Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

# Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing

Chaoli Wang, Jinzhu Gao and Han-Wei Shen

The Ohio State Univeristy
Columbus, Ohio 43210, USA
E-mail: {wangcha, gao, hwshen}@cis.ohio-state.edu

**Abstract**

*We present a new parallel multiresolution volume rendering algorithm for visualizing large data sets. Using the wavelet transform, the raw data is first converted into a multiresolution wavelet tree. To eliminate the parent-child data dependency for reconstruction and achieve load-balanced rendering, we design a novel algorithm to partition the tree and distribute the data along a hierarchical space-filling curve with error-guided bucketization. At run time, the wavelet tree is traversed according to the user-specified error tolerance, data blocks of different resolutions are decompressed and rendered to compose the final image in parallel. Experimental results showed that our algorithm can reduce the run-time communication cost to a minimum and ensure a well-balanced workload among processors when visualizing gigabytes of data with arbitrary error tolerances.*

Categories and Subject Descriptors (according to ACM CCS): E.4 [Coding and Information Theory]: Data compaction and compression; I.3.1 [Computer Graphics]: Parallel processing; 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

## 1. Introduction

An increasing number of scientific applications are now generating high resolution three-dimensional data sets on a regular basis. The sizes of those data sets are often so large that it is almost impossible to perform interactive data analysis using only a single PC or workstation. Take the time-dependent Richtmyer-Meshkov turbulence simulation [MCC*99] as an example, at each time step the simulation produced about 7.5 gigabytes of data defined on a $2048 \times 2048 \times 1920$ rectilinear grid. Not surprisingly, data of this scale can not be handled easily by a single machine with limited computational resources. A viable solution to address this challenge is to utilize a cluster of PCs to distribute the data and perform the computation and rendering in parallel.

As visualization is an iterative and exploratory process, rendering a lower resolution of data sometimes is sufficient for the purpose of obtaining an overview of the data before the user can query further details in selected regions. Given the physical limitation in the current generation of display devices, it is also not always desirable to render the entire data set at the finest resolution considering that the projection of such a large data set can be far beyond the highest screen resolution currently available. For this reason, many visualization algorithms provide the user with the capability of multiresolution rendering so that it is possible to perform interactive data navigation in an adaptive manner.

In this paper, we present a parallel algorithm for multiresolution volume rendering. Although researchers previously have proposed various techniques for multiresolution encoding and rendering of large scale volumes on a single graphics workstation [ZCK97, LHJ99, GWGS02], fewer studies were focused on designing parallel algorithms for such a purpose using PC clusters. Several issues need to be addressed in order to achieve efficiency and scalability when parallel multiresolution volume rendering is performed. One is the issue of designing an effective data distribution scheme that can minimize both space and run-time computation overheads for storing and reconstructing the multiresolution volumes. When hierarchical encoding schemes such as the

wavelet transform is used, the multiresolution data are often represented in the form of a hierarchical tree [GWGS02]. Obviously, the sheer size of the data prohibits the replication of the entire tree in every processor so it is necessary to partition and distribute the multiresolution hierarchy. Since there is often a great deal of dependency among the parent and child nodes in the data hierarchy, it is critical to design an efficient partitioning and distribution algorithm to minimize such dependency and thus reduce the run time interprocessor communication cost for data reconstruction. Another issue that needs to be addressed is load balancing. At run time, when the user specifies arbitrary error tolerance to visualize the volume, different spatial subvolumes will be reconstructed with various levels of detail, which could cause uneven rendering workloads among the processors. It is important to design an effective workload distribution scheme so that the rendering subtasks can be evenly distributed among the processors for any given error tolerance used to traverse the multiresolution data hierarchy. It is also crucial for the workload distribution algorithm to work hand-in-hand with the data distribution scheme so as to avoid expensive data redistribution dynamically at run time.

In our algorithm, we exploit the wavelet transform and convert the data into a hierarchical multiresolution representation, called a *wavelet tree*. To alleviate the long chains of parent-child node dependencies when reconstructing volumes of different resolutions, we partition the wavelet tree into *distribution units* which can eliminate the data dependency among processors and also limit the reconstruction cost. To avoid run-time data redistribution and balance the workload of volume rendering, we utilize a *hierarchical space-filling curve* to distribute the data and the rendering tasks, guided by a *hierarchical error metric*.

The remainder of the paper is organized as follows. First, we review related work in Section 2. From Section 3 to Section 6, we describe our parallel multiresolution volume rendering algorithm, including the construction of the wavelet tree with hierarchical error metric calculation, data distribution with error-guided bucketization, and the run-time parallel multiresolution volume rendering algorithm. Results on multiresolution rendering and load balancing among different processors are given in Section 7 and the paper is concluded in Section 8 with future work for our research.

## 2. Related Work

Having the ability to visualize data at different resolutions allows the user to identify features in different scales, and to balance image quality and computation speed. Along this direction, a number of techniques have been introduced to provide hierarchical data representation for volume data. Burt and Adelson [BA83] proposed the *Laplacian Pyramid* as a compact hierarchical image code. This technique was extended to 3D by Ghavamnia and Yang [GY95] and applied to volumetric data. Their Laplacian pyramid is constructed using a Guassian low-pass filter and encoded by uniform quantization. Voxel values are reconstructed at run time by traversing the pyramid bottom up. To reduce the high reconstruction overhead, they suggested a cache data structure. LaMar *et al.* [LHJ99] proposed an octree-based hierarchy for volume rendering where the octree nodes store volume blocks resampled to a fixed resolution and rendered using 3D texture hardware. A similar technique was introduced by Boada *et al.* [BNS01]. Their hierarchical texture memory representation and management policy benefits nearly homogeneous regions and regions of lower interest.

Wavelets are used to represent functions hierarchically, and have gained much popularity in several areas of computer graphics [SDS96]. Over the past decade, many wavelet transform and compression schemes have been applied to volumetric data. Muraki [Mur92, Mur93] introduced the idea of using the wavelet transform to obtain a unique shape description of an object, where 2D wavelet transform is extended to 3D and applied to eliminate wavelet coefficients of lower importance. The use of single 3D [IP98, KS99] or multiple 2D [Rod99] Haar wavelet transformations for large 3D volume data has been well studied, resulting in high compression ratios with fast random access of data at run time. More recently, Guthe *et al.* [GWGS02] presented a hierarchical wavelet representation for large volume data sets that supports interactive walkthrough on a single commodity PC. Only the levels of detail necessary for display are extracted and sent to texture hardware for viewing.

Parallel computing has been widely used in large volume visualization. Hansen and Hinker [HH92] proposed a parallel algorithm on SIMD machines to speed up isosurface extraction. Ellsiepen [Ell95] introduced a parallel implementation for unstructured isosurface extraction with a dynamical block distribution scheme. Crossno and Angel [CA97] gave an isosurface extraction algorithm using particle systems and its parallel implementation. An isosurface extraction algorithm in span space and the corresponding parallelization were described in [SHLJ96]. To speed up the volume rendering process, Ma *et al.* [MPHK94] proposed a parallel algorithm that distributes data evenly to the available computing resources and produces the final image using binary-swap compositing. Schulze and Lang [SL02] provided a parallelized version of perspective shear-warp volume rendering algorithm [LM94]. A scalable volume rendering technique was presented in [LMC02], utilizing lossy compression to render time-varying scalar data sets interactively. To further reduce the rendering time of large-scale data sets, several parallel visualization algorithms [HSC*00, GS01, GHSK03] with visibility culling were introduced to render only visible portion of a data set in parallel.

Balancing the workload among the processors is always a key issue in a parallel implementation. In [CDF*03], Campbell *et al.* showed a load-balanced technique using the space-filling curve [Sag94] traversal. In this method, the spatial lo-

cality preserved by a space-filling curve was utilized to balance the workload. Gao *et al.* [GHSK03] also showed that, even after visibility culling, the parallel volume rendering algorithm can still achieve well-balanced workload by distributing volume blocks to processors along a space-filling curve.

## 3. Algorithm Overview

Our parallel multiresolution volume rendering algorithm consists of two stages - preprocessing and run-time rendering. The purpose of the preprocessing stage is to create a multiresolution hierarchy for the input volume using the wavelet transform. It also partitions and distributes the wavelet multiresolution hierarchy to different processors. Based on the preprocessed information, our run-time algorithm performs multiresolution volume rendering in parallel, ensuring high performance and balanced workload distribution for any user-specified error tolerance.

In the preprocessing stage, we first construct a hierarchical wavelet tree and then compress the wavelet coefficients using a combination of run-length and Huffman encoding. Coupled with the construction of the wavelet tree, a *hierarchical error metric* is used to calculate the approximation error for each of the tree nodes, which will be used to control the tradeoff between image quality and rendering speed at run time. This error metric can be rapidly computed, and also guarantees that the error value of a parent node will be greater than or equal to those of its eight child nodes. The data blocks associated with the wavelet tree nodes are then distributed among different processors along a hierarchical space-filling curve with an error-guided bucketization scheme to ensure load balancing.

At run time, our parallel multiresolution volume rendering algorithm is performed according to a user-specified error tolerance. The wavelet tree is first traversed front to back to identify the nodes with varied resolutions that satisfy the error tolerance. Then, the wavelet-compressed data associated with those nodes are decompressed and the data blocks are reconstructed on the fly. Finally, the processors render the selected data blocks with various levels of detail in parallel. The final image is generated by compositing the partial images rendered at different processors.

In the following, we describe each stage of our algorithm in detail. We first present the multiresolution wavelet tree construction algorithm and the error metric. Then, we discuss our data distribution scheme for the purpose of minimizing the dependency among processors and ensuring run-time load balancing. Implementation details about our parallel volume rendering will follow.

## 4. Wavelet Tree Construction with Hierarchical Error Metric Calculation

Our hierarchical wavelet tree construction algorithm is similar to the methods described in [GWGS02, WS04], where a bottom-up blockwise wavelet transform and compression scheme is used. The algorithm starts with subdividing the original three-dimensional data into a sequence of blocks. We assume each raw volume block has $n$ voxels. Without loss of generality, we also assume $n = 2^i \times 2^j \times 2^k$, where $i$, $j$, $k$ are all integers and greater than zero. 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 then collected and grouped into a single block of $n$ voxels, which will become the low resolution data block associated with the parent node in the next level of the wavelet hierarchy. We recursively apply this 3D wavelet transform and subblock grouping process until the root of the tree is reached, where a single block of size $n$ is used to represent the entire volume. As we arrive at the root of the wavelet tree, since the root node has no parent, no 3D wavelet transform is performed. To save space and time for the wavelet tree construction, unnecessary wavelet transform computation could be avoided by checking the uniformity of the data block. If the data block is uniform, we can skip the 3D wavelet transform process and set the low-pass filtered subblock to that uniform value and all its corresponding wavelet coefficients to zero.

To reduce the size of the coefficients stored in the wavelet tree, the wavelet coefficients associated with a tree node resulting from the 3D wavelet transform will be compared against a user-provided threshold and set to zero if they are smaller than the threshold. These wavelet coefficients are then compressed using run-length encoding combined with a fixed Huffman encoder [GWGS02]. This bit-level run-length encoding scheme exhibits good compression ratio if many consecutive zero subsequences are present in the wavelet coefficient sequence and is very fast to decompress. The compressed bit stream is saved into an individual file.

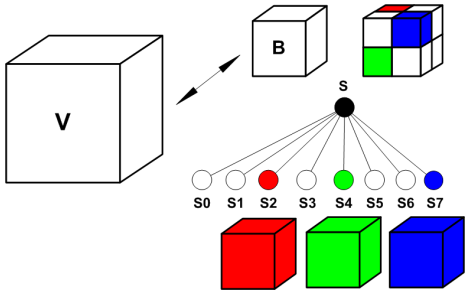



**Figure 1:** *Calculating the error metric of a wavelet tree node S. B is the low resolution data block associated with S, representing the raw data subvolume V. The three colored nodes and their associated data blocks are examples used to illustrate the data relationship of the parent node S and it child nodes $S_i$, where $0 \leq i \leq 7$.*

Coupled with the construction of the wavelet tree, an error value is calculated at every tree node. Our error metric is based on the mean square error (MSE) calculation. As shown in Figure 1, let us assume that the current wavelet tree node in question is $S$, the $i$th child node of $S$ is $S_i$, $i \in [0,7]$, and the data block of $n$ voxels associated with $S$ that approximates the original subvolume $V$ is $B$. One way to calculate the error metric is to compute the MSE between the low resolution data block $B$ and the corresponding raw data in subvolume $V$ using the following formula:

$$E = \frac{(\sum_{(x,y,z)\in V} (v(x,y,z) - f(x,y,z))^2)}{m}$$

where $v(x,y,z)$ is the original scalar data value at the location $(x,y,z)$ in $V$. $f(x,y,z)$ is the interpolated data value at its corresponding position in $B$. $m$ is the total number of voxels in $V$. The interpolation function for obtaining the approximated data value can be either nearest neighbors or linear. For any wavelet tree leaf node, we define $E = 0$.

The main drawback of calculating the error metric this way is that when the underlying data set is large, it can be very slow to perform the error computation. The MSE calculation will become progressively more expensive as we traverse toward the wavelet tree root since the size of the volume covered by a tree node will increase proportionally. Furthermore, a large I/O overhead is involved because the computation requires retrieving the raw data as well as the approximated data.

To overcome these problems, we propose a much faster way to calculate the error metric which considers the MSE between the data in a parent node and the data in its eight immediate child nodes, taking the maximum error value of the child nodes into account. We compute the error as follows:

$$E = \frac{\sum_{i=0}^{7} (\sum_{(x,y,z)\in B} (b_i(x,y,z) - f(x,y,z))^2)}{8n} + maxE$$

where $b_i(x,y,z)$ is the data value at location $(x,y,z)$ in the data block associated with $S_i$. $f(x,y,z)$ is the interpolated data value at its corresponding position in $B$. $maxE$ is the maximum error of $S_i$, where $0 \le i \le 7$. Again, the interpolation function for getting the approximated data value can be either nearest neighbors or linear. Essentially, the error $E$ of a parent node $S$ is calculated by adding $maxE$ to the MSE between eight data blocks associated with child nodes $S_i$ and their corresponding low-pass filtered data in $B$. For any wavelet tree leaf node, we define $E = 0$.

A nice feature of this error metric is that it guarantees that the error value of any parent node is greater than or equal to those of its corresponding eight child nodes. Our design of this *hierarchical* error metric is useful for flexible error control when we perform the wavelet tree traversal during the actual rendering.

## 5. Hierarchical Data Distribution with Error-Guided Bucketization

For large scale data sets, the resulting wavelet hierarchy needs to be partitioned and distributed among the processors since it is impractical to replicate the data everywhere. To ensure the scalability of the parallel algorithm, it is important that the partitioning result will minimize the dependency among the processors and ensure a balanced workload. In this section, we describe our data distribution and load balancing algorithm in detail.

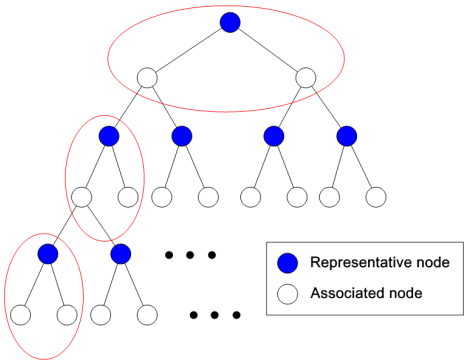



**Figure 2:** *Only the nodes at every k levels starting from the root (drawn in blue) store the data blocks. The red ellipsis show examples of the distribution units. In the figure, $k = 2$ and $h = 6$. A binary tree rather than an octree tree is drawn here for illustration purpose only.*

One of the primary issues to be addressed when designing the data distribution scheme is to minimize the dependency among the processors. In the wavelet tree structure mentioned above, there exist long chains of parent-child node dependencies - a node needs to recursively request the low-pass filtered subblock from its parent node in order to reconstruct its own data. When nodes with such dependencies are assigned to different processors, expensive communications at run time become inevitable. To eliminating such dependency among processors, we design the following storage strategy to arrange the multiresolution data blocks in the wavelet tree. Instead of having the leaf and intermediate nodes store wavelet coefficients, and only the root node store the low resolution data block, we reconstruct and store low resolution data blocks for nodes at every $k$ levels starting from the root, where $k < h$, and $h$ is the height of the wavelet tree. (In practice, $h$ may not be an exact multiple of $k$ and this can be easily handled.) We call a node that stores the reconstructed data block a *representative node*, while a node that stores only wavelet coefficients an *associated node*. By default, the root of the wavelet tree is a rep-

resentative node and all the leaf nodes of the tree are associated nodes. Figure 2 shows an example of such schemes where $k = 2$ and $h = 6$. It is clear that data reconstruction only needs to be performed for the associated nodes by recursively requesting their parent nodes up to the closest representative node, where the low resolution data has already been reconstructed. We define a *distribution unit* as the data at a representative node along with the wavelet coefficients at all its descendent nodes which depend on the representative node. This definition implies that there must be one and only one representative node in a distribution unit and all the nodes in one distribution unit are independent of nodes in any other distribution units. We use the distribution units to form a *partition* of the wavelet tree, and a distribution unit is used as the *minimum* unit to be assigned to a processor. Since there is no data dependency among distribution units during wavelet reconstruction, we are able to eliminate the dependency among processors at run time.

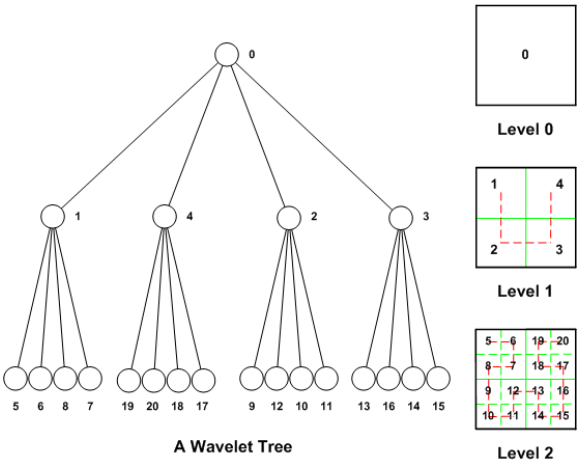



**Figure 3:** *A simplified 2D example of data distribution along the hierarchical space-filling curve. All the wavelet tree nodes are traversed level by level in a breadth-first search manner. The numbers associated with the tree nodes indicate the traversal order given by the space-filling curve. A popular space-filling curve, the Hilbert curve, is used in this example.*

An optimal data distribution scheme should ensure that all the processors have an equal amount of rendering workload at run time. However, when multiresolution rendering is performed, different data resolutions, and thus different rendering workload, will be chosen to approximate the local regions. This makes the workload distribution task more complicated. In the following, we describe a static load distribution scheme to solve the load balancing problem.

In general, a volumetric data set usually exhibits strong spatial coherence. Given an error tolerance, if a particular data resolution is chosen for a subvolume, it is more likely that a similar resolution will also be used for the neighboring subvolumes. In our multiresolution algorithm, this means if a block at a certain level is selected to be rendered, it is most likely that its neighboring blocks in the same tree level will also be rendered. Thus, if neighboring data blocks at a similar resolution are evenly distributed to different processors, each processor will receive approximately the same rendering workload in that local neighborhood. Based on this idea, a space-filling curve [Sag94] is utilized in our data distribution scheme to assign the distribution units to different processors. The space-filling curve is used for its ability to preserve spatial locality, i.e., the traversal path along a space-filling curve always visits the adjacent blocks before it leaves the local neighborhood. The hierarchical property of a space-filling curve also makes it suitable to be applied to a hierarchical algorithm. In Figure 3, we give a simplified 2D example of a wavelet tree and its corresponding space-filling curve at each level. The numbers in the figure show the traversal order along the hierarchal space-filling curve.

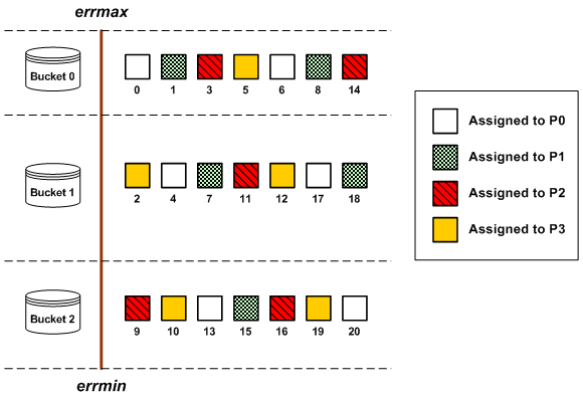



**Figure 4:** *An example of data distribution along the hierarchical space-filling curve with error-guided bucketization. The numbers associated with the distribution units shown in the figure indicate the traversal order given by the space-filling curve. In this example, a total of 21 distribution units with three different resolutions are distributed among four processors.*

To ensure load balancing at run time under different error tolerances, data blocks with similar error values should be distributed to different processors since they are most likely to be selected together for rendering. To achieve this, in addition to the hierarchal space-filling curve traversal, we include an error-guided bucketization mechanism in our data distribution scheme. As illustrated in Figure 4, our algorithm works as follows: The whole error range $[errmin, errmax]$ is first partitioned into several error intervals, where *errmin* and *errmax* are the minimum and maximum error values of the representative nodes from all the distribution units. Then, we traverse along a hierarchical space-filling curve, where every distribution unit encountered is sorted, according to

the traversal order, into a bucket associated with an error interval when the error value of its representative node falls into that interval range. The intervals of the buckets will be adjusted so that each bucket holds similar number of distribution units. Finally, all sorted distribution units in each of the buckets are distributed among processors in a round-robin manner.

Utilizing our hierarchical error-guided data distribution scheme, neighboring distribution units with similar errors will be distributed to different processors. As demonstrated in Section 7, our error-guided hierarchical data distribution scheme can achieve well-balanced workload among processors.

## 6. Parallel Multiresolution Volume Rendering

Before rendering each frame, the wavelet tree is traversed if the viewing parameter or the error tolerance has been changed. This could be done either by the host processor and then broadcasting the traversal result to all the other processors, or by all processors simultaneously (each processor only needs to have a copy of the wavelet tree skeleton with error at each node regardless of whether it actually has been assigned the data block or not), obviating the communication among the processors. Our error-guided tree traversal algorithm allows the user to specify an error tolerance as the stopping criterion so that regions having smaller errors can be rendered at lower resolutions. The nodes in the wavelet tree are recursively visited in the front-to-back order according to the viewing direction and a series of subvolumes with different resolutions that satisfy the error tolerance are identified. If the data blocks associated with those selected subvolumes have not been reconstructed, we need to perform reconstruction before the actual rendering begins.

A data block is reconstructed as follows: the low-pass filtered subblock of size $n/8$ is first retrieved from its parent node. This may entail a sequence of recursive requests of the low resolution data blocks associated with its ancestor nodes. Reconstructions will be performed in those nodes if necessary. Our wavelet tree partition and data distribution scheme ensures that such data dependencies could only exist within a distribution unit. This will reduce the overall reconstruction time since the cost of retrieving the low-pass filtered subblock is bounded by the height of the subtree corresponding to a distribution unit, which is usually a small number, two or three in our experiments. The high frequency wavelet coefficients of size $7n/8$ are obtained by decoding the corresponding bit stream. Finally, we group the low-pass filtered subblock and the wavelet coefficients and then apply an inverse 3D wavelet transform to reconstruct the data block.

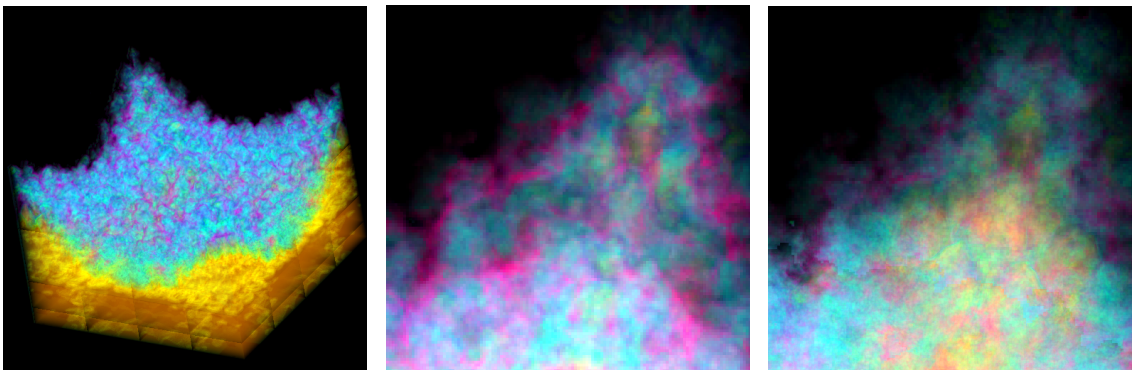
During the actual rendering, each processor only renders the data blocks preassigned to it during the data distribution stage, so there is no expensive data redistribution between processors. The screen projection of the entire volume's bounding box is partitioned into smaller tiles with the same size, where the number of the tiles equals the number of processors. Each processor is assigned one tile and is responsible for the composition of the final image for that tile. Each time a processor finishes rendering one data block, the resulting partial image is sent to those processors whose tiles overlap with the block's screen projection. After rendering all the data blocks, the partial images received at each processor are composited together to generate the final image for its assigned tile. Finally, the host processor collects the partial image tiles and creates the final image.

## 7. Results

In this section, we present the experimental results of our parallel multiresolution volume rendering algorithm running on a PC cluster that consists of 32 1.53GHz AMD Athlon 1800MP processors connected by Myrinet. The test data set was the 7.5GB $2048 \times 2048 \times 1920$ Richtmyer-Meshkov Instability (RMI) data set from Lawrence Livermore National Laboratory.

The dimensions of the leaf node blocks in our wavelet tree were set to be $128 \times 128 \times 64$, or 1MB in the total size. This is a tradeoff between the cost of performing the wavelet transform for a single data block, and the rendering and communication overheads for final image generation. Since each voxel value is represented using a single byte, Haar wavelet transform with a lifting scheme was used to construct the data hierarchy for simplicity and efficiency reasons. Lossless compression scheme was used with the threshold set to zero. We considered one voxel overlapping boundaries between neighboring blocks in each dimension when loading data from original brick data files in order to produce correct rendering results. The wavelet tree we constructed has a depth of six with 10,499 non-empty nodes. We chose the EVERY2 scheme (storing the reconstructed data blocks at every two levels of the wavelet tree) to partition the tree and form the distribution units. The construction of the wavelet tree was performed on a 2.0GHz Intel Pentium 4 processor with 768MB main memory. It took about an hour to complete and the compressed data size was 2.65GB. The data associated with the wavelet tree nodes were distributed using the hierarchical data distribution scheme with error-guided bucketization described in Section 5. The space-filling curve used in our implementation was the Hilbert curve.

Figure 5 shows several results with different levels of detail for the RMI data set rendered using software raycasting. When the error tolerance became higher, data of lower resolutions were used, which resulted in a smaller number of blocks being rendered. It can be seen that, although more delicate details of the data are revealed when lowering the error tolerance, images of reasonable quality can still be obtained at lower resolutions. The use of wavelet-based compression also allowed us to produce images of good visual quality with much smaller space overhead.



(a) A rendering of the data set (b) $E = 40,000$, 21 blocks rendered (c) $E = 30,000$, 120 blocks rendered

**Figure 5:** *Multiresolution rendering of the RMI data set. The resolution of the output images is 512 × 512. Image (a) shows a rendering of the data. Images (b) and (c) were zoomed-in views using different error tolerances with the same viewing setting. As can be seen, the lower the error tolerance was, the more delicate details of the data were revealed.*

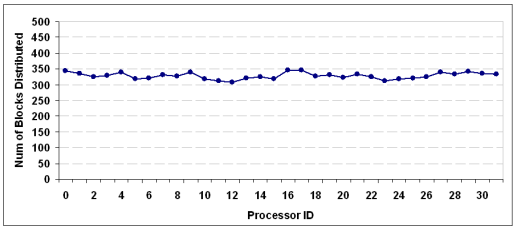



**Figure 6:** *The number of data blocks distributed to each of the 32 processors for the EVERY2 scheme.*

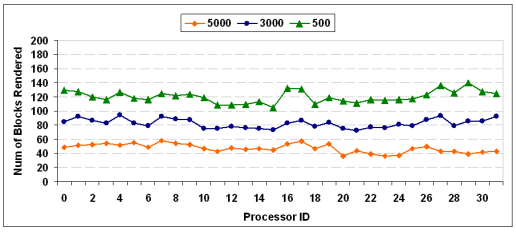



**Figure 7:** *The number of data blocks rendered at each of the 32 processors for the EVERY2 scheme with three different error tolerances. A total of 1,510, 2,640, and 3,850 blocks were rendered for error tolerances of 5,000, 3,000, and 500 respectively.*

The error-guided hierarchical data distribution allowed our parallel multiresolution volume rendering algorithm to effectively balance the workload. Figure 6 shows the number of data blocks distributed to each of the 32 processors. Figure 7 shows the numbers of data blocks rendered at each of the 32 processors when three different error tolerances were used. Since the processors rendered approximately an equal number of blocks, it can be seen that good load-balancing was achieved. The well-balanced workload implies that our parallel algorithm is highly scalable. When the error tolerance was set to 3,000, it took 48.74 seconds to render 2,640 $128 \times 128 \times 64$ blocks using 32 processors, which included the time to perform wavelet reconstruction and disk I/O (26.35sec), software raycasting (22.36sec), and image composition (0.03sec). Our algorithm can achieve approximately 97.6%, 96.8% and 87.3% parallel CPU utilization for 8, 16 and 32 processors respectively.

## 8. Conclusion and Future Work

We have presented an efficient parallel multiresolution volume rendering algorithm. A multiresolution wavelet tree is used to allow for interactive analysis of large data and flexible run-time tradeoff between image quality and rendering speed. To ensure the algorithm's scalability, we propose a unique tree partitioning and distribution algorithm and utilize a hierarchical space-filling curve with error-guided bucketization scheme to eliminate the parent-child node wavelet reconstruction dependencies, balance the rendering workload, and reduce the run-time communication cost to a minimum. The experimental results demonstrate the effectiveness and utility of our parallel algorithm. Future work includes utilizing graphics hardware to perform wavelet reconstruction and rendering, and extending our parallel multiresolution volume rendering algorithm to large scale time-varying data visualization.

### Acknowledgements

The work was supported by NSF ITR grant ACI-0325934, DOE Early Career Principal Investigator award DE-FG02-03ER25572, and NSF Career Award CCF-0346883. The RMI data set was provided by Mark Duchaineau at Lawrence Livermore National Laboratory. Special thanks to Don Stredney and Dennis Sessanna from Ohio Supercomputer Center for providing the test environment.

## References


[BA83] BURT P. J., ADELSON E. H.: The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications 31*, 4 (1983), 532–540. 2

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution Volume Visualization with a Texture-Based Octree. *The Visual Computer 17*, 3 (2001), 185–197. 2

[CA97] CROSSNO P., ANGEL E.: Isosurface Extraction Using Particle Systems. In *Proceedings of IEEE Visualization '97* (1997), pp. 495–498. 2

[CDF*03] CAMPBELL P. C., DEVINE K. D., FLAHERTY J. E., GERVASIO L. G., TERESCO J. D.: *Dynamic Octree Load Balancing Using Space-Filling Curves*. Tech. Rep. CS-03-01, Williams College Department of Computer Science, 2003. 2

[Ell95] ELLSIEPEN P.: Parallel Isosurfacing in Large Unstructed Datasets. In *Visualization in Scientifc Computing* (1995), pp. 9–23. 2

[GHSK03] GAO J., HUANG J., SHEN H. W., KOHL J. A.: Visibility Culling Using Plenoptic Opacity Functions for Large Data Visualization. In *Proceedings of IEEE Visualization '03* (2003), pp. 341–348. 2, 3

[GS01] GAO J., SHEN H. W.: Parallel View-Dependent Isosurface Extraction Using Multi-Pass Occlusion Culling. In *Proceedings of IEEE Symposium in Parallel and Large Data Visualization and Graphics '01* (2001), pp. 67–74. 2

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive Rendering of Large Volume Data Sets. In *Proceedings of IEEE Visualization '02* (2002), pp. 53–60. 1, 2, 3

[GY95] GHAVAMNIA M. H., YANG X. D.: Direct Rendering of Laplacian Pyramid Compressed Volume Data. In *Proceedings of IEEE Visualization '95* (1995), pp. 192–199. 2

[HH92] HANSEN C., HINKER P.: Massively Parallel Isosurface Extraction. In *Proceedings of IEEE Visualization '92* (1992), pp. 189–195. 2

[HSC*00] HUANG J., SHAREEF N., CRAWFIS R., SADAYAPPAN P., MUELLER K.: A Parallel Splatting Algorithm with Occlusion Culling. In *Proceedings of Eurographics Workshop on Parallel Graphics and Visualization '00* (2000), pp. 125–132. 2

[IP98] IHM I., PARK S.: Wavelet-Based 3D Compression Scheme for Very Large Volume Data. In *Proceedings of Graphics Interface '98* (1998), pp. 107–116. 2

[KS99] KIM T. Y., SHIN Y. G.: An Efficient Wavelet-Based Compression Method for Volume Rendering. In *Proceedings of Pacific Graphics '99* (1999), pp. 147–157. 2

[LHJ99] LAMAR E., HAMANN B., JOY K. I.: Multiresolution Techniques for Interactive Texture-Based Volume Visualization. In *Proceedings of IEEE Visualization '99* (1999), pp. 355–362. 1, 2

[LM94] LACROUTE P., MARC L.: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proceedings of ACM SIGGRAPH '94* (1994), pp. 451–458. 2

[LMC02] LUM E., MA K. L., CLYNE J.: A Hardware-Assisted Scalable Solution for Interactive Volume Rendering of Time-Varying Data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 286–301. 2

[MCC*99] MIRIN A. A., COHEN R. H., CURTIS B. C., DANNEVIK W. P., DIMITS A. M., DUCHAINEAU M. A., ELIASON D. E., SCHIKORE D. R., ANDERSON S. E., PORTER D. H., WOODWARD P. R., SHIEH L. J., WHITE S. W.: Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System. In *Proceedings of ACM/IEEE Supercomputing Conference '99* (1999). 1

[MPHK94] MA K. L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications 14*, 4 (1994), 59–68. 2

[Mur92] MURAKI S.: Approximation and Rendering of Volume Data Using Wavelet Transforms. In *Proceedings of IEEE Visualization '92* (1992), pp. 21–28. 2

[Mur93] MURAKI S.: Volume Data and Wavelet Transforms. *IEEE Computer Graphics and Applications 13*, 4 (1993), 50–56. 2

[Rod99] RODLER F. F.: Wavelet-Based 3D Compression with Fast Random Access for Very Large Volume Data. In *Proceedings of Pacific Graphics '99* (1999), pp. 108–117. 2

[Sag94] SAGAN H.: *Space-Filling Curves*. Springer-Verlag, New York, 1994. 2, 5

[SDS96] STOLLNITZ E. J., DEROSE T. D., SALESIN D. H.: *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, 1996. 2

[SHLJ96] SHEN H. W., HANSEN C. D., LIVNAT Y., JOHNSON C. R.: Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proceedings of IEEE Visualization '96* (1996), pp. 287–294. 2

[SL02] SCHULZE P., LANG U.: The Parallelization of the Perspective Shear-Warp Volume Rendering Algorithm. In *Proceedings of Eurographics Workshop on Parallel Graphics and Visualization '02* (2002), pp. 61–69. 2

[WS04] WANG C., SHEN H. W.: *A Framework for Rendering Large Time-Varying Data Using Wavelet-Based Time-Space Partitioning (WTSP) Tree*. Tech. Rep. OSU-CISRC-1/04-TR05, Department of Computer and Information Science, The Ohio State University, January 2004. 3

[ZCK97] ZHOU Y., CHEN B., KAUFMAN A.: Multiresolution Tetrahedral Framework for Visualizing Regular Volume Data. In *Proceedings of IEEE Visualization '97* (1997), pp. 135–142. 1