Fast Time-Varying Volume Rendering Using Time-Space Partition (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‡
ICASE

Abstract

We present a new algorithm for rapid rendering of time-varying volumes. A new hierarchical data structure that is capable of capturing both the temporal and the spatial coherence is proposed. 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.

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 perform 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]. However, most of those methods focus on the rendering of a single volume, and only a few approaches were proposed for time-varying volume rendering [5, 6]

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 volume 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 that how a direct volume rendering method can benefit from the new data structure. Finally, we present experimental results from several time-varying volume data sets.

2 Related Work

Many researchers have proposed the use of hierarchical data structures to speed up rendering of a steady-state volume. 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 data accuracy at each node allows trading the image quality for faster rendering speed. Wilhelms and Van Gelder further extend this idea and store voxel and cell trilinear functions at the octree node[6]. They also show that the multi-dimensional hierarchical scheme can straightforwardly sup-

*NASA Ames Research Center, Mail Stop T27A-2, Moffett Field, CA 94035 (hwshen@nas.nasa.gov)
†NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA 94035 (lchiang@nas.nasa.gov)
‡NASA Ames Research Center, Mail Stop T27A-1, Moffett Field, CA 94035 (lchiang@nas.nasa.gov)

port four-dimensional data such as time-varying scalar fields.

To explicitly exploit the temporal coherence, Shen and Johnson[5] 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 the 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 [7]. In his method, 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, Westerman 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 perform on the wavelet domain directly.

The technique introduced in this paper primarily focuses on direct volume rendering on 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 data set 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 Partition Tree

In this section, we describe our new data structure, Time-Space Partition Tree (TSP Tree), which is used to represent a time-varying volume hierarchically in both spatial and temporal domains. While the octree data structures can be extended to four-dimensional trees with one extra dimension representing time, there are several noticeable 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 is not detected.

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 [8] 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[9]. 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 those frames in different time steps, but which have temporal coherence in certain regions, links between the nodes in the time tree are created to express the relationships. Both of the data structures mentioned cannot be directly adopted for direct volume rendering. The temporal hierarchical index tree does not maintain the spatial locality of the volume cells, but this locality is fairly important for direct volume rendering. In the case of the time tree, the fixed links between nodes preclude the possiblity of adjusting the error tolerance that is used to define the coherence at run time. In addition, given the fact that the octree nodes need to be drawn in 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.

In the following, we present a new data structure called *Time-Space Partition (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 reuse the same image for the animation sequence. Combined with the exploitation of spatial coherence, the time-varying volume rendering speed can be accelerated.

3.1 Data Structure

The TSP tree is a time-supplemented octree. The skeleton of a TSP tree is equivalent to a regular 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 in the time span $[0, t]$ associated with the time-varying field. Similar to Finkelstein *et al.* 's time tree[9] and Shen's temporal hierarchical index tree[8], the binary tree bisects the given time span until a unit time step is reached. Figure [??] depicts the TSP and binary time trees.

Every node in the binary time tree of a TSP tree node represents the same subdomain in the volume but a different time span. The information stored in a subnode includes:

- The mean value of the voxels in the subvolume across all the time steps in the particular time span
- The standard deviation to the voxel values in the subdomain
- A measurement of the temporal coherence for the subvolume within the time span

The mean value and the standard deviation can be computed straightforwardly. That is:

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

where $v_{i,t}$ is the value of voxel i at time step t, and N is the total number of voxels in the subvolume across all the time steps. Note

that the formula shown here for computing the standard deviation is equivalent to the standard formula usually presented in statistics literature but is more suitable for the one-pass bottom-up octree construction.

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

- $s(v_i) = \sqrt{\frac{\sum_{t=t1}^{t=t2} {v_{i,t}}^2}{t2-t1+1} - (\frac{\sum_{t=t1}^{t=t2} v_{i,t}}{t2-t1+1})^2}$
- Temporal Coherence $= \frac{\sum_i s(v_i)}{n}$

where the $s(v_i)$ is the voxel v_i's standard deviation in time span $[t1, t2]$, and n is the number of voxels within the subvolume. The temporal coherence, as defined above, is more effective than the overall standard deviation in four-dimensional space. For, the data variation within the spatial domain at an individual time step does not have a direct effect on our measurement. Thus, the temporal coherence, which exists in those subvolumes that does not have any spatial coherence, can still be detected.

The mean, standard deviation, and temporal coherence quantities are used for the traversal of the TSP tree during the volume rendering process, which is explained in the following sections.

3.2 Tree Traversal

The TSP tree adopts an opposite approach for combining spatial and temporal hierarchies comparing to Finkelstein *et al.* 's time tree [9] which uses the binary time tree as the main skelton and encodes a spatial quadtree into each time tree node. The intention behind our design is to maintain the visibility order and spatial locality among the subvolumes in the process of TSP tree raversing.

For a given series of time-varying volumes, the TSP only needs to be contructed once and can be repeatedly employed when performing volume rendering. To initiate a rendering at run time, the user supplies the current time step and the error tolerances for both the standard deviation of the volume and the temporal coherence. The tolerance for the standard deviation provides a stopping criterion during the tree traversal so that the regions satisfying this criterion, i.e., having tolerable spatial coherence, are rendered using their mean values for the encompassed volumes. The error tolerance for the temporal coherence, i.e., mean of the individual voxels' standard deviations over time, is used to identify regions where their volume renderered images can be reused across several time steps. Based on the error tolerances and the current time step, the traversal of the TSP tree is done by using the regular octree traversal strategy. That is, we recursively visit the tree node starting from the root to check if its spatial and temporal coherence satisfy the user's tolerance. If yes, we render this subvolume as a whole. Otherwise we walk down to its eight children following the back-to-front visibility order, which can be readily determined based on the viewing direction [10], and continue the traversal.

As mentioned previously, each TSP tree node itself is a binary time tree. For each TSP tree node, we perform the error checking, starting from the root of the time tree, by using the following steps:

- Check if the temporal coherence measure at this node is smaller than the user tolerance. If no, traverse down to the branch of the time tree that corresponds to the current time and repeat this step. Otherwise, go to the next step.
- Check if the standard deviation is smaller than the user tolerance. If yes, then the associated mean value is used to draw the entire subvolume and terminate the TSP tree traversal along this path. Otherwise, return to the TSP traversal main routine and indicate that it is necessary to go down to the children of the current TSP tree node.

The end result of the TSP traversal consists of two types of subvolumes. The first type has low spatial variation and, therefore, is represented by average values. The second type has high variation and, therefore, the actual data are used.

3.3 Volume Rendering

Our volume rendering algorithm adopts the divide-and-conquer paradiagm. That is, the subvolumes gathered during the traversal process are rendered independently, and the final image is a composition of the intermediate results from the subvolumes. In the case of performing volume rendering by using ray casting techniques, the intermediate results are subimages with additional opacity information for the final blending. If the three-dimensional texture mapping hardware is used, the intermediate results may be trasient polygons from the slicing planes.

One of the main goals of the TSP tree algorithm is to accelerate the time-varying volume rendering. To achieve this, we save the intermediate results from the rendering of the subvolumes in their associated TSP tree nodes. When the user continues to render a different time step, we repeat the tree traversal process as mentioned above. If nodes that have high temporal coherence are encountered and if the intermediate rendering results stored from a previous rendering still cover the current time step, we can entirely skip the rendering process and directly reuse the results. This is illustrated in Figure[??].

The performance gain by using the TSP tree is determined by both the data characteristics and by the error tolerance specified by the user. If the user wishes to perform a quick preview of a time-varying volume animation, a higher error tolerance can be used. On the other hand, if the user demands full accuracy and if temporal coherence resides in the data set, the TSP will detect and utilize this coherence for speeding up the rendering.

3.4 Memory Optimization and Out-of-Core Rendering

In the process of constructing the TSP tree, the volume is not subdivided into individual voxels. Instead, we use a predefined minimum size for the subvolume so that the leaf nodes of the TSP tree in fact are volume bricks. There are two primary reasons for this. First, by limiting the minimum size of the subvolume, the storage space required by the TSP tree is significantly reduced. Second, since our divide-and-conquer approach renders each subvolume independently, not limiting the minimum brick size would inevitably increase the overhead of compositing a large number of small subimages to form the final image.

In addition to reducing storage and the overhead for rendering, the use of bricks as the basic blocks in the TSP tree guarantees the locality of memory access. This characteristics is extremely important for out-of-core applications [11, 12]. In fact, the high memory access locality provided by the TSP tree also facilitates an efficient use of the three-dimensional texture hardware which usually has a much more limited memory capacity. We have implemented a volume rendering algorithm which contains the the application-control demand paging system proposed by Cox and Ellsworth. Experimental results are shown in section?.

4 Conclusions

We have presented a new data structure called *Time-Space Partition (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 reuse the same image for the animation sequence. Combined with the exploitation of spatial coherence, the time-varying volume rendering speed can be accelerated.

Acknowledgments

This work was supported in part by NASA contract NAS2-14303. We would like to thank Ravi Samtaney, Neal Chaderjian, and Peggy Li for providing their data sets. Special thanks to Randy Kaemmerer for his meticulous proofreading of this manuscript and valuable suggestions. We also thank 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] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] S. Fang, R. Srinvasan, 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.

[11] 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.

[12] 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.