taylor series expansion, we get the *discretized volume rendering integral* (DVRI) [22], also known as the compositing equation [18]:

$$I_r = \sum_{i=0}^{n} c(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1}(1-\alpha(s_j)) \qquad (2)$$

where $c(s)$ and $\alpha(s)$ define the color and opacity transfer function. This equation denotes that at each discrete sample position i along the viewing ray r in the volume, light is emitted according to the term $c(s_i)\alpha(s_i)$, which is absorbed by the volume at all positions along r in front of i according to the term $\alpha(s_j)$. Equation 2 serves as the foundation for our design of importance values for multiresolution data blocks.

4.2 Importance Value Design

The DVRI in Equation 2 evaluates the amount of light on a per-ray basis. It is also possible to look at the equation on a per-slice basis, which leads to the popular slice-based compositing technique for volume rendering. In this paper, the underlying entity for our LOD selection and rendering algorithm is a data block. Therefore, we evaluate the importance values of multiresolution data blocks by approximating Equation 2 on a per-block basis. The importance value of a data block b along the viewing direction r is calculated as follows:

$$I_b = (c(\mu)\alpha(\mu) \cdot t \cdot a) \cdot v \qquad (3)$$

where μ is the mean scalar data value of block b; $c(\mu)$ and $\alpha(\mu)$ define the color and opacity transfer function respectively (we actually calculate the magnitude of its corresponding CIELUV color, as explained later in Equation 9); t is the average thickness (the length of the viewing ray segment inside the block) of block b; a is the screen projection area of the block, and v is its estimated visibility. Similar to Equation 2, here $((c(\mu)\alpha(\mu) \cdot t \cdot a)$ approximates the emission of block b along direction r, and v accounts for the attenuation. Given a viewing direction r, I_b essentially evaluates the contribution of block b to the final image.

If we record the mean scalar value μ of each block during the construction of the multiresolution data hierarchy, we can quickly compute $c(\mu)$ and $\alpha(\mu)$ at run time. Also, given a viewing direction r, the average thickness t and projection area a of a block can be easily calculated. However, to obtain the estimated visibility v of a block is non-trivial, and we will describe our real-time GPU-based solution in Section 4.5.

Even if two multiresolution data blocks have the same approximate emission and absorption terms, there still might be different distortions between the two blocks and their children. Taking account of the relative distortion, we modulate the importance value I with the multiresolution error between low and high resolution data blocks. Equation 3 becomes:

$$I_b = (c(\mu)\alpha(\mu) \cdot \varepsilon \cdot t \cdot a) \cdot v \qquad (4)$$

where ε is the distortion between block b and its higher resolution child blocks, normalized to $[0,1]$. The motivation behind this modulation is that if a block contains larger distortion, then it should receive a higher priority value for LOD refinement.

4.3 Multiresolution Error Evaluation

The multiresolution error ε evaluates the distortion of low and high (or full) resolution data blocks in the data hierarchy. Previously, researchers have proposed various ways to calculate the error in the scalar data space [17, 1, 29], and in the RGB [5, 11] or CIELUV [20] color space. In this paper, we take an image-space approach and opt to evaluate the multiresolution error in the perceptually adapted CIELUV color space, as suggested by Glassner [10].

First, let us consider two data blocks b_i and b_j, where b_j is an immediate child block of b_i. We define the multiresolution error between b_i and b_j as follows:

$$\varepsilon_{ij} = \tilde{\sigma}_{ij} \cdot \frac{\tilde{\mu}_i^2 + \tilde{\mu}_j^2 + C_1}{2\tilde{\mu}_i\tilde{\mu}_j + C_1} \cdot \frac{\tilde{\sigma}_i^2 + \tilde{\sigma}_j^2 + C_2}{2\tilde{\sigma}_i\tilde{\sigma}_j + C_2} \qquad (5)$$

where $\tilde{\sigma}_{ij}$ is the covariance between b_i and b_j; $\tilde{\mu}_i$ and $\tilde{\mu}_j$ are the mean values of b_i and b_j respectively; $\tilde{\sigma}_i$ and $\tilde{\sigma}_j$ are the standard deviations of b_i and b_j respectively (small constants C_1 and C_2 are included to avoid instability when $\tilde{\mu}_i\tilde{\mu}_j$ and $\tilde{\sigma}_i\tilde{\sigma}_j$ are very close to zero):

$$\tilde{\sigma}_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(\tilde{x}_{ik} - \tilde{\mu}_i)(\tilde{x}_{jk} - \tilde{\mu}_j) \; ; \qquad (6)$$

$$\tilde{\sigma}_i = \frac{1}{N-1}\sum_{k=1}^{N}(\tilde{x}_{ik} - \tilde{\mu}_i)^2 \; ; \quad \tilde{\sigma}_j = \frac{1}{N-1}\sum_{k=1}^{N}(\tilde{x}_{jk} - \tilde{\mu}_j)^2 \; . \qquad (7)$$

Here, N is the number of voxels in the block, and $\tilde{x}$ is the volume source term. Equation 5 consists of three parts, namely, *covariance*, *luminance distortion*, and *contrast distortion*. The first part is the covariance between b_i and b_j, which measures the degree of linear correlation between the two blocks ($\tilde{\sigma}_{ij}$ is always non-negative since we actually calculate $\tilde{\sigma}_{ij}$ based on the CIELUV color differences of the pairs $(\tilde{x}_{ik}, \tilde{\mu}_i)$ and $(\tilde{x}_{jk}, \tilde{\mu}_j)$, as explained in Equations 8 - 10). Even though b_i and b_j are linearly related, there still might be relative distortions between them. Therefore, we add two more parts to the equation. The second one, measures how close the mean luminance is between b_i and b_j. The minimum value of 1.0 is achieved if and only if $\tilde{\mu}_i = \tilde{\mu}_j$. On the other hand, $\tilde{\sigma}_i$ and $\tilde{\sigma}_j$ can be viewed as estimate of the contrast of b_i and b_j, so the third part measures how similar the contrasts of the two blocks are. Also, the minimum value of 1.0 is achieved if and only if $\tilde{\sigma}_i = \tilde{\sigma}_j$. Collectively, these three parts capture the distortion between the two blocks. The luminance distortion and contrast distortion are originally from the image quality assessment literature [30], and have been shown to be consistent with the luminance masking and contrast masking features in the HVS, respectively.

One should note that the input source terms, $\tilde{x}$ and $\tilde{\mu}$, are CIELUV color values, rather than original scalar data values. Accordingly, we define $\tilde{x}_{ik}$ and $\tilde{\mu}_i$ as follows ($\tilde{x}_{jk}$ and $\tilde{\mu}_j$ can be defined similarly):

$$\tilde{x}_{ik} = \Delta E(f(c_{rgb}(x_{ik}) \cdot \alpha(x_{ik})), \, (0,0,0)) \qquad (8)$$

$$\tilde{\mu}_i = \Delta E(f(c_{rgb}(\mu_i) \cdot \alpha(\mu_i)), \, (0,0,0)) \qquad (9)$$

where x_{ik} is the scalar data value at the kth voxel position in block b_i; μ_i is the mean scalar value of b_i; c_{rgb} and α define the color and opacity transfer function; f is the function that converts an RGB color to its CIELUV color [6]; ΔE is the Euclidean distance between a pair of colors specified in the CIELUV color space:

$$\Delta E = \sqrt{\Delta L^{*2} + \Delta u^{*2} + \Delta v^{*2}} \qquad (10)$$

where ΔL^*, Δu^*, and Δv^* are the differences of L^*, u^*, and v^* components for the pair of CIELUV colors.

Equation 5 only calculates the multiresolution error ε_{ij} between a pair of parent-child blocks: b_i and b_j. In our multiresolution data hierarchy, a parent block has eight immediate child blocks, thus we compute the error as the summation of the errors between the parent block and its eight child blocks. We also take into account the maximum error of the child blocks, as an approximation of the error between the parent block and the original full-resolution data block it represents. Written in equation:

$$\varepsilon_i = \sum_{j=0}^{7} \varepsilon_{ij} + \max\{\varepsilon_j|_{j=0}^{7}\} \tag{11}$$

where ε_i and ε_j are the multiresolution errors of blocks b_i and b_j respectively.

As a special case, if block b_i is associated with a leaf node in the hierarchy, we define $\varepsilon_i = C_3$, where C_3 is a small constant. All the multiresolution errors in the data hierarchy are normalized before being used in Equation 4 for the calculation of importance values.

4.4 Summary Table Scheme

As we can see, the calculation of multiresolution error ε in Equation 11 requires the input of the color and opacity transfer function (Equations 8 and 9). At run time, whenever the user adjusts the transfer function, the multiresolution errors in the entire data hierarchy have to be recomputed from the beginning. This entails a considerable amount of computation overhead and makes the whole LOD selection and rendering process non-interactive. In the following, we describe a summary table scheme that ensures real-time update of the errors in response to transfer function changes.

Our summary table scheme is based on the observation that, for large data sets, the range size of the scalar data is often many orders of magnitude smaller than the spatial size of the volume. For instance, the RMI data set is byte (8-bit) data with range size of 256. However, the spatial size of the volume is $2048 \times 2048 \times 1920$. Therefore, instead of calculating Equations 6 and 7 for each voxel, it suffices to count the frequencies of unique error terms, which is much faster (similar observations have been made and utilized in [5, 16]). In the case of byte data, there are $256^2 = 65536$ combinations for $\tilde{\sigma}_{ij}$, and only 256 cases for $\tilde{\sigma}_i$ or $\tilde{\sigma}_j$. To compute the error, rather than adding individual error terms voxel by voxel, we add the products of a unique error term and the frequency of that term.

To realize this, first of all, for each of the data blocks at the multiresolution hierarchy, we precompute the mean scalar value μ, and keep a local *histogram table* $\mathbb{H}$ (256 entries):

$$\mathbb{H}\,(x_i,\ m)$$

where x_i is the scalar value, m is the frequency of x_i in the block.

Next, for each data block associated with a non-leaf node in the hierarchy, we keep a local *correspondence table* $\mathbb{C}$ (65536 entries):

$$\mathbb{C}\,(x_i,\ x_j,\ m)$$

where x_i is the scalar data value in the current (parent) block; x_j is the data value in its immediate eight child blocks; m is the frequency of the data pair. We refer to these histogram and correspondence tables as *summary tables*. They are created during the construction of the data hierarchy and are precomputed only once. Besides this, we keep a global *distance table* $\mathbb{D}$ ($1+2+\ldots+255 = 32640$ entries):

$$\mathbb{D}\,(x_i,\ x_j,\ \Delta E)$$

where x_i and x_j are scalar data values, and $x_i < x_j$; ΔE is the distance between x_i and x_j in the CIELUV color space.

Finally, we keep a global *function table* $\mathbb{F}$ (the number of entries in the transfer function, usually 256):

$$\mathbb{F}\,(rgb\alpha,\ L^*u^*v^*)$$

where $rgb\alpha$ is the RGB color and opacity in the current transfer function, $L^*u^*v^*$ is the corresponding CIELUV color. The global distance table and function table are initialized at run time and are updated when the transfer function changes.

At run time, we can quickly calculate the multiresolution error ε for each block using Equations 5 - 11, by looking up the mean scalar value μ and summary tables ($\mathbb{H}$ and $\mathbb{C}$) stored in each of the blocks, as well as the global distance table $\mathbb{D}$ and function table $\mathbb{F}$. The lookup relationships are as follows:

$$\begin{aligned} \tilde{\mu}_i, \tilde{\mu}_j &\leftarrow \mu, \mathbb{F}\,; \\ \tilde{\sigma}_i, \tilde{\sigma}_j &\leftarrow \mathbb{H}, \mathbb{F}\,; \\ \tilde{\sigma}_{ij} &\leftarrow \mathbb{C}, \mathbb{D}\,. \end{aligned}$$

Whenever the user changes the transfer function, only the global distance table and function table need update.

For data sets other than byte data, quantization is necessary in order to reduce the size of summary tables (otherwise, the size of these tables could be even larger than the size of actual data blocks, and the time for error calculation would increase dramatically). For example, one can quantize the scalar data range into 256 levels either uniformly or based on the histogram of the whole data set. In this way, the total size of the summary tables will remain small regardless of the data type of the input volume.

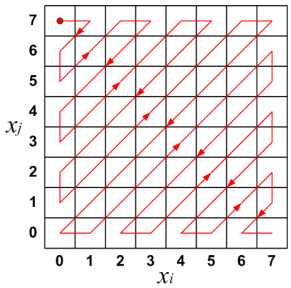

Figure 1: Run-length encoding on the correspondence table $\mathbb{C}$ in a zigzag manner. An example of an 8×8 table is shown here. The encoding starts from the red circle, and follows the red arrows.

One can observe that, usually there is a strong degree of correlation between parent and child blocks in the data hierarchy. This means that in the correspondence table $\mathbb{C}$, when x_i is close to x_j (i.e., the entry is close to the major diagonal of the table), the frequency m is large. m is smaller, actually often zero, if the entry is further away from the major diagonal. Leveraging this observation, we can perform run-length encoding on the correspondence table $\mathbb{C}$ in a zigzag manner, as illustrated in Figure 1. The zigzag run-length encoding not only reduces the storage of correspondence tables, but also improves run-time performance. Testing on the RMI data set shows that by storing the run-length encoded correspondence tables $\mathbb{C}$, the total size of summary tables reduces from 208MB to 44.1MB, and accordingly, the time to update multiresolution errors decreases from 43 seconds to 13 seconds.

4.5 GPU-Based Visibility Estimation

Obtaining the exact visibility of the multiresolution data blocks requires rendering the blocks in the stage of visibility determination. This is similar to rendering the entire hierarchy, which could be rather slow and defeats the purpose of the visibility test. For coarse-grained multiresolution rendering, getting an approximate visibility of a block suffices, and the visibility computation should be done prior to the actual rendering of blocks.

For multiresolution volume rendering, opacity corrections are necessary to generate correct images to compensate for the varying slice distances within data blocks of different resolutions. For a data block at the lth level of the hierarchy ($l = 0$ is the leaf node level), its opacity α_l is corrected from α_0 as follows:

$$\alpha_l = 1 - (1 - \alpha_0)^{\gamma^{-l}} \tag{12}$$

where α_0 is the opacity corresponding to the full resolution data, γ ($= 2.0$ for an octree-based hierarchy) is the refinement factor between two consecutive levels. If the opacities of multiresolution data blocks are corrected according to Equation 12, it follows that the visibility of a data block is independent of the resolutions of all the occluding blocks in front of it. Therefore, we can always render a lower resolution of the data (for example, the root of the data hierarchy) by drawing front-to-back view-aligned slices, and evaluate the approximate visibility of all the blocks during the slicing, as illustrated in Figure 2. The visibility of a block is computed as $(1-\alpha)$, where α is the *average* opacity within the block's screen projection on the opacity map, accumulated right before the first slice intersecting the block (α is considered as the accumulated opacity in front of that block). Note that a conservative way of taking the *minimum* opacity, commonly used in occlusion culling, is unnecessary. For occlusion culling, the decision is to either render or discard a block, and getting the minimum opacity is essential to avoid producing incorrect images by leaving holes. For multiresolution rendering, the whole volume is rendered anyway, because the question is to select proper LODs for different regions within the volume, rather than to render or discard a region. Therefore, it is reasonable to get the average instead of the minimum opacity.

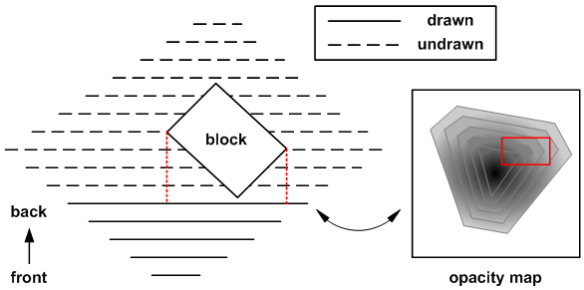

Figure 2: Visibility estimation via rendering a lower resolution of the data. The visibility of a block is acquired when its nearest vertex is in-between the current slice and the latest drawn one.

To compute the average opacity for a data block, a naive way is to read the alpha channel of the framebuffer to an off-screen buffer after a certain number of slices are rendered, and iterate through the pixels that the data block projects to and obtain an average of the opacities. This software approach is slow due to the framebuffer reads from the GPU to the CPU. The testing time is linear to the size of output images and the number of pixels each block projects to. OpenGL's extensions, such as the occlusion queries, are not suitable for our case because it is the alpha channel and not the depth channel that must be considered.

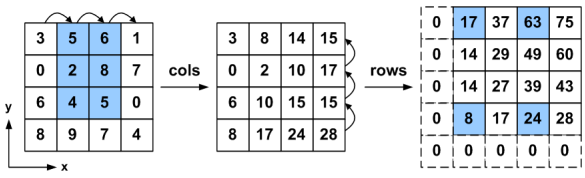

Figure 3: The construction of the SAT. An example of a 4×4 table $\mathbb{T}$ is shown here. The sum of any rectangular area bounded by $[x_0,y_0]$ and $[x_1,y_1]$ is calculated as: $\mathbb{T}(x_1,y_1)-\mathbb{T}(x_0-1,y_1)-\mathbb{T}(x_1,y_0-1)+\mathbb{T}(x_0-1,y_0-1)$. Any table entry out of bound uses zero.

Crow [3] conceived the *summed area tables* (SATs) that have been used for antialiasing. The construction of such a table is linear to the number of pixels on the area being considered, in our case the whole rendering screen. However, it only takes constant time to retrieve the sum over any rectangular area, which is done in one addition and two subtractions. This fits perfectly in getting the corresponding averages from the projections of the blocks. Building a SAT can be done by reading the framebuffer, and successively adding the columns from left to right and then the rows from bottom to top, as shown in Figure 3. To minimize the transferring of pixels from the GPU to the CPU, we move all operations to the GPU. In the following, we describe our GPU reduction scheme in detail.

GPUs and fragment programs provide the means to construct SATs. Nowadays, GPUs also support texture objects that have 32-bit floating-point channels, which is important for SATs since the sums require more precision. Following the same way as we build SATs in the CPU, an immediate choice is to switch the implementation of SATs to a *pbuffer* in the GPU. However, this requires that the pbuffer is treated as both an input and an output texture, which highly depends on the kinds of hardware and graphics library available. To date, most implementations do not have this capability. Therefore, we take an alternative and build the SATs in passes with the support of pbuffers having double auxiliary buffers: one is the input, and the other is the output. Both auxiliary pbuffers have the same size as the rendering pbuffer.

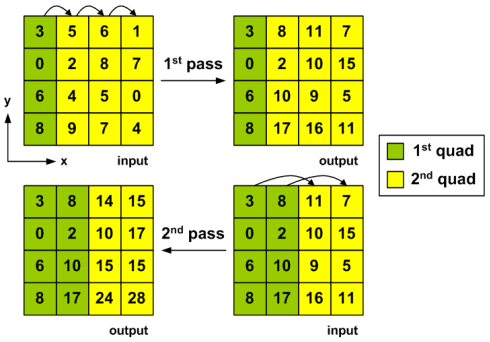

Figure 4: The construction of the SAT in the GPU. An example of a 4×4 table is shown here. Only the passes operating on the columns are illustrated.

As illustrated in Figure 4, the construction of a SAT in the GPU is as follows: First of all, we treat the alpha values from the rendering pbuffer as a texture and map it to two quads in the input pbuffer: the first quad only covers the texels of the first column, and the second covers the rest of the texture. In the first pass, each texel of the first quad is directly output to the same position at the output pbuffer, while each texel of the second quad adds the texel left to it as the output. In the second pass, the auxiliary pbuffers are swapped and the first quad doubles in size. Each texel in the first quad remains output unchanged, and each texel of the second quad now adds the texel two positions left to it as the output. This process continues until the first quad is half the size of the rendering pbuffer. Then, the process is repeated over the rows in a similar way to complete the SAT. For a rendering screen with resolution of n^2, the number of passes needed is $2\log(n)$.

Getting the average opacity for a block is performed by another fragment program that looks up the texture in the output auxiliary pbuffer holding the SAT. The program renders a quad that covers one pixel and has two sets of texture coordinates: the first set is the upper-right corner of the projection of the block in query, and the second the dimensions of that projection. With those values, the four corners of the projection are built in the fragment program and used to lookup the sums, which are combined to produced the sum in that area, as shown in Figure 3. Dividing the sum by the area of the projection we get the average for the block. The average value is then returned to the CPU.

Finally, producing the SAT in the GPU requires that the rendering of the lower resolution data is done to a pbuffer. Techni-

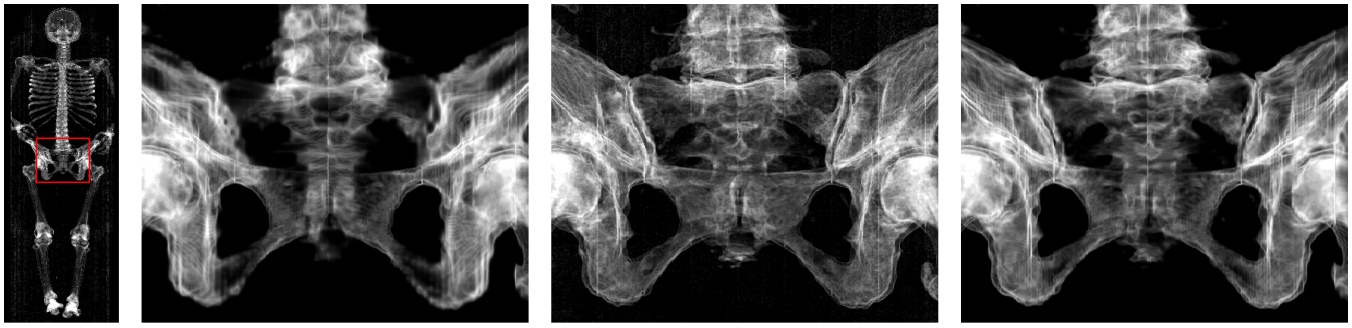

Figure 5: (a) shows an overview of the VisWoman data set and (b)-(d) show a zoom to the pelvis. One can observe that image (d) (image-based, 71 blocks, 7.31%) shows more details than image (b) (MSE-based, 75 blocks, 7.72%). The reference image (c) is rendered with full resolution (972 blocks).

cally, blending in a pbuffer using OpenGL blending functions is extremely slow, so we must resort to do the blending by ourselves. A slice of the rendering is blended in the pbuffer with a fragment program that performs the *over* operator [26] of volume rendering.

4.6 LOD Selection and Rendering

At run time, the user specifies the number of blocks as a budget for rendering. Given a viewing direction, the LOD selection is made based on a priority queue scheme. The priority values of blocks are set as their importance values calculated according to Equation 4 (where ε is updated per view and v per transfer function). Thus, a block with a higher importance value is more likely to be selected for refinement during the wavelet tree traversal. Constrained by the given budget, the traversal maintains the LOD as a cut through the multiresolution data hierarchy, and refines the blocks on the cut in a greedy manner.

The LOD selection and rendering works as follows: First, we initialize the priority queue with the data block of the lowest resolution, i.e., the root of the multiresolution data hierarchy. Then, we successively refine the block with the highest priority value in the queue until the budget is met. The refinement is performed by deleting the block b with the highest priority value, updating the importance values of b's eight child blocks, and inserting the child blocks into the queue. Finally, all the data blocks in the queue are sorted in front-to-back viewing order. These blocks are reconstructed, if necessary, and sent to texture hardware for rendering.

As we may anticipate reusing most of the reconstructed data blocks for subsequent frames due to the spatial locality and coherence exploited by the multiresolution data hierarchy, it is desirable to cache the data blocks that have already been reconstructed for better performance. The user can predefine a fixed amount of disk space and memory dedicated for the caching purpose. Upon requesting a data block for the rendering, we retrieve its data from the memory, provided the block is cached in the main memory. Otherwise, we need to load the data from the disk if the reconstructed data block is cached on disk. If it is neither cached in memory nor on disk, then we need to reconstruct the data block and load it into the main memory. When the system runs short of the available storage for caching the reconstructed blocks, our replacement scheme will swap out a data block that has been visited least often.

5 Results and Discussion

We experimented with our LOD selection and rendering algorithm on the VisWoman and RMI data sets, as listed in Table 1. The decision for the block size was a tradeoff between the cost of performing the wavelet transform for a single data block, and the rendering overhead for final image generation. For both data sets, the

data set (type)	VisWoman (short)	RMI (byte)
volume dimension	$512 \times 512 \times 1728$	$2048 \times 2048 \times 1920$
block dimension	$32 \times 32 \times 64$	$128 \times 128 \times 64$
volume/block size	864MB/128KB	7.5GB/1MB
# non-empty blocks	9446	10499
compression ratio	2.37:1	5.60:1

Table 1: The VisWoman and RMI data sets.

Haar wavelet transform with a lifting scheme was used to construct the data hierarchies. A *lossless* compression scheme was used with the threshold set to zero to compress the wavelet coefficients. We extended one voxel overlapping boundaries between neighboring blocks in each dimension when breaking the original volume data into blocks in order to produce seamless rendering. Both hierarchies have a tree depth of six. All tests were performed on a 3.0GHz Intel Xeon processor with 3GB main memory, and an nVidia Quadro FX 3400 graphics card with 256MB video memory.

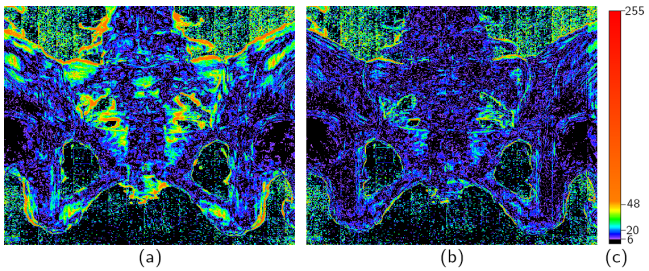

Figure 6: Objective image comparison in the CIELUV color space. (a) shows the difference between Figure 5 (b) and Figure 5 (c). (b) shows the difference between Figure 5 (d) and Figure 5 (c). The color map (c) maps ΔE to color.

Figure 5 shows the LOD rendering of the VisWoman data set using the traditional MSE-based metric and our image-based quality metric. The full-resolution reference image is put in between for subjective comparison. A transfer function was used to highlight the skeleton. Similar numbers of blocks were set for both cases for fair comparison. For the MSE-based metric, we calculated the multiresolution error using Equation 11, while ε_{ij} is the MSE of the scalar data values of blocks b_i and b_j. It can be observed that when we rendered the data in low resolution, the LOD selection using the image-based quality metric performs better than the MSE-based metric.

An objective comparison was also conducted to testify the superior performance of our image-based quality metric. We calculated the pixel-wise differences between the low resolution image and the reference image in the CIELUV color space. The difference

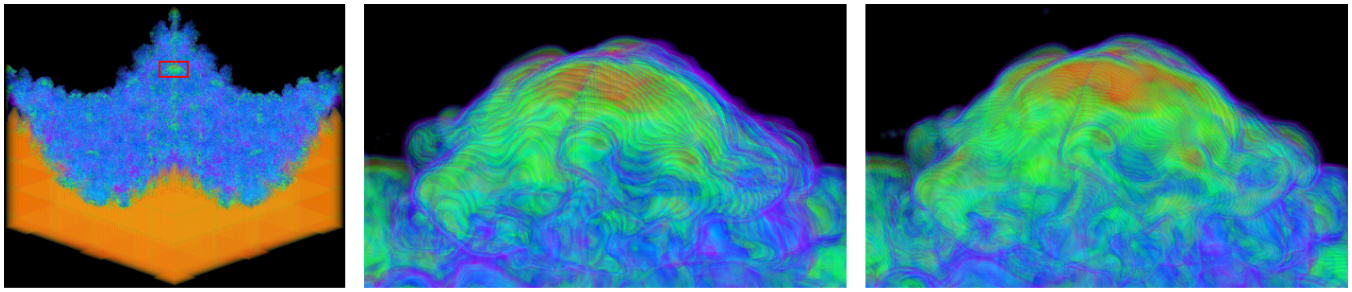

Figure 8: (a) shows an overview of the RMI data set. (b) (MSE-based, 52 blocks) and (c) (image-based, 51 blocks) show a zoom to the center of the data.

threshold $\Delta E \geq 6.0$ gives the noticeable pixel distortion [20]. Figure 6 shows the two difference images side by side. Clearly, the one with the MSE-based metric contains larger visual distortion. Another rendering example of the VisWoman data set is shown in Figure 7. Again, we can see that the image-based LOD selection performs better than the MSE-based one.

Figure 7: (a) shows a zoom to the upper skeleton of the VisWoman data set, rendered in full resolution. (b) (MSE-based, 58 blocks) and (c) (image-based, 55 blocks) show a closer zoom to the spine while rendered in low resolution.

Figure 8 shows the LOD selection and rendering of the RMI data set using the two metrics. We zoomed into a portion of the data and compared fine details after an overview. The image-based quality metric takes into account the multiresolution error and visibility of each data block, thus puts more refinement effort on the blocks that have larger visual contribution. Figure 9 shows the numbers of blocks rendered in each of the ten visibility levels for Figure 8 (b) and (c) respectively. As we can see, compared with the one with MSE, the image-based one selected more blocks with higher visibility. The images in Figure 8 as well as the chart in Figure 9 confirm the effectiveness of our image-based LOD selection algorithm.

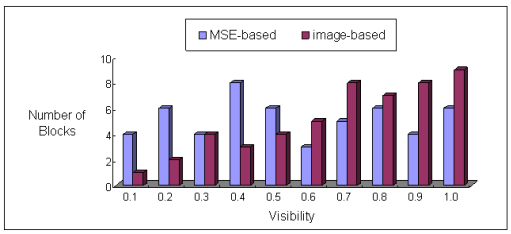

Figure 9: The numbers of blocks rendered in each of the ten visibility levels for Figure 8 (b) and (c) respectively.

We also experimented with our summary table scheme for updating the multiresolution errors. For the RMI data set, it took 44.1MB to store the summary tables and 13 seconds to update the multiresolution errors. For the VisWoman data set, the storage size was 9.22MB and the time for error update was 5 seconds. Compared with the original data sizes, the storage overhead was only 0.574% and 1.067% for the RMI and VisWoman data sets respectively. The summary table scheme proved very efficient in response to the transfer function changes with negligible storage overhead. A rendering example of the VisWoman data set is shown in Figure 10, where a different transfer function was used to highlight both the skin and the skeleton. We zoomed into the left foot and rendered it close to full resolution. Although the two methods generated closer results as we approached the full resolution, it can be seen that the MSE-based one still contains much noise coming from the 3D test bed surrounding the cadaver (the blocks corresponding to the test bed have larger MSEs yet less visual importance than the blocks corresponding to the foot.).

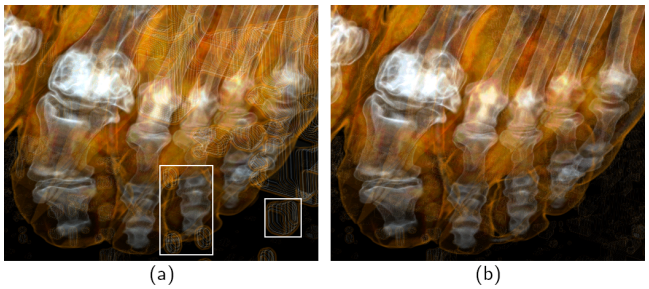

Figure 10: (a) (MSE-based, 97 blocks) and (b) (image-based, 88 blocks) show a zoom to the left foot of the VisWoman data set, rendered close to full resolution. White frames are drawn in (a) to indicate some of the differences.

Testing shows that the time to perform visibility estimation is not negligible. For instance, using the GPU reduction scheme with image resolution of 512^2, it took about 0.4 second to update the visibility of 10499 non-empty blocks for the RMI data set. If we perform such a test for every frame, then the frame rate would be limited to 2.5fps. To overcome this constraint, we incorporate the following two strategies to improve the rendering frame rates.

First, the number of block budget the user specifies is usually much smaller than the total number of blocks in the data hierarchy. For such a typical block budget and a given transfer function, normally a large portion of the updated visibility of blocks farther away from the viewpoint (more likely to be occluded from the blocks in front of them) never gets a chance to contribute to the current LOD decision. Actually, for the RMI data set, tests show that about 30 - 50% of the total number of blocks fall into this category. In view of this, we can only draw the front slices up to a certain percentage of the total number of slices, and update the corresponding visibility of blocks that are closer to the viewpoint. Any block whose visibility is not updated in this run uses whatever it has from the latest previous run. In this way, we can reduce the visibility estimation time to around 0.24 second for the RMI data set, if we only update 60% of the front slices and blocks.

Second, the visibility of the blocks only changes a little bit if the viewing does not change greatly. Therefore, if the angle between the current viewing direction and the latest one with the visibility

updated is less than a threshold angle θ, we do not update the visibility and use whatever we have from the latest run. Otherwise, we need to update again. Here, θ is a predefined small angle (initialized to 5 degrees in our test), and is adaptive to the zooming of the data during the rendering.

By reducing the load to perform each run of visibility estimation and the frequency of performing such estimations, we can reuse visibility computation and utilize frame to frame coherence, achieving much smoother rendering and better frame rates.

After applying the two strategies for improving the performance, we compared the timing of visibility estimation for software and hardware approaches. With output image resolution of 512^2, we tested the RMI data set for a wide variety of block budgets (from 10 to 10000), combined with different scaling factors (from 0.1 to 10.0) and rotations. The test gave the timing range of $[0.140, 0.274]$ for the hardware approach, and $[0.226, 1.009]$ for the software one, both in second. This result convinces us that the solution with the GPU is much faster and more stable than the CPU one.

6 Conclusion and Future Work

The focus of this work is to develop an image-based LOD selection algorithm for large volumetric data, and produce images of better quality compared with traditional data-based LOD selection algorithms, under similar block budgets. In this paper, we have presented an interactive LOD selection and rendering algorithm through the use of an image-based quality metric. Experimental results on large scientific and medical data sets show that our image-based LOD selection is superior to the MSE-based one.

Our approach is promising due to its generality and flexibility. The summary table scheme greatly alleviates the dependency of the error calculation on the transfer function, and thus allows one to update the errors within seconds whenever the transfer function changes. The GPU reduction scheme for visibility estimation is not limited to multiresolution volume rendering, and is readily applicable to other large volume visualization scenarios that capitalize on the visibility information. Moreover, the hierarchical data representation and the user-specified budget for rendering make our LOD selection scheme suitable for time-critical multiresolution volume rendering applications. Finally, one can have different definitions and thus different ways of measurement for the multiresolution error in Equation 4, which we would like to explore more. In the future, we also would like to extend our method for large-scale time-varying data visualization.

References

[1] I. Boada, I. Navazo, and R. Scopigno. Multiresolution Volume Visualization with a Texture-Based Octree. *The Visual Computer*, 17(3):185–197, 2001.

[2] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Trans. on Visualization & Computer Graphics*, 9(3):412–431, 2003.

[3] F. C. Crow. Summed-Area Tables for Texture Mapping. In *ACM SIGGRAPH '84*, pages 207–212, 1984.

[4] J. El-Sana, N. Sokolovsky, and C. T. Silva. Integrating Occlusion Culling with View-Dependent Rendering. In *IEEE Vis '01*, pages 371–575, 2001.

[5] D. Ellsworth, L. J. Chiang, and H. W. Shen. Accelerating Time-Varying Hardware Volume Rendering Using TSP Trees and Color-Based Error Metrics. In *IEEE VolVis '00*, pages 119–129, 2000.

[6] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles & Practice in C (2nd Edition)*. Addison Wesley, 1995.

[7] A. Gaddipati, R. Machiraju, and R. Yagel. Steering Image Generation with Wavelet Based Perceptual Metric. In *Eurographics '97*, 1997.

[8] J. Gao, J. Huang, H. W. Shen, and J. A. Kohl. Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization. In *IEEE Vis '03*, pages 341–348, 2003.

[9] M. H. Ghavamnia and X. D. Yang. Direct Rendering of Laplacian Pyramid Compressed Volume Data. In *IEEE Vis '95*, pages 192–199, 1995.

[10] A. S. Glassner. *Principle of Digital Image Synthesis, Volume 1*. Morgan Kaufmann, 1995.

[11] S. Guthe and W. Straßer. Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics*, 28(1):51–58, 2004.

[12] S. Guthe, M. Wand, J. Gonser, and W. Straßer. Interactive Rendering of Large Volume Data Sets. In *IEEE Vis '02*, pages 53–60, 2002.

[13] C. E. Jacobs, A. Finkelstein, and D. H. Salesin. Fast Multiresolution Image Querying. In *ACM SIGGRAPH '95*, pages 277–286, 1995.

[14] J. T. Klosowski and C. T. Silva. The Prioritized-Layered Projection Algorithm for Visible Set Estimation. *IEEE Trans. on Visualization & Computer Graphics*, 6(2):108–123, 2000.

[15] E. LaMar, B. Hamann, and K. I. Joy. Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *IEEE Vis '99*, pages 355–362, 1999.

[16] E. LaMar, B. Hamann, and K. I. Joy. Efficient Error Calculation for Multiresolution Texture-Based Volume Visualization. In *Hierarchical & Geometrical Methods in Scientific Visualization*, pages 51–62, 2003.

[17] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. In *ACM SIGGRAPH '91*, pages 285–288, 1991.

[18] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Trans. on Graphics*, 9(3):245–261, 1990.

[19] X. Li and H. W. Shen. Time-Critical Multiresolution Volume Rendering Using 3D Texture Mapping Hardware. In *IEEE VolVis '02*, pages 29–36, 2002.

[20] P. Ljung, C. Lundström, A. Ynnerman, and K. Museth. Transfer Function Based Adaptive Decompression for Volume Rendering of Large Medical Data Sets. In *IEEE VolVis '04*, pages 25–32, 2004.

[21] N. Max. Optical Models for Direct Volume Rendering. *IEEE Trans. on Visualization & Computer Graphics*, 1(2):99–108, 1995.

[22] M. Meißner, J. Huang, D. Bartz, K. Muller, and R. Crawfis. A Practical Evaluation of Popular Volume Rendering Algorithms. In *IEEE VolVis '00*, pages 81–90, 2000.

[23] S. Muraki. Approximation and Rendering of Volume Data Using Wavelet Transforms. In *IEEE Vis '92*, pages 21–28, 1992.

[24] P. Ning and L. Hesselink. Vector Quantization for Volume Rendering. In *IEEE VolVis '92*, pages 69–74, 1992.

[25] C. O'Sullivan, S. Howlett, Y. Morvan, R. McDonnell, and K. O'Conor. Perceptually Adaptive Graphics. Eurographics State of the Art Reports '04, 2004.

[26] T. Porter and T. Duff. Compositing Digital Images. In *ACM SIGGRAPH '84*, pages 253–259, 1984.

[27] N. Sahasrabudhe, J. E. West, R. Machiraju, and M. Janus. Structured Spatial Domain Image and Data Comparison Metrics. In *IEEE Vis '99*, pages 97–104, 1999.

[28] J. Schneider and R. Westermann. Compression Domain Volume Rendering. In *IEEE Vis '03*, pages 293–300, 2003.

[29] C. Wang, J. Gao, and H. W. Shen. Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing. In *Eurographics Parallel Graphics & Visualization '04*, pages 23–30, 2004.

[30] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Trans. on Image Processing*, 13(4):600–612, 2004.

[31] R. Westermann. A Multiresolution Framework for Volume Rendering. In *IEEE VolVis '94*, pages 51–58, 1994.

[32] J. Wilhelms and A. van Gelder. Multi-Dimensional Trees for Controlled Volume Rendering and Compression. In *IEEE VolVis '94*, pages 27–34, 1994.

[33] H. Zhou, M. Chen, and M. F. Webster. Comparative Evaluation of Visualization and Experimental Results Using Image Comparison Metrics. In *IEEE Vis '02*, pages 315–322, 2002.