Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

A Multiresolution Volume Rendering Framework for Large-Scale Time-Varying Data Visualization

Chaoli Wang[†1], Jinzhu Gao[‡2], Liya Li[§1] and Han-Wei Shen[¶1]

[1] The Ohio State University, Columbus, OH 43210, USA
[2] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

Abstract

We present a new parallel multiresolution volume rendering framework for large-scale time-varying data visualization using the wavelet-based time-space partitioning (WTSP) tree. Utilizing the wavelet transform, a large-scale time-varying data set is converted into a space-time multiresolution data hierarchy, and is stored in a time-space partitioning (TSP) tree. To eliminate the parent-child data dependency for reconstruction and achieve load-balanced rendering, we design an algorithm to partition the WTSP tree and distribute the wavelet-compressed data along hierarchical space-filling curves with error-guided bucketization. At run time, the WTSP tree is traversed according to the user-specified time step and tolerances of both spatial and temporal errors. Data blocks of different spatio-temporal resolutions are reconstructed and rendered to compose the final image in parallel. We demonstrate 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 time-varying data on a PC cluster.

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 GenerationViewing algorithms; I.3.6 [Computer Graphics]: Methodology and TechniquesGraphics data structures and data types

1. Introduction

While direct volume rendering techniques using 3D texture mapping hardware have made it possible to visualize static three-dimensional or small time-varying data sets at interactive frame rates, the challenge is to manage and render very large-scale data sets. Nowadays, a complex scientific simulation can generate output with hundreds or thousands of time steps, and each time step can contain millions or billions of voxels. While it is not unusual now for a simulation to produce terabytes or petabytes of data, the available texture memory in the state-of-the-art high-end graphics hardware is limited to only several hundred megabytes. This great disparity makes it very difficult to develop visualization systems that can scale adequately. As the speed of processors and the size of disk and memory continue to increase, the size of data sets will likely increase at an even higher rate. As a result, the gap between the data size and our ability to perform interactive visualization will widen in the foreseeable future.

A viable solution to address this discrepancy is to reduce the actual amount of data sent to the rendering pipeline. For instance, to give the user a quick overview of the data, it is useful to render a large time-varying data set at lower spatial and/or temporal resolutions. As the user zooms into the data and requests further details in local regions or time intervals of interest, different portions of the data can be retrieved and rendered at their higher resolutions on demand. To support this kind of "*overview first, zoom and filter, and then details-on-demand*" data exploration paradigm [Shn96], it is crucial to provide an efficient spatio-temporal multiresolution

† wangcha@cse.ohio-state.edu
‡ gaoj@ornl.gov
§ lil@cse.ohio-state.edu
¶ hwshen@cse.ohio-state.edu

data management and rendering framework, in which various amount of data at different spatio-temporal resolutions can be extracted from the multiresolution data hierarchy and used for rendering.

Previously, researchers have introduced a number of techniques for encoding and rendering of large-scale static volumes [ZCK97, LHJ99, GWGS02] and time-varying data [SCM99, GS01, SBS02] on a single PC, but fewer studies have focused on designing multiresolution data management and rendering algorithms for large-scale time-varying data in parallel computing environment. In this paper, we present a framework of managing and visualizing large-scale time-varying data sets for multiresolution volume rendering over a PC cluster. In our algorithm, a data structure, called a *wavelet-based time-space partitioning (WTSP) tree*, is utilized to convert the time-varying data into a wavelet-compressed multiresolution data representation. To deploy the multiresolution framework over a PC cluster, we introduce an algorithm to partition the WTSP tree into *distribution units*, and distribute the partitioned data along *hierarchical space-filling curves* among different processors, guided by a *hierarchical spatial and temporal error metric*. Our algorithm can eliminate the data dependency among processors and balance the workload of volume rendering.

The remainder of the paper is organized as follows: First, we review related work in Section 2. In Section 3, we describe our multiresolution volume rendering framework in detail, including multiresolution data representation using the WTSP tree, WTSP tree partition and data distribution, and run-time parallel volume rendering. Results on multiresolution rendering and load balancing over a PC cluster are given in Section 4. The paper is concluded in Section 5 with future work for our research.

2. Related Work

In this section, we give a brief review of related work in the areas of multiresolution data representation, wavelet transform and compression, and parallel volume rendering.

Multiresolution Data Representation: Having the capability of visualizing data at different resolutions allows the user to identify features in different scales, and to balance image quality and computation speed. A number of techniques have been introduced to provide hierarchical data representation and indexing schemes for three-dimensional volumetric data [BA83, Wes94, GY95, LHJ99, BNS01, PF01]. For time-varying data hierarchical representation, Finkelstein *et al.* [FJS96] proposed *multiresolution video*, a representation for time-varying image data that allows for varying spatial and temporal resolutions adaptive to the video sequence. An approach for dealing with large-scale time-varying fields was described in [SCM99, ECS00]. The data structure, called the *time-space partitioning (TSP) tree*, encodes the spatial and temporal coherence of the data. Linsen *et al.* [LPD*02] presented a four-dimensional multiresolution approach for time-varying volume data that supports a hierarchy with spatial and temporal scalability. In their scheme, temporal and spatial dimensions are treated equally in a single hierarchical framework.

Wavelet Transform and Compression: Over the past decade, many wavelet-based techniques have been applied to compress, manage, and render three-dimensional volumetric data [Mur92, Mur93, IP98, KS99, Rod99] and RGB image data [BIP01], resulting in high compression ratios with fast random access of data at run time. The wavelet analysis has been applied for feature detection and compression-domain volume rendering [Wes94, Wes95]. More recently, Guthe *et al.* [GWGS02] presented a hierarchical wavelet representation for large volume data sets that supports interactive walkthroughs on a single PC. To visualize time-varying data sets, Guthe and Straßer [GS01] introduced an algorithm that uses the wavelet transform to encode each spatial volume, and then applies a windowed motion compensation strategy to match the volume blocks in adjacent time steps. Sohn *et al.* [SBS02] described a compression scheme where the wavelet transform is used to create intra-coded volumes and the difference encoding is applied to compress the time sequence. While all these methods can perform efficient data compression and rendering, their goals are not for supporting flexible spatio-temporal multiresolution data browsing.

Parallel Volume Rendering: Parallel computing has been widely used in large data visualization to accelerate volume rendering. 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. Kniss *et al.* [KMM*01] developed the *TRex* system for large-scale time-varying data visualization. Near-interactive frame rates were achieved by utilizing parallel graphic hardware in combination with software-based compositing and high performance I/O. A scalable volume rendering technique was presented in [LMC02], where a transform encoding and color table animation scheme was used to render time-varying scalar data sets interactively. Wang *et al.* [WGS04] presented a parallel multiresolution volume rendering algorithm for large three-dimensional volumetric data, where a well-balanced workload among processors was achieved by partitioning the wavelet tree, and distributing the data along hierarchical space-filling curves with an error-guided bucketization scheme. In this paper, we extend their ideas for large-scale time-varying data visualization.

3. The Algorithm

When the size of a large-scale time-varying data set is larger than what is tractable by rendering systems, it becomes crucial to manage the data through a framework for high-performance multiresolution volume rendering. In response, we devise a multiresolution spatio-temporal hierarchy that encodes the input time-varying data set. The hierarchy is based upon the wavelet transform, where the data set is stored in a data structure called the WTSP tree. The

WTSP tree exploits both spatial and temporal locality and coherence of the underlying time-varying data, thus allowing flexible spatio-temporal level-of-detail data selection and retrieval at run time. A hierarchical spatial and temporal error metric is used to calculate the approximate spatial and temporal errors for each of the tree nodes. This metric is used to control the run-time tradeoff between image quality and rendering speed. To alleviate long chains of parent-child node dependencies for data reconstruction, we introduce an algorithm to store the reconstructed data at nodes in selective WTSP tree levels, which effectively reduces the overall data reconstruction cost at run time.

When the WTSP tree is used in conjunction with parallel volume rendering, in order to eliminate the data dependency among processors for the wavelet reconstruction, and minimize the communication cost, we design an algorithm which partitions the WTSP tree into distribution units. Our algorithm then distributes these units to different processors. At run time, the WTSP tree is traversed according to the user-specified time step and tolerances of both spatial and temporal errors. Data blocks of different spatio-temporal resolutions are reconstructed and rendered to compose the final image in parallel. Our tree partition and data distribution algorithm ensures a well-balanced reconstruction and rendering workload among processors for any user-specified time sequences and error tolerances.

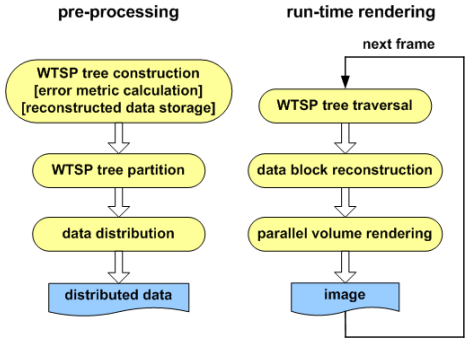

Figure 1: *The algorithm flow for large-scale time-varying data visualization.*

Figure 1 illustrates the flow of our algorithm for large-scale time-varying data visualization. In the following, we describe each stage of the algorithm in detail.

3.1. The Wavelet-Based Time-Space Partitioning Tree

Originating from the TSP tree [SCM99], the *wavelet-based time-space partitioning (WTSP) tree* [WS04] is a space-time hierarchical data structure used to organize multiresolution time-varying volume data. To construct the WTSP tree, a blockwise two-stage wavelet transform and compression process is performed, as illustrated in Figure 2. The first stage is to build a spatial hierarchy in the form of an octree (similar to a *wavelet tree* [GWGS02]) for each time step, where each node in the tree represents a subvolume with a certain spatial resolution at that particular time step. As shown in Figure 2 (a), the volumetric data for one time step is subdivided into a sequence of data blocks/subvolumes of the same size (assuming each has n voxels). A method similar to [GWGS02] is used to perform a 3D wavelet transform for each of the volume blocks. This will produce a low-pass filtered subblock of size $n/8$ and wavelet coefficients of size $7n/8$. The wavelet coefficients are compared against a user-specified threshold and clamped to zero if they are smaller than the threshold. The low-pass filtered subblocks from eight neighboring subvolumes are collected and grouped into a single block of n voxels, which becomes the parent node of the eight subvolumes in the spatial octree hierarchy (note that the wavelet coefficients are kept at the child nodes, while the low-pass filtered subblocks are passed up to the parent). We recursively apply this 3D wavelet transform and subblock grouping process until the root of the octree is reached, where a single block of size n is used to represent the entire volume. We repeat this process and construct one multiresolution spatial octree hierarchy for every time step.

In the second stage, to create the temporal hierarchy from the octrees of all time steps, we apply 1D wavelet transforms to the wavelet coefficients associated with octree nodes having the same spatial location and resolution across the time sequence. Using Haar wavelets, this will produce a binary time tree similar to the *error tree* algorithm [MVW98], as illustrated in Figure 2 (b). The wavelet coefficients resulting from 1D wavelet transforms 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. For the root nodes of the octrees, the temporal hierarchy is built from the low resolution data blocks rather than wavelet coefficients. As a result, the 1D wavelet transform process merges all the spatial octrees across time into a single unified spatio-temporal hierarchical data structure. In essence, the WTSP tree is an octree (spatial hierarchy) of binary trees (temporal hierarchy). There is only one octree skeleton, and at each octree node, there is a binary time tree. Each time tree spans the entire time sequence and combines data from multiple octrees.

We can save space and time for the WTSP tree construction by performing several additional checks to avoid unnecessary wavelet transform computation. First, if the data block is uniform, we can skip the 3D wavelet transform process and set the low-pass filtered subblock to the uniform value and all its corresponding wavelet coefficients to zero. Second, if the wavelet coefficients of the corresponding octree nodes in the time sequence are all zero, we can skip the 1D wavelet transform process.

Using Haar wavelets, the wavelet coefficients associated

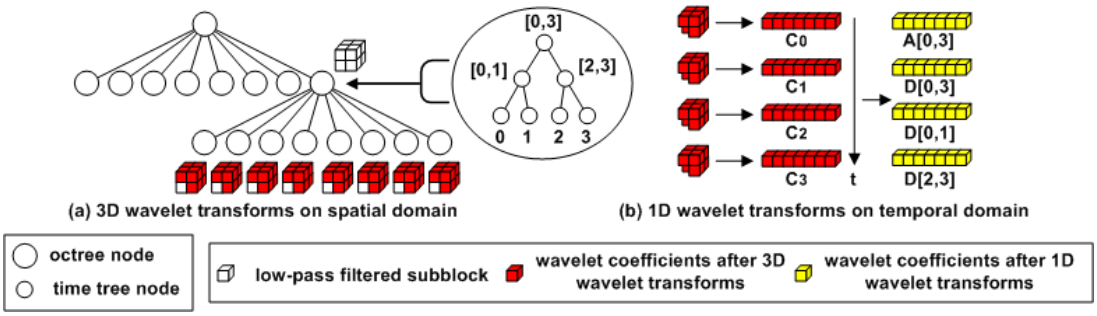

Figure 2: *The conceptual diagram of constructing the WTSP tree. In this figure, the time-varying data has four time steps. Ct are the wavelet coefficients at time step t. $A[t_1,t_2]$ and $D[t_1,t_2]$ represent the averages and differences of the coefficients across time span $[t_1,t_2]$ respectively. During the construction, we compute and store $A[0,3]$, $D[0,3]$, $D[0,1]$ and $D[2,3]$. At run time, they are used to reconstruct $A[0,1]$, or $A[2,3]$, or one of Ct ($0 \leq t \leq 3$) by traversing the time tree from top down, depending on the time step in query and the error tolerances.*

with octree nodes can be naturally organized into binary time trees to create the temporal hierarchy. Unlike the 1D temporal wavelet transforms, the 3D spatial wavelet transforms are not limited to Haar wavelets and higher order wavelets can be used to achieve better rendering image quality. However, when choosing a wavelet transform on the spatial domain, there is a tradeoff between the image quality we can obtain and the reconstruction cost it entails at run time.

3.2. Hierarchical Spatial and Temporal Error Metric

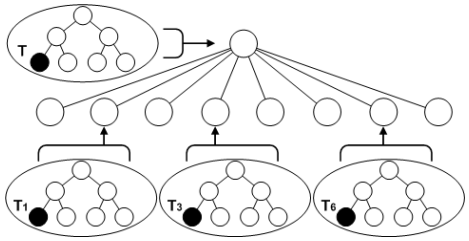

Figure 3: *Calculating the spatial error for a time tree leaf node T. The time tree nodes involved in the calculation are drawn in black. Due to space limit, only three out of eight spatial child nodes of T are drawn here.*

Coupled with the bottom-up construction of the WTSP tree, spatial error (*se*) and temporal error (*te*) are calculated for each of the time tree nodes. Our error metric is based on *mean square error* (MSE) calculation. A time tree leaf node T, which corresponds to a single time step rather than a time interval, is shown in Figure 3. We calculate the spatial error of node T as the sum of the MSE between the data at node T and the data of the same time step at T's eight spatial child nodes in the next spatial level of the octree (three of them are shown in Figure 3), adding the maximum spatial error of the child nodes. Written in formula:

$$se(T) = \sum_{i=0}^{7} MSE(T,T_i) + MAX\{se(T_i)|_{i=0}^{7}\}$$

where $T_i, i \in \{0,1,...,7\}$ are the eight spatial child nodes of node T that represent the same time step at a higher spatial resolution. If node T's time tree is associated with an octree's leaf node, we define $se(T) = 0$. Note that rather than calculating the error by computing the MSE between the lower resolution data block and the original space-time data, the error calculated in this way can be computed more quickly as we evaluate the approximation errors in a bottom-up manner. For the non-leaf nodes of a time tree, they represent data at lower resolutions in the temporal domain. We want to make sure that when such a node is chosen for rendering, none of the data blocks from the individual time steps used to construct this lower temporal resolution data will violate the user-specified spatial error tolerance (the violation happens when the error value of a parent node is less than the value of one of its descendent nodes in the time tree). To ensure this, the spatial error associated with a non-leaf time tree node is set to be the *maximum* spatial error of its left and right children in the time tree.

The temporal error of node T is calculated as follows: If T is a time tree leaf node, we define $te(T) = 0$. Otherwise, let T_l and T_r be its left and right children respectively. We calculate $te(T)$ as the sum of the MSE between the data at node T and the data at its children, adding the maximum temporal error of its children:

$$te(T) = MSE(T,T_l) + MSE(T,T_r) + MAX\{te(T_l), te(T_r)\}$$

We note that many other error norms, such as *L-infinity*, can replace the MSE in the above two formulae and be used, and our error metric is not limited to the selected MSE here. A feature of this error metric is that it guarantees that the spatial or temporal error of a parent node is greater than or equal to those of its children. Our design of this *hierarchical* error metric is useful for flexible error control when we perform the WTSP tree traversal during the rendering.

3.3. Storing Reconstructed Data for Space-Time Tradeoff

For a large time-varying data set, long chains of parent-child node dependencies exist in the WTSP tree - in order to reconstruct its own data, a node needs to recursively request the low-pass filtered subblock from its ancestor nodes. This dependency dramatically slows down the overall data reconstruction process at run time. To alleviate long chains of parent-child node dependencies and reduce the overall run-time reconstruction cost, we introduce the *EVERY-K* scheme which pre-computes and stores the reconstructed data at nodes in selective WTSP tree levels. The pre-computation is performed during the WTSP tree construction, and it serves as a way to trade disk space with reconstruction time.

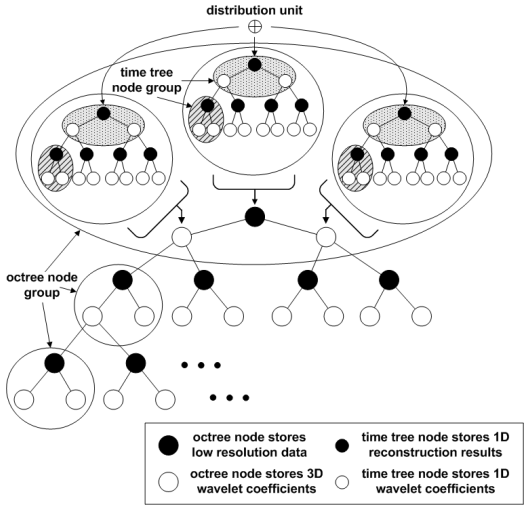

Figure 4: *The EVERY-K partition scheme. Due to space limit, for the WTSP tree skeleton, a binary tree rather than an octree is drawn here for illustration purpose. The octree nodes and time tree nodes that store reconstruction results are drawn in black. The set of time tree node groups (drawn with same pattern) that span the same time interval within one octree node group is an example of a distribution unit. In the figure, $k_o = 2$, $h_o = 6$, and $k_t = 2$, $h_t = 4$.*

In the standard WTSP tree construction algorithm discussed earlier, the time tree in each octree node is built from 1D wavelet transforms (across the time dimension) on the 3D wavelet coefficients generated from the spatial hierarchy for each time step. In the EVERY-K scheme, during the first stage of the WTSP tree construction, we perform the wavelet reconstruction in advance and store the low resolution data for octree nodes at every k_o level starting from the octree root, where $k_o < h_o$ and h_o is the height of the octree. We then build the time trees in the second stage based on the low resolution data. For the rest of octree levels, 1D wavelet transforms still apply to the wavelet coefficients as usual. To further eliminate parent-child node dependency when reconstructing data of various temporal resolutions, in each of the time trees, we pre-compute the 1D inverse wavelet transforms from the coefficients and store the reconstructed results for time tree nodes at every k_t level starting from the time tree root, where $k_t < h_t$ and h_t is the height of the time tree. In practice, h_o and h_t may not be an exact multiple of k_o and k_t respectively and this can be easily handled.

Figure 4 shows an example of the EVERY-K scheme. With the EVERY-K scheme, the problem of having long chains of parent-child node dependencies is mitigated. This is because now a node only needs to request the low-pass filtered data and wavelet coefficients up to its closest ancestor octree node and time tree node respectively, where the wavelet reconstructions have already been performed. This also allows us to partition the WTSP tree into a set of disjoint distribution units and hence eliminate data dependency for wavelet reconstruction among processors, which will be described in Section 3.4.

3.4. WTSP Tree Partition and Data Distribution

For parallel volume rendering, the WTSP tree needs to be partitioned and distributed among different processors since it is impractical to replicate the large amount of data everywhere in a cluster environment. However, if nodes with parent-child dependencies are assigned to different processors, then communication between the processors becomes inevitable. In this section we present a WTSP tree partition and data distribution scheme which eliminate the data dependency among processors and ensure a balanced workload for rendering on a PC cluster.

First of all, the WTSP tree is partitioned at every k_o levels of the spatial octree according to our EVERY-K scheme. Within each of such a partition, for each octree node that has stored the low resolution data, and the descendent octree nodes storing the 3D wavelet coefficients that depend on it, we form an octree node group. Then, for every time tree in an octree tree node group, we partition the time tree at every k_t levels according to the EVERY-K scheme. Within each of such a partition, for each time tree node that has stored the 1D wavelet reconstructed results, and the descendent time tree nodes storing the 1D wavelet coefficients that depend on it, we form a time tree node group. In each octree node group, all the time tree node groups that cover the same time span are joined into a *distribution unit*, as illustrated in Figure 4. We use the distribution units to form a *partition* of the WTSP tree, and a distribution unit is treated as a *minimum* unit that can be assigned to a processor. Since there is no data dependency between distribution units during the data reconstruction, we are able to eliminate the dependency among processors at run time.

A good data distribution scheme should ensure that all the processors receive a near equal amount of rendering workload when the user specifies the time step and error tolerances for the rendering. However, when multiresolution vol-

ume rendering is performed, different data resolutions, and thus different rendering workloads, will be chosen as an approximation of the spatio-temporal region. This makes the workload distribution task more complicated. In the following, we describe our data distribution scheme to address the load-balancing problem.

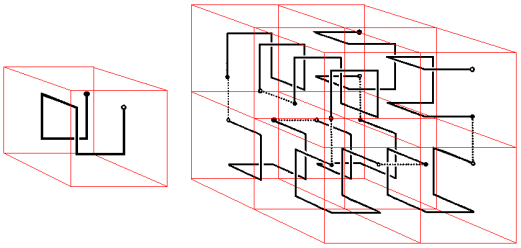

Figure 5: *The first two iterates of the Hilbert curve. In constructing one iterate from the previous one, the direction of the curve determines the orientation of the smaller cubes inside the larger one.*

Generally speaking, time-varying data usually exhibit strong spatial and temporal coherence. This implies that during the rendering, if a data block at a certain spatial and temporal resolution is selected for rendering, it is likely that the same spatio-temporal resolution for its neighboring blocks will also be selected. Therefore, if neighboring data blocks of similar spatio-temporal resolution are evenly distributed to different processors, each processor will receive approximately the same rendering workload in that local neighborhood. To achieve these, *space-filling curves* [Sag94] are utilized in our data distribution algorithm to assign the distribution units to different processors. The space-filling curve is used for its unique ability to preserve locality, meaning the traversal path along the curve always visits the adjacent blocks before it leaves the local neighborhood. An exhibition of the three-dimensional space-filling curve that fills a cube is shown in Figure 5. Previously, space-filling curves have been applied to large data visualization in parallel [CDF*03, GHSK03] and distributed client-server [PLF*03] environments to balance the workload. In our algorithm, hierarchical space-filling curves are used to traverse through the volume to create a one-dimensional ordering of the underlying volume blocks. This ordering will be used as a basis to distribute the volume blocks to different processors in a round-robin manner. Since each distribution unit in our scheme spans across k_o spatial levels, we only need to use one space-filling curve of a particular resolution for every k_o levels in the spatial hierarchy. Once the ordering of volume blocks is determined, the same order will be used to traverse the volumes with different temporal resolutions.

To ensure load balancing at run time for different spatial and temporal error tolerances, data blocks with similar spatial and temporal errors should be distributed to different processors since our error-based WTSP tree traversal algorithm usually select them together for rendering. To ensure proper distribution, in addition to the hierarchical space-filling curve traversal, we include an error-guided bucketization mechanism into our data distribution scheme. First, we partition the spatial error range and the temporal error range of the nodes in the WTSP tree into discrete intervals. Then, we iterate through all possible combinations of spatial error intervals and temporal error intervals, and create buckets for the spatial-temporal error interval combinations. Next, we traverse the WTSP tree along hierarchical space-filling curves as described above. During the traversal, every distribution unit encountered is placed into a bucket, if the *maximum* spatial and temporal errors of all the nodes in the distribution unit fall into the spatial-temporal interval of the bucket. The order of the distribution units stored in a bucket is determined by the order of the space-filling curve traversal. The intervals of the buckets are adjusted so that each bucket holds similar number of distribution units. Finally, all distribution units in each of the buckets are distributed among processors in a round-robin fashion.

3.5. WTSP Tree Traversal and Data Block Reconstruction

At run time, the user specifies the time step and the tolerances of both spatial and temporal errors to traverse the WTSP tree. The WTSP tree traversal is similar to the TSP tree traversal algorithm presented in [SCM99]. We traverse both the WTSP tree's octree skeleton and the binary time tree associated with each encountered octree node. The octree nodes in the WTSP tree are recursively visited in the front-to-back order according to the viewing direction. The tolerance for the spatial error provides a stopping criterion for the octree traversal so that the regions having tolerable spatial variations can be rendered using lower spatial resolutions. The tolerance for the temporal error is used to identify regions where it is appropriate to use data of lower temporal resolutions due to their small temporal variations. This allows us to reuse the data of those subvolumes for multiple time steps. The result of the WTSP tree traversal is a sequence of subvolumes with different sizes and characteristics of spatial and temporal coherence. 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 at a certain spatial and temporal resolution is reconstructed in the following manner: If the corresponding octree node is the root of the octree, we retrieve the data block of n voxels from its time tree hierarchy according to the time step in question. The corresponding bit stream file is accessed, and inverse 1D wavelet transforms are performed to reconstruct the data block. If the corresponding octree node is not the root of the octree, we recursively request the data block associated with its ancestor nodes, and reconstruct the corresponding data blocks if necessary. This is for extracting the low-pass filtered subblock of size $n/8$ from the data block associated with its parent node. The wavelet

coefficients of size $7n/8$ are obtained from its time tree hierarchy by decoding the corresponding bit stream file and applying inverse 1D wavelet transforms. Then, we group the low-pass filtered subblock and the wavelet coefficients into a single block of n voxels and apply an inverse 3D wavelet transform to reconstruct the data block.

Note that the reconstruction of wavelet coefficients only covers the nodes along the path from the leaf (corresponding to the time step in query) to the root in the time tree, and the recursive requesting of the data block covers the nodes along the path from the current octree node to the root of the octree. Therefore, the reconstruction time for getting the data block is $O(c_1 \times h_o + c_2 \times h_o \times h_t)$, where c_1 is the time to perform an inverse 3D wavelet transform, c_2 is the time to perform an inverse 1D wavelet transform, h_o is the height of the octree, and h_t is the height of the time tree. Utilizing the EVERY-K scheme, the cost of getting a data block is bounded by the number of levels in an octree node group (k_o) and the number of levels in a time tree node group (k_t) within a distribution unit. Accordingly, the reconstruction time is reduced to $O(c_1 \times k_o + c_2 \times k_o \times k_t)$, where $k_o < h_o$, and $k_t < h_t$ are small numbers, two or three in our experiments.

3.6. Parallel Volume Rendering

During the actual rendering, each processor only renders the data blocks identified by the WTSP tree traversal and preassigned to it during the data distribution stage, so there is no expensive data redistribution among processors. The WTSP tree traversal is done by the host processor, which will broadcast the traversal result to all the other processors, or by all processors simultaneously traversing the WTSP tree, avoiding communication among processors. Each processor only needs to have a copy of the WTSP tree skeleton with spatial and temporal errors recorded at each of the time tree nodes. 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.

As we may anticipate reusing most of the reconstructed data blocks for subsequent frames due to the spatial and temporal coherence exploited by the WTSP tree structure, it is desirable to cache the data blocks that have already been reconstructed from the WTSP tree for better rendering performance. The user can predefine a fixed amount of disk space and memory dedicated for the caching purpose at each processor. Upon requesting a data block for 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 not cached in memory or on disk, then we need to reconstruct the data 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.

4. Results

data (type)	RMI (byte)
range (threshold)	[0, 255] (0)
volume (size)	1024 * 1024 * 960 * 32 (30GB)
block (size)	64 * 64 * 32 (128KB)
tree depth	6 (octree) and 6 (time tree)
wavelet transform	Haar with lifting (both space and time)

Figure 6: *The RMI data set.*

data (type)	SPOT (float)
range (threshold)	[0.0, 10.109] (0.005)
volume (size)	512 * 512 * 256 * 30 (7.5GB)
block (size)	32 * 32 * 16 (64KB)
tree depth	6 (octree) and 6 (time tree)
wavelet transform	Daubechies 4 (space) and Haar (time)

Figure 7: *The SPOT data set.*

The two time-varying data sets used in our tests are listed in Figure 6 and Figure 7 respectively. To construct the WTSP tree, we chose the block size to be $64 \times 64 \times 32$ for the RMI data set, and $32 \times 32 \times 16$ for the SPOT data set. The decision for the block size is a tradeoff between the cost of performing the wavelet transform for a data block, and the rendering and communication overheads for final image generation. For the RMI data set, since each voxel is represented by a byte, the 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 to compress the wavelet coefficients. For the floating point SPOT data set, we use the Daubechies wavelet transform for the spatial domain and the Haar wavelet transform for the temporal domain. Lossy compression scheme was used with the threshold set to 0.005 to compress the wavelet coefficients. Setting the thresholds to zero (RMI data set) or near zero (SPOT data set) ensures the fidelity of reconstructed data and the quality of rendered images. We extend one voxel overlapping boundaries between neighboring blocks in each dimension when loading data from the original brick data files in order to produce seamless rendering. The WTSP trees constructed from the data have 10499 and 5799 non-empty octree nodes for the RMI and the SPOT data sets respectively.

The compressed data sizes are 5.843GB for the RMI data set and 3.572GB for the SPOT data set. More compression can be exploited by increasing the threshold or using vector quantization technique, at the price of sacrificing the image quality. For a given time-varying data set, its corresponding WTSP tree only needs to be constructed once and can then be used repeatedly.

The parallel rendering using software raycasting was performed on a PC cluster consisting of 32 2.4GHz Pentium 4 processors connected by Dolphin Networks. For the EVERY-K partitioning scheme, we chose $k_o = 2$ and $k_t = 2$ to partition the WTSP tree for both of the data sets. The wavelet-compressed data associated with the WTSP tree nodes were distributed among the 32 processors. We used the Hilbert curve as the space-filling curve for the data distribution.

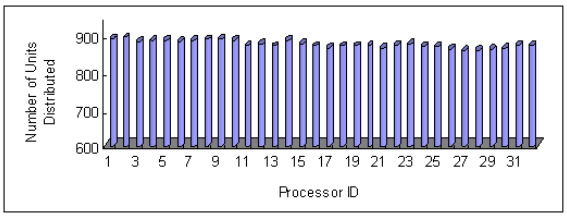

Figure 8: *The number of distribution units distributed to each of the 32 processors for the RMI data set.*

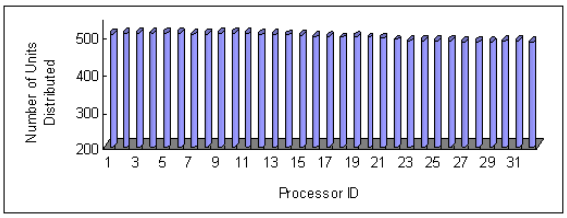

Figure 9: *The number of distribution units distributed to each of the 32 processors for the SPOT data set.*

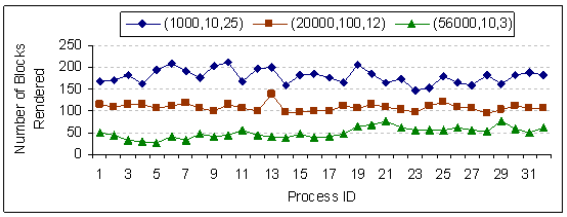

Figure 10: *The number of data blocks rendered at each of the 32 processors with three different queries of* (se,te,t) *for the RMI data set. A total of 1597, 3462, and 5723 blocks were rendered for* $(56000,10,3)$, $(20000,100,12)$ *and* $(1000,10,25)$ *respectively.*

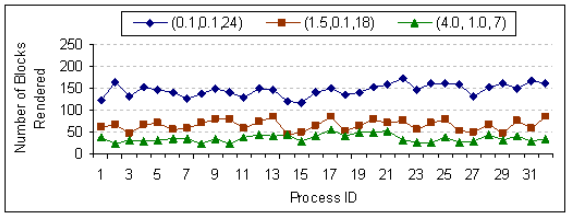

Figure 11: *The number of data blocks rendered at each of the 32 processors with three different queries of* (se,te,t) *for the SPOT data set. A total of 1142, 2019, and 4696 blocks were rendered for* $(4.0,1.0,7)$, $(1.5,0.1,18)$ *and* $(0.1,0.1,24)$ *respectively.*

data set	RMI	SPOT
(*se*, *te*, *t*)	(50, 10, 29)	(0.05, 0.01, 23)
number of blocks	6218	4840
wavelet reconstruction	15.637s	4.253s
software raycasting	10.810s	2.715s
image composition	0.118s	0.070s
overhead	3.093s	1.719s
total time	29.658s	8.757s
difference time	2.043s	0.241s

Figure 12: *The timing results for rendering the RMI and the SPOT data sets with output image resolution of* 512×512*. The difference time is the maximum timing difference between the processors.*

The hierarchical data distribution with error-guided bucketization scheme allowed the parallel volume rendering algorithm to balance the workload. Figure 8 and Figure 9 show the number of distribution units distributed to each of the 32 processors for the RMI and the SPOT data sets respectively. Figure 10 and Figure 11 show the number of data blocks rendered at each of the 32 processors for the two data sets, when three different time steps and tolerances of both spatial and temporal errors were used. It can be seen that good load-balancing was achieved, because the processors rendered approximately equal numbers of blocks. Figure 12 gives the timing results for rendering the two data sets. The well-balanced workload implies that our parallel algorithm is highly scalable. Our algorithm can achieve approximately 96.53% parallel CPU utilization, or a speedup of 30.89 times for 32 processors.

Figure 13 and Figure 14 show the rendering of the RMI and the SPOT data sets respectively at selected time steps with different spatial and temporal error tolerances. The images of the RMI data set in Figure 13 tend to have block rendering appearance, this is because Haar wavelets were used to perform the wavelet transform in both spatial and

temporal domains. In contrast, the images of the SPOT data set in Figure 14 have less block effect, as the higher order Daubechies wavelet transform was used to build the multiresolution hierarchy. Other higher order wavelets, such as quadratic spline wavelets, can also be used to perform 3D spatial wavelet transforms. However, it would require more time to reconstruct data during the rendering. Figure 15 shows multiresolution rendering results of different levels of detail for the two data sets. When the spatial and temporal error tolerances were higher, data blocks of lower resolutions were reconstructed, which resulted in a smaller number of blocks being rendered. It can be observed that, finer details of the data are kept when reducing the error tolerances, but images of reasonable quality can still be obtained at lower resolutions. The use of wavelet-based compression allowed us to produce images of good visual quality with much smaller storage space commitment.

5. Conclusion and Future Work

We have presented a multiresolution data management and rendering framework for large-scale time-varying data visualization. A hierarchical WTSP tree is designed for organizing the time-varying data that supports flexible level-of-detail data selections at different spatial and temporal resolutions. To alleviate long chains of parent-child node dependencies for data reconstruction, we proposed an algorithm to store the reconstructed data at nodes in selective tree levels and effectively reduce the overall data reconstruction cost at run time. For parallel rendering over a PC cluster, we introduced a WTSP tree partition and distribution scheme to eliminate the data dependency among processors and ensure a well-balanced workload for any user-specified time step and tolerances of both spatial and temporal errors. The experimental results with rendering of gigabytes of time-varying data demonstrated the effectiveness and utility of our framework. Future work includes utilizing graphics hardware to perform wavelet reconstruction and rendering for the run-time speedup, and incorporating optimal feature-preserving wavelet transforms into our multiresolution framework for feature detections in large-scale time-varying data sets.

Acknowledgements

This 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 is courtesy of Mark Duchaineau at Lawrence Livermore National Laboratory, and the SPOT data set is courtesy of John Clyne at National Center for Atmospheric Research. Special thanks to Jack Dongarra and Clay England from The University of Tennessee, and 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.

[BIP01] BAJAJ C., IHM I., PARK S.: 3D RGB Image Compression for Interactive Applications. *ACM Transactions on Graphics 20*, 1 (2001), 10–38.

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

[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.

[ECS00] ELLSWORTH D., CHIANG L. J., SHEN H. W.: Accelerating Time-Varying Hardware Volume Rendering Using TSP Trees and Color-Based Error Metrics. In *IEEE Volume Visualization '00* (2000), pp. 119–129.

[FJS96] FINKELSTEIN A., JACOBS C. E., SALESIN D. H.: Multiresolution Video. In *ACM SIGGRAPH '96* (1996), pp. 281–290.

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

[GS01] GUTHE S., STRASSER W.: Real-Time Decompression and Visualization of Animated Volume Data. In *IEEE Visualization '01* (2001), pp. 349–356.

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

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

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

[KMM*01] KNISS J., MCCORMICK P., MCPHERSON A., AHRENS J. P., PAINTER J., KEAHEY A., HANSEN C. D.: Interactive Texture-Based Volume Rendering for Large Data Sets. *IEEE Computer Graphics and Applications 21*, 4 (2001), 52–61.

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

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

[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.

[LPD*02] LINSEN L., PASCUCCI V., DUCHAINEAU M. A., HAMANN B., JOY K. I.: Hierarchical Representation of Time-Varying Volume Data with $\sqrt[4]{2}$ Subdivision and Quadrilinear B-Spline Wavelets. In *Pacific Graphics '02* (2002), pp. 346–355.

[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.

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

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

[MVW98] MATIAS Y., VITTER J. S., WANG M.: Wavelet-Based Histograms for Selectivity Estimation. In *ACM SIGMOD '98* (1998), pp. 448–459.

[PF01] PASCUCCI V., FRANK R. J.: Global Static Indexing for Real-Time Exploration of Very Large Regular Grids. In *ACM/IEEE Supercomputing '01* (2001).

[PLF*03] PASCUCCI V., LANEY D. E., FRANK R. J., SCORZELLI G., LINSEN L., HAMANN B., GYGI F.: Real-Time Monitoring of Large Scientific Simulations. In *ACM Applied Computing '03* (2003), pp. 194–198.

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

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

[SBS02] SOHN B. S., BAJAJ C., SIDDAVANAHALLI V.: Feature Based Volumetric Video Compression for Interactive Playback. In *IEEE Volume Visualization '02* (2002), pp. 89–96.

[SCM99] SHEN H. W., CHIANG L. J., MA K. L.: A Fast Volume Rendering Algorithm for Time-Varying Fields Using a Time-Space Partitioning (TSP) Tree. In *IEEE Visualization '99* (1999), pp. 371–377.

[Shn96] SHNEIDERMAN B.: The Eyes Have It: A Task by Data Type and Taxonomy for Information Visualizations. In *IEEE Visual Languages '96* (1996), pp. 336–343.

[Wes94] WESTERMANN R.: A Multiresolution Framework for Volume Rendering. In *IEEE Volume Visualization '94* (1994), pp. 51–58.

[Wes95] WESTERMANN R.: Compression Domain Rendering of Time-Resolved Volume Data. In *IEEE Visualization '95* (1995), pp. 168–175.

[WGS04] WANG C., GAO J., SHEN H. W.: Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing. In *Eurographics Parallel Graphics and Visualization '04* (2004), pp. 23–30.

[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.

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