A Fast Volume Rendering Algorithm for Time-Varying Fields Using a Time-Space Partitioning (TSP) Tree

Han-Wei Shen*
MRJ Technology Solutions / NASA Ames Research Center
Ling-Jen Chiang†
MRJ Technology Solutions / NASA Ames Research Center
Kwan-Liu Ma‡
University of California, Davis

Abstract

This paper presents a fast volume rendering algorithm for time-varying fields. We propose a new data structure, called Time-Space Partitioning (TSP) tree, that can effectively capture both the spatial and the temporal coherence from a time-varying field. Using the proposed data structure, the rendering speed is substantially improved. In addition, our data structure helps to maintain the memory access locality and to provide the sparse data traversal so that our algorithm becomes suitable for large-scale out-of-core applications. Finally, our algorithm allows flexible error control for both the temporal and the spatial coherence so that a trade-off between image quality and rendering speed is possible. We demonstrate the utility and speed of our algorithm with data from several time-varying CFD simulations. Our rendering algorithm can achieve substantial speedup while the storage space overhead for the TSP tree is kept at a minimum.

Keywords: scalar field visualization, volume visualization, volume rendering, time-varying fields.

1 Introduction

Visualizing large-scale time-varying fields remains one of the most challenging research problems. While a majority of the steady-state visualization techniques can be readily applied to time-varying fields, the sheer size of data often makes the task of interactive exploration impossible. The difficulties mainly come from the fact that only a small portion of data in the entire time series can fit into main memory at a time, and that the computation cost is often too high for the algorithm to run in real-time. This paper proposes an algorithm that addresses both issues to facilitate an efficient rendering of three-dimensional time-varying fields. The underlying visualization method is direct volume rendering, which has been widely used in various areas such as medical imaging, structure analysis, earth science, and computational fluid dynamics. The advantage of using direct volume rendering techniques is that both opaque and translucent structures can be visualized at the same time. Unfortunately, the computation cost of direct volume rendering is often too high for interactive applications. To improve the performance, various software and hardware solutions have been proposed in the past [1, 2, 3, 4, 5]. However, most of those methods focus on the rendering of steady-state volumes, and only a few approaches were proposed for time-varying volume rendering [6, 7].

In this paper, we present a new algorithm for rapid rendering of time-varying volumes. We note that temporal coherence is frequently present in a time-series field and, using that coherence appropriately, we can save rendering time and reduce the I/O overhead. We propose a new hierarchical data structure that is capable of capturing both the temporal and the spatial coherence. Conventional hierarchical data structures such as octrees are effective in characterizing the homogeneity of the field values existing in the spatial domain. However, when treating time merely as another dimension for a time-varying field, difficulties frequently arise due to the discrepancy between the field's spatial and temporal resolutions. In addition, treating spatial and temporal dimensions equally often prevents the possibility of detecting the coherence that is unique in the temporal domain. Using the proposed data structure, our algorithm can meet the following goals. First, both spatial and temporal coherence are identified and exploited for accelerating the rendering process. Second, our algorithm allows the user to supply the desired error tolerances at run time for the purpose of image-quality/rendering-speed trade-off. Third, the amount of data that are required to be loaded into main memory is reduced, and thus the I/O overhead is minimized. This low I/O overhead makes our algorithm suitable for out-of-core applications.

In the following, we first discuss related work on hierarchical data structures and time-varying volume rendering. Our new spatial-temporal hierarchical data structure is then described. We show how a direct volume rendering method can benefit from the new data structure. Finally, we present experimental results from several time-varying volume datasets.

2 Related Work

Many researchers have proposed the use of hierarchical data structures to speed up rendering of steady-state volumes. Levoy [4] classifies the volume into a binary representation based on the underlying voxels' opacities. Utilizing the classification, a pyramid is constructed for the purpose of space-leaping and adaptive termination of ray tracing. Laur and Hanrahan [3] proposed to store the voxels' mean value and standard deviation at each node of the pyramid. Given a user-supplied error tolerance, an octree is fit to the pyramid, and the traversal of the octree allows the volume to be drawn in different resolutions. The idea of storing the error at each node allows trading the image quality for a faster rendering speed. Wilhelms and Van Gelder further extend this idea and store voxel and cell trilinear functions at the octree node [7]. They also show that the multi-dimensional hierarchical scheme can straightforwardly support four-dimensional data such as time-varying scalar fields.

To explicitly exploit the temporal coherence, Shen and Johnson

*Current affiliation: Department of Computer and Information Science, The Ohio State University, 2015 Neil Ave. 395 Dreese Lab., Columbus, OH 43210 (hwshen@cis.ohio-state.edu)

†NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA 94035 (lchiang@nas.nasa.gov)

‡Department of Computer Science, University of California, One Shields Avenue, Davis, CA 95616-8562 (ma@cs.ucdavis.edu)

[6] proposed a differential volume rendering algorithm, which employs a difference encoding scheme to extract the volume's evolution over time. To start a volume animation, an initial image is first generated using a regular volume rendering method. For the subsequent time steps, only pixels that correspond to the voxels that change values are updated by casting new sampling rays. The differential volume rendering algorithm can save not only on rendering time, but also on disk space used to store the volume series. However, the lossless difference encoding scheme might not have the best performance when floating point data are encountered.

A different approach of volume rendering is proposed by Westermann [8]. In his method, a wavelet transform is employed to construct volumes of multi-resolutions in the form of wavelet coefficients. To extract the temporal evolution of the volume data, Westermann proposed to use the Lipschitz exponents to analyze the wavelet coefficients in time and to detect local regularity. For those regions with higher temporal variation, finer resolutions are used, and volume rendering is performed on the wavelet domain directly.

The technique introduced in this paper primarily focuses on direct volume rendering in the physical domain. We devise a hierarchical data representation similar to octrees, but one that is more suitable for capturing both temporal and spatial coherence for time-varying data. In addition, we pay special attention to the fact that the size of a typical time-varying dataset often exceeds the capacities of both texture memory and main memory existing in a workstation. Furthermore, we believe that the adaptive error control proposed by Laur and Hanrahan, and Wilhelms and Van Gelder, is important for interactive applications; therefore this capability is built into our algorithm.

3 Time-Space Partitioning Tree

In this section, we describe our new data structure that is used to represent a time-varying volume hierarchically in both the spatial and temporal domains. While the traditional octree data structures can be extended to four-dimensional trees with one extra dimension representing time, there are several noteworthy problems. First, the spatial and temporal resolutions could be very different, and this discrepancy makes it difficult to locate the temporal coherence in certain regions. We demonstrate the problem using an extreme but representative example. Let us assume that there is a time-varying $512 \times 512 \times 512$ volume with two time steps. It is only possible to subdivide the four-dimensional array into sixteen $256 \times 256 \times 256$ subvolumes with divisible time, and the subsequent branchings involve only spatial subdivisions. This implies that no temporal coherence for subvolumes smaller than $256 \times 256 \times 256$ can be detected. Another problem of using the four-dimensional trees is that coupling spatial and temporal domains makes it difficult to locate regions with only temporal coherence but not spatial coherence. This problem can be demonstrated by another example. Let us assume that a subvolume has a dramatic value variation within the spatial domain but remains unchanged across several time steps. In four-dimensional space the overall value coherence would appear to be low even though the temporal coherence alone has a strong presence. As a result, the temporal coherence can be easily missed.

Techniques that decouple temporal and spatial domains for a better utilization of the temporal coherence have been proposed in different applications. Shen proposed a temporal hierarchical index tree [9] for isosurface extraction in time-varying scalar fields. The tree recursively bisects the time domain and classifies data cells into different time spans based on the cells' temporal coherence. Shen uses the data structure to reduce the size of the isosurface cell search index and to reduce the I/O overhead. A similar approach was proposed by Finkelstein *et al.* in generating multiresolution videos [10]. In their method, a binary tree in the time domain, called time tree, is employed to store image frames corresponding to different time spans. The image frame at each node of the binary tree is represented by a quadtree data structure which can capture the spatial coherence. For frames in different time steps with temporal coherence in certain regions, links between the nodes in the time tree are created to express the relationships. Both data structures mentioned cannot be readily adopted for direct volume rendering. The temporal hierarchical index tree does not necessarily maintain the spatial locality of the volume cells, but this locality is fairly important for volume rendering. In the case of Finkelstein *et al.*'s time tree, the fixed links between nodes preclude the possibility of adjusting the error tolerance that is used to define the coherence at run time. In addition, given the fact that voxels need to be drawn in an appropriate visibility order in direct volume rendering, the procedure of following the links to access all the necessary subvolumes in correct order would be very complicated.

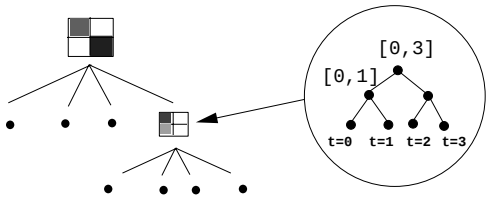


Figure 1: The TSP tree's skeleton is an octree, and each of the TSP tree nodes is a binary time tree. In the example here, the time-varying field has four time steps.

In the following, we present a new data structure called *Time-Space Partitioning (TSP)* tree. The TSP tree is designed to hierarchically represent a time-varying volume both in temporal and spatial domains. The temporal coherence is exploited based on the idea that, if the data in the volume are unchanged in a given time span, it is only necessary to perform rendering once and to reuse the same image for the animation sequence.

3.1 Data Structure

The TSP tree is a time-supplemented octree. The skeleton of a TSP tree is a standard complete octree, which recursively subdivides the volume spatially until all subvolumes reach a predefined minimum size. The difference between a TSP tree and a regular octree is that the TSP tree node contains both spatial and temporal information about the underlying data in the subvolume, while a regular octree node only contains the spatial information. To store the temporal information, each TSP tree node itself is a binary tree. Similar to Finkelstein *et al.*'s time tree [10] and Shen's temporal hierarchical index tree [9], the binary tree bisects the time span $[0, t]$ associated with the time-varying field until a unit time step is reached. Figure 1 depicts the TSP tree and one of its tree nodes in the form of a binary time tree. A quadtree is used in all the figures throughout the paper only for the purpose of illustration. The TSP tree adopts a reverse approach for combining spatial and temporal hierarchies compared to Finkelstein *et al.*'s time tree [10] which uses the binary time tree as the main skeleton and encodes a spatial quadtree into each time tree node, as shown in Figure 2. The intention behind our design is to maintain the visibility order and spatial locality among the subvolumes while traversing the TSP tree.

Every node in the binary time tree associated with a TSP tree node represents the same subvolume in the spatial domain but a different time span. The information stored in a binary time tree node includes:

- The mean value of the voxels within the subvolume in the given time span

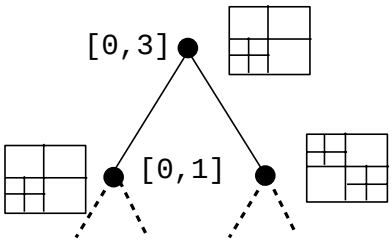


Figure 2: Finkelstein *et al.*'s time tree has a reverse combination of the octree (quadtree) and the binary time tree.

- A measurement of the subvolume's spatial error in the given time span
- A measurement of the subvolume's temporal error in the given time span

To measure the spatial error, we use the coefficient of variation, which can be seen as a normalized version of the standard deviation. This spatial error measurement serves as an indication of the subvolume's spatial coherence. That is, the lower is the spatial error, the higher is the spatial coherence. The coefficient of variation can be computed straightforwardly:

$$m = \frac{\sum_{i,t} v_{i,t}}{N}$$

$$s = \sqrt{\frac{\sum_{i,t} {v_{i,t}}^2}{N} - \left(\frac{\sum_{i,t} v_{i,t}}{N}\right)^2}$$

- Coefficient of Variation $= \frac{s}{m}$

where $v_{i,t}$ is the value of voxel i at time step t, N is the total number of voxels in the subvolume across all the time steps, m is the mean value of the voxels, and s is the subvolume's standard deviation.

To quantify a volume's temporal error in a given time span $[t1, t2]$, we propose to use the mean of the individual voxels' coefficients of variation over time. That is, we treat each voxel as an independent variable and compute its coefficient of variation among the $t2 - t1 + 1$ samples in the time span $[t1, t2]$. We then compute the average value of the coefficients of variation from all the voxels in the given subvolume and use this value as a measurement for the subvolume's temporal error. Mathematically, that is:

$$m(v_i) = \frac{\sum_{t=t1}^{t=t2} v_{i,t}}{t2-t1+1}$$

$$s(v_i) = \sqrt{\frac{\sum_{t=t1}^{t=t2} {v_{i,t}}^2}{t2-t1+1} - \left(\frac{\sum_{t=t1}^{t=t2} v_{i,t}}{t2-t1+1}\right)^2}$$

$$c(v_i) = \frac{s(v_i)}{m(v_i)}$$

- Temporal Error $= \frac{\sum_i c(v_i)}{n}$

where $m(v_i)$ is the voxel v_i's mean value in the time span $[t1, t2]$, $s(v_i)$ is the voxel v_i's standard deviation in the time span $[t1, t2]$, $c(v_i)$ is the voxel v_i's coefficient of variation, and n is the number of voxels within the subvolume. This formula is more effective in capturing the temporal coherence because the data variation in the spatial domain does not affect the result. This characteristic is important for identifying the temporal coherence that is uniquely present in a time-varying volume series that does not have any spatial coherence.

The mean, spatial error, and temporal error associated with each binary tree node in the TSP tree are used for the tree traversal during the volume rendering process, which is explained in the following sections.

3.2 Tree Traversal

For a time-varying volume series, the TSP tree only needs to be constructed once and can then be employed repeatedly. To perform volume rendering at run time, the TSP tree is first traversed to identify the subvolumes that satisfy the user-supplied error tolerances. The located subvolumes are then rendered in the correct order to construct the final image. In this section, we focus on the process of tree traversal. The volume rendering process is explained in the next section.

Our tree traversal algorithm consists of traversing the TSP tree's octree skeleton and traversing the binary time tree associated with each encountered TSP tree node. At run time, the user specifies the time step and the tolerances for both the spatial and temporal errors. The tolerance for the spatial error provides a stopping criterion for the octree traversal so that the regions having tolerable spatial variations are rendered using their mean values. The tolerance for the temporal error, i.e., mean of the individual voxels' coefficients of variation over time, is used to identify regions where the rendering results can be reused for multiple time steps due to their small temporal variations.

The idea of the TSP tree traversal is similar to the traversal of a standard octree. That is, starting from the root of the TSP tree's octree skeleton, we recursively walk down the tree and check whether the encountered node's spatial and temporal errors satisfy the user's error tolerances. Because each TSP tree node is in fact a binary time tree, the error checking of a TSP tree node requires a traversal to the TSP tree node's corresponding binary time tree. This time tree traversal is performed using the following algorithm. Starting from the root of the time tree, we perform:

- **Step 1. Temporal error checking**: Check whether the temporal error at the current time tree node is smaller than the user-supplied tolerance. If not, we traverse down to the branch of the time tree that spans the current time step and repeat the process in this step. Otherwise, we mark that the subvolume has an acceptable temporal coherence in the time span represented by the current time tree node and go to the next step.
- **Step 2. Spatial error checking**: Check whether the spatial error at the current binary time tree node is smaller than the user tolerance. If yes, we stop the traversal and report that the error checking for the TSP tree node has succeeded. Otherwise, we traverse down to the branch of the time tree that spans the current time step and go back to the process in step 1. If the current time tree node is a leaf node, we report that the error checking for the TSP tree node has failed.

If the current TSP tree node has passed the error checking, we can use the precomputed mean value stored at the current time tree node to represent the subvolume. Otherwise, we recursively walk down to the TSP tree node's eight children in the octree skeleton. If the current TSP tree node is a leaf node, we need to use the actual volume data to represent this region.

When the recursive TSP tree traversal is completed, a series of subvolumes with different sizes and characteristics of spatial and temporal coherence are collected. Some subvolumes have low spatial variations and, therefore, are represented by their mean values.

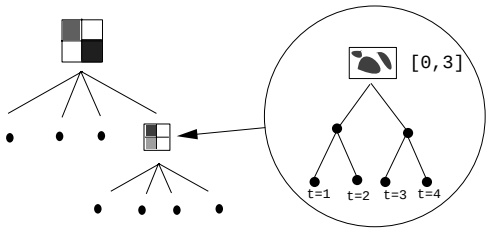


Figure 3: The volume rendered image can be shared among several time steps if the TSP tree node has a high temporal coherence.

On the other hand, for the subvolumes that have high spatial variations, the actual voxel data are used. Based on their temporal coherence, the partial rendering results from the subvolumes are used to construct the final image either for only one time step, when their temporal coherence is low, or for several time steps due to the subvolumes' high temporal coherence. In the next section, we describe the process of volume rendering.

3.3 Volume Rendering

During the tree traversal, the nodes in the TSP tree are recursively visited in the front-to-back visibility order according to the viewing direction. For a regular Cartesian grid volume, this visibility order among octree nodes can be straightforwardly determined [11]. Our rendering algorithm adopts a divide-and-conquer paradigm. That is, the subvolumes that are collected during the traversal process are rendered independently. The final image is then constructed by compositing the partial images' colors and opacities. We note that there is no restriction on the type of rendering algorithms to be used for generating the partial results, and the TSP tree data structure can readily adopt suitable accelerating techniques. It is also worth mentioning that in order to produce a correct rendering result, the adjacent subvolumes need to have overlapping boundaries if the trilinear interpolation scheme is used. This is to ensure that there is no gap between subimages.

To accelerate the time-varying volume rendering process, we store the partial images generated from the subvolumes in their associated nodes in the TSP tree. The time span that corresponds to the subimage according to the subvolume's temporal coherence is also saved. When the user chooses to render the volume at a different time step, the tree traversal process described above is performed again. During the traversal, in case that the viewing parameters remain the same, if a subvolume that has high temporal coherence is encountered and if the subimage cached previously has a time span containing the current time step, this cached image is directly used, and the rendering of the subvolume is entirely skipped. The utilization of previously cached images due to the high temporal coherence of the subvolume allows us to save on rendering time. In the example shown in Figure 3, the image that is generated for the time span $[0, 3]$ at the TSP tree node can be used for the underlying four time steps if the node has enough temporal coherence.

The performance of our rendering algorithm depends on the degree of temporal coherence existing in the data and can be controlled by the user. If the user desires to preview the volume animation in a coarse resolution, a higher temporal error tolerance can be used. On the other hand, if the data is highly coherent in time, even when the user demands full accuracy, our algorithm can still detect the coherence and perform the rendering efficiently.

3.4 Memory Optimization and Out-of-Core Rendering

It is known that octrees can incur substantial memory overhead due to the sheer size of the tree nodes. This overhead often prevents the use of octrees for visualizing large-scale time-varying data. To reduce the memory overhead, we restrict the size of the subvolumes in the TSP tree by stopping the volume subdivision when the leaf node reaches a predefined minimum size during the tree construction. We found that using this "shallow" octree as the TSP skeleton can significantly reduce the tree size.

In fact, using "shallow" TSP trees is necessary for reducing the rendering overhead. In our rendering algorithm, the TSP tree nodes are rendered independently, and the subimages are composited to form the final image. Without limiting the subvolume size, a great number of very small volumes can be generated as the result of the tree traversal. This will incur a huge overhead in the image compositing.

The TSP tree algorithm is suitable for out-of-core volume rendering because of its high degree of memory access locality [12, 13]. Due to the divide-and-conquer paradigm used by our algorithm, each subvolume is rendered separately. Therefore, it is not necessary for the entire volume to remain in main memory at the same time. When employing the TSP tree algorithm in an out-of-core application, the basic I/O unit, also called *brick*, is the leaf node in the TSP tree. A brick needs to be loaded into main memory only when the subvolume is being rendered and its spatial coherence is too low so that the original data is needed. When rendering a time series, the utilization of the temporal coherence in our algorithm further reduces the amount of brick I/O because the rendering of those bricks that do not change over time is avoided. We have incorporated the application-control demand paging system proposed by Cox and Ellsworth [13] into our implementation. Performance studies are shown in the next section.

4 Results and Discussion

We have implemented a time-varying volume rendering algorithm using the TSP tree data structure. Our implementation adopts a straightforward ray casting paradigm in which a sampling ray is cast from each pixel into the volume space. The sampling process includes trilinear interpolations, lighting calculation, and color mapping. In this section, we present experimental results on the TSP tree algorithm for steady-state and time-varying volume rendering. The performance measurements were done on an SGI Onyx2 workstation with a 195MHz MIPS R10000 processor and 512 megabyte memory. The main focus of our studies is on understanding the performance gains that can be achieved using the TSP tree under various user-supplied error tolerances. We are also interested in studying the trade-offs between rendering speed and image quality. Three regular Cartesian grid datasets, as shown in Table 1, were used in our studies. The shock wave dataset was generated from a simulation of the unsteady interaction of a planar shock wave with a randomly-perturbed contact discontinuity [14]. The shear flow dataset was obtained from a study of the generation and evolution of turbulent structures in shear flows. The delta wing dataset was computed on a curvilinear grid in physical space and rendered in its corresponding computational space. Table 2 shows the size of the TSP tree, the percentage to the original dataset, and the TSP tree construction time for each of the test datasets. The Branch-On-Need (BON) method [15] was used in the construction of the TSP skeleton, and the minimum block size for the TSP tree node was restricted to $16 \times 16 \times 16$. It can be seen that the space required by the TSP tree is fairly small, namely lower than eight percent of the original data, and the preprocessing time for the tree construction is not overly excessive.

Data Set	# Time Steps	Dimensions
Shock Wave	30	$512 \times 64 \times 64$
Shear Flow	40	$128 \times 128 \times 128$
Delta Wing	50	$67 \times 209 \times 49$

Table 1: Experimental datasets

Data Set	TSP Tree Size	Percentage	Time
Shock Wave	18.3	7.3%	116
Shear Flow	20.01	5.9%	156
Delta Wing	0.53	0.39%	59

Table 2: TSP tree sizes (in megabytes), the percentages to the original datasets, and the construction time (in seconds)

The TSP tree can be used to speed up the rendering of a steady-state volume by utilizing the spatial coherence. Table 3 shows the rendering speed of the TSP tree algorithm for a single time step of the shock wave dataset. The image size is 300×300. We used four different spatial error tolerances, which are the minimum coefficients of variation allowed for the volume blocks. It can be seen that the TSP tree is capable of utilizing the spatial coherence existing in the dataset for speeding up the rendering. The loss of the image quality is visually tolerable for the shockwave dataset, as shown in the Color Plate. We are currently investigating the design of appropriate error metrics to measure the degradation of the image quality due to the error introduced during the rendering. Table 4 and Table 5 show the results for the delta wing and the shear flow datasets. The delta wing dataset has a higher spatial coherence, so even with very low error tolerances we can still obtain a good speedup. The turbulent shear flow data are much less coherent. Therefore, higher error tolerances were used in order to obtain speedup. However, the loss of certain fine features became visible. The Color Plate shows the volume rendered images for each of the datasets.

One of the TSP tree's main goals is to accelerate the rendering of large-scale time-series volumes. To test the effectiveness of our algorithm, we performed volume rendering on each of the test datasets by sequentially stepping through each time step of the volumes and measuring the rendering speed. Table 6 lists the rendering time of five selected time steps of the shock wave dataset. The error tolerance for the temporal coherence was 0.01, and we used zero error tolerance for the spatial coherence so that we can concentrate on analyzing the utilization of the temporal coherence. For the first time step of the volume, a complete rendering was needed, so no speedup was gained. However, for the subsequent time steps, the temporal coherence was utilized, and only a portion of the volume bricks at each time step needed to be rendered. As a result, it can be seen from the table that we can achieve speedup factors of 4.7 to 6.9. A snapshot of the animation sequence for the time-varying shock wave data is shown at the bottom of the Color Plate, where

Shock Wave				
Error Tolerance	0	0.01	0.05	0.08
Rendering Time (seconds)	7.32	2.45	2.35	1.99
Speedup Factor	1	2.99	3.11	3.67

Table 3: Rendering time and speedup factors with four different spatial error tolerances for a single time step of the shock wave dataset

Delta Wing				
Error Tolerance	0	0.005	0.02	0.03
Rendering Time (seconds)	15.8	10.7	4.4	4.38
Speedup Factor	1	1.47	3.59	3.61

Table 4: Rendering time and speedup factors with four different spatial error tolerances for a single time step of the delta wing dataset

Shear Flow				
Error Tolerance	0	0.7	0.9	0.95
Rendering Time (seconds)	30.6	19.6	14.2	12.8
Speedup Factor	1	1.56	2.15	2.39

Table 5: Rendering time and speedup factors with four different spatial error tolerances for a single time step of the shear flow dataset

we compare the volume rendered images of time step 14 generated using 0 and 0.01 temporal error tolerances. It can be seen that the degradation of the image quality is fairly small while the rendering time was reduced 6.84 times when 0.01 error tolerance was used. Table 7 and Table 8 show the results for the delta wing and the shear flow datasets. The temporal error tolerances were 0.001 for the delta wing dataset and 0.8 for the shear flow dataset. The turbulent shear flow dataset is less coherent in the temporal domain so higher temporal error tolerances were used and image degradations were visible. The delta wing dataset can be rendered efficiently without an excessive amount of feature missing due to its high degree of temporal coherence.

To understand the suitability of the TSP tree algorithm for out-of-core applications, we measured the sparseness of the data access in the TSP tree algorithm. The data used in our experiments was a four-time-step $1024 \times 128 \times 128$ shock wave dataset, and we used $32 \times 32 \times 32$ as the minimum brick size. The results were measured on an SGI Maximum Impact workstation with 128 megabyte main memory and a 195 MHz MIPS R10000 processor. We incorporated the application-controlled demand-paging algorithm proposed by Cox and Ellsworth [13] into our TSP tree implementation. In the test, we used a 0.01 error tolerance for the temporal coherence. Table 9 shows the rendering time and the number of the bricks that were needed at each time step. It can be seen that the temporal coherence existing in the dataset allows us to load only about 13% of bricks during the volume animation, and the rendering speed was also accelerated. This sparse traversal characteristic makes the TSP tree algorithm a good candidate for out-of-core applications.

5 Conclusions and Future Work

We have presented a fast volume rendering algorithm for three-dimensional time-varying fields. The core component of our algorithm is a new data structure called Time-Space Partitioning tree which can capture the temporal coherence more effectively than the conventional high-dimensional octrees. This effectiveness mainly comes from the fact that we decouple the temporal and the spatial domains when analyzing the time-varying data so that the coherence uniquely existing in the time domain can be identified. Our new algorithm successfully achieves the following goals. First, both the spatial and the temporal coherence are utilized for accelerating the time-varying volume rendering. Second, the amount of volume data I/O is reduced, and the locality of the data access is improved. Third, the user has flexible control of the errors so that it is possible

Shock Wave					
Time Step	0	7	14	21	28
Rendering Time	7.87	1.66	1.15	1.14	1.21
Speedup Factor	1	4.74	6.84	6.90	6.50

Table 6: Rendering time (in seconds) and speedup factors for five different time steps of the shock wave dataset. The temporal error tolerance was 0.01.

Delta Wing					
Time Step	0	12	24	36	48
Rendering Time	15.7	6.3	2.86	2.87	5.3
Speedup Factor	1	2.5	5.4	5.4	3.1

Table 7: Rendering time (in seconds) and speedup factors for five different time steps of the delta wing dataset. The temporal error tolerance was 0.001.

to trade the image quality for the rendering speed.

Future work includes further studies of the applicability of our algorithm for out-of-core applications. The relationship between the TSP tree brick size and the rendering time and the I/O overhead will be investigated. We are also incorporating the TSP tree into a hardware volume rendering program using three-dimensional texture mapping. Focus will be on reducing the data traffic between the main memory and the texture hardware and rendering very large-scale data on machines with a limited texture memory capacity.

Acknowledgments

This work was supported in part by NASA contract NAS2-14303. We would like to thank Ravi Samtaney, Neal Chaderjian, and John Shebalin for providing the datasets. Special thanks to Randy Kaemmerer for his meticulous proofreading of this manuscript and valuable suggestions. We also thank David Ellsworth, Tim Sandstrom, and other members in the Data Analysis Group at NASA Ames Research Center for their helpful comments and technical support.

References

[1] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458. ACM SIGGRAPH, 1994.

[2] K.-L. Ma, J.S. Painter, C.D. Hansen, and M.F. Krogh. Parallel volume rendering using binary-swap image composition. *IEEE Computer Graphics and Applications*, 14(4):59–68, 1994.

[3] D. Laur and P. Hanrahan. Hierarchical splating: A progressive refinement algorithm for volume rendering. In *Proceedings of SIGGRAPH 91*, pages 285–287. ACM SIGGRAPH, 1991.

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

[5] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 91–98, 1994.

Shear Flow					
Time Step	0	9	18	27	36
Rendering Time	31.1	7.2	7.9	12.3	6.1
Speedup Factor	1	4.3	3.9	2.5	5.1

Table 8: Rendering time (in seconds) and speedup factors for five different time steps of the shear flow dataset. The temporal error tolerance was 0.8.

Shock Wave ($1024 \times 128 \times 128$)				
Time Step	0	1	2	3
# Bricks Loaded	561	73	75	72
Percentage	100%	13.0%	13.3%	12.8%
Rendering Time (in seconds)	51.5	26.1	27.1	27.9

Table 9: Number of bricks needs to be loaded into main memory in the time-varying volume rendering for the shock wave dataset. The brick size was $32 \times 32 \times 32$.

[6] H.-W. Shen and C.R. Johnson. Differential volume rendering: A fast algorithm for flow animation. In *Proceedings of Visualization '94*, pages 188–195. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[7] J. Wilhelms and A. Van Gelder. Multi-dimensional tree for controlled volume rendering and compression. In *Proceedings of 1994 Symposium on Volume Visualization*, pages 27–34. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[8] R. Westermann. Compression domain rendering of time-resolved volume data. In *Proceedings of Visualization '95*, pages 168–178. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[9] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *Proceedings of Visualization '98*, pages 159–166. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[10] A. Finkelstein, C.E. Jacobs, and D.H. Salesin. Multiresolution video. In *Proceedings of ACM SIGGRAPH '96*, pages 281–290, 1996.

[11] S. Fang, R. Srinivasan, S. Huang, and R. Raghavan. Deformable volume rendering by 3d texture mapping and octree encoding. In *Proceedings of Visualization '96*, pages 73–80. IEEE Computer Society Press, Los Alamitos, CA, 1996.

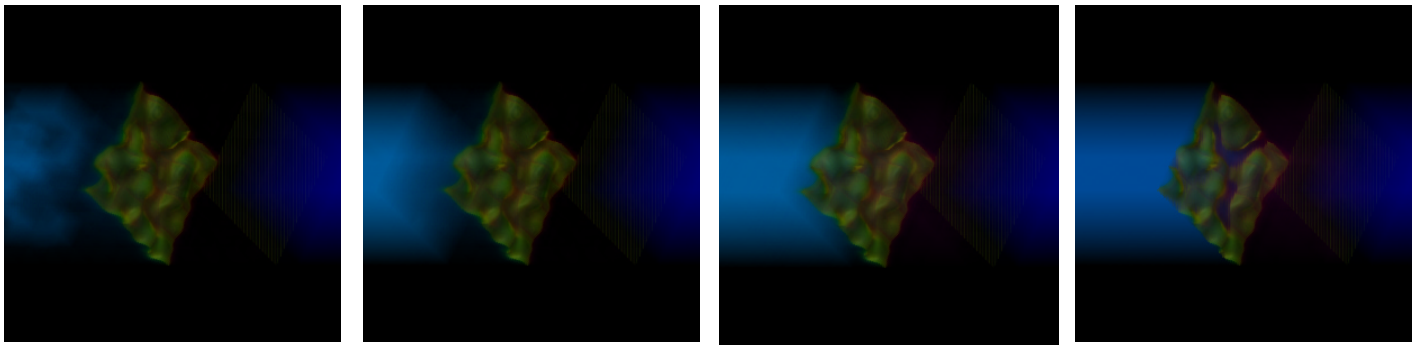
[12] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P. Sloan. Interactive ray tracing for isosurface rendering. In *Proceedings of Visualization '98*, pages 233–238. IEEE Computer Society Press, Los Alamitos, CA, 1998.

[13] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of Visualization '97*, pages 235–244. IEEE Computer Society Press, Los Alamitos, CA, 1997.

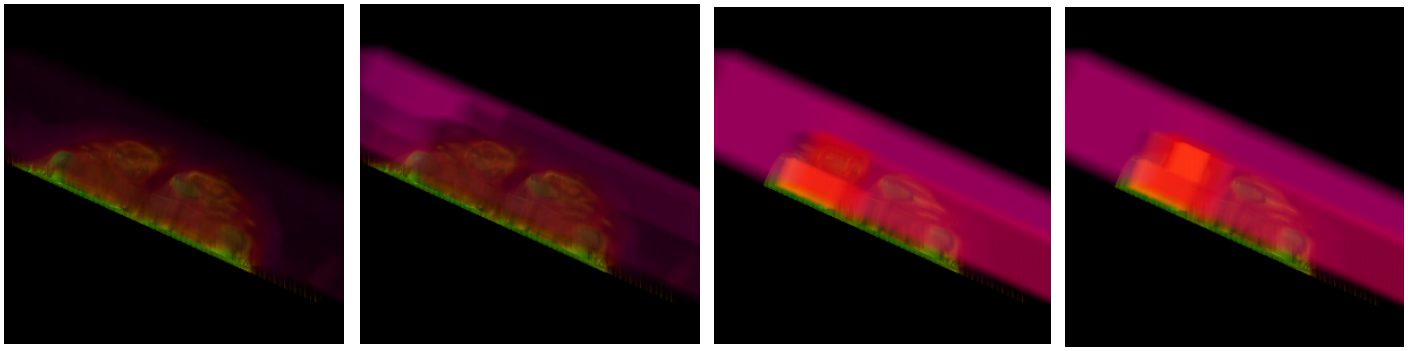
[14] D.I. Meiron and R. Samtaney. 3D simulations of the Richtmyer-Meshkov instability with re-shock. *Bulletin of the American Physical Society*, 43(9):2104.

[15] J. Wilhelm and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

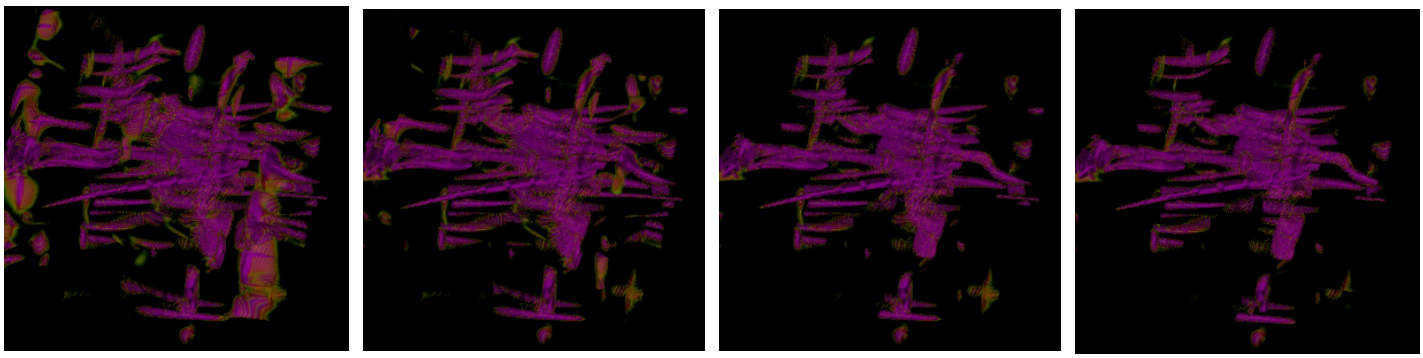
Steady Shock Wave (error = 0, 0.01, 0.05, 0.08 from left to right)



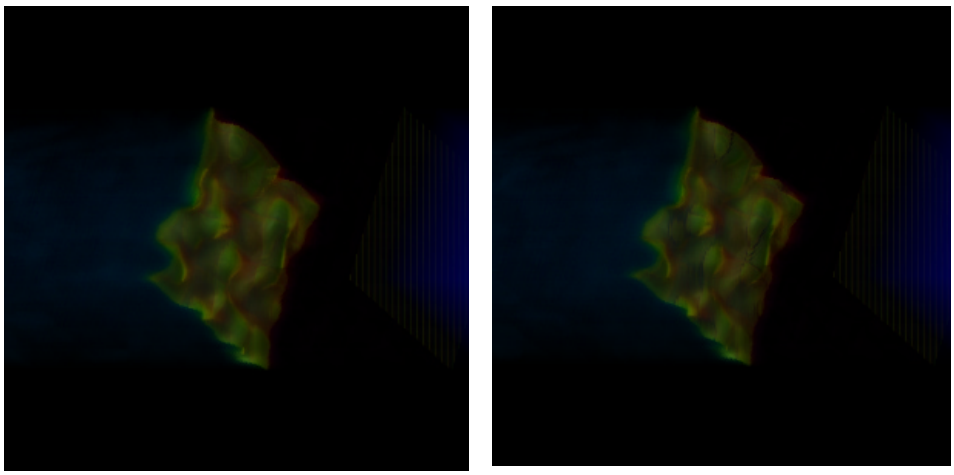
Steady Delta Wing (error = 0,0.005,0.02,0.03 from left to right)



Steady Shear Flow (error = 0,0.7,0.9,0.95 from left to right)



Time–Varying Shock Wave (error = 0, 0.01, time step = 14)



Color Plate: Image Comparisons of Steady and Time–Varying Volume Rendering using the TSP Trees