Visualization Techniques for Time-Varying Volume Data

Kwan-Liu Ma *
University of California, Davis

Han-Wei Shen †
Ohio State University

Abstract

Visualizations of time-varying volume data sets, which may be obtained from numerical simulations or sensing instruments, provide scientists insights into the detailed dynamics of the phenomenon under study. This paper is concerned with efficient representation and rendering of time-varying volume data. We studied the characteristics of typical time-varying data sets and derived a flexible representation for the data to facilitate visualization and rendering calculations. A variety of time-varying volume data sets were used for our study. In many cases over 90% reduction in both storage space and inter-frame delay were achieved by exploiting the high level of spatial and temperal coherence in the data set.

1 Introduction

The ability to study time-varying phenomena helps scientists understand complex problems. The size of time-varying data sets not only demands excessive storage space but also presents difficult problems for both data analysis and visualization. For example, a relatively small, single-variable time-varying volume data set consisting of one hundred time steps each of which stores $256\times256\times256$ floating point numbers already requires 6.4 gigabytes storage space.

This paper discusses strategies integrating compression and rendering techniques to achieve flexible and efficient rendering of time-varying volume data. Although volume data compression has been studied by many researchers [1, 11, 2, 3, 7], very few have considered the additional dimension of time-varying data. With our strategy, compression is achieved using scalar quantization along with an octree and difference encoding. By exploiting spatial and temporal coherence in the data, neighboring voxels may be fused into macro voxels if they have similar values, and two subtrees at consecutive time steps may be merged if they are identical.

A ray-casting volume renderer was restructured to efficiently render the encoded data. With the tree representation it is possible to perform selective rendering, and when appropriate to distribute both the data and rendering calculations to multiple workstations to achieve desirable interaction.

Four time-varying volume data sets were used for our study. We show how some of the data sets may be encoded according to data statistics or user's knowledge to achieve better space and rendering efficiency. We also discuss how to eliminate or hide various overheads introduced by using the tree representation. Our test results show that in general the amount of savings we can obtain in storage space as well as in rendering time justifies our approach.

*Department of Computer Science, University of California, One Shields Avenue, Davis, CA, 95616, ma@cs.ucdavis.edu.

†Department of Computer and Information Science, the Ohio State University, 2015 Neil Avenue, Columbus, OH 43210, hwshen@cis.ohio-state.edu.

2 Related Work

The previous work most closely related to ours is the thorough study done by Wilhelms and Van Gelder [14] on the design of hierarchical data structures for controlled compression and volume rendering. They extend octrees and a branch-on-need (BON) subdivision strategy [13] to handle multi-dimensional data. The basis of their work is a hierarchical data model which is well described in their paper. The resulting multi-dimensional tree stores a model of the data and evaluation information about the error of the model as well as importance of the data to control compression rate and image quality. They also propose eight evaluation metrics for performing selective traversal and visualization of the encoded data. Among the nine data sets used for their study, seven are three-dimensional data and two are time-varying data.

Another closely related work is the ray-cast rendering strategy introduced by Shen and Johnson [10] which they call *differential volume rendering*. By exploiting the data coherency between consecutive time steps, they are able to reduce not only the rendering time but also the storage space by 90% or more for their two test data sets which are highly temporally correlated and contain spatially coherent *byte* data. Differential volume rendering is potentially parallelizable and a caching technique [6] may be integrated into the renderer to avoid recalculations for visualizing irregular data.

Based on similar concepts, Westermann [12] performs wavelet encoding of each time step separately to generate a compressed multiscale tree structure. Feature extraction and tracking as well as further compression can be obtained by examining the resulting multiscale tree structures and wavelet coefficients. Using wavelet transform offers an underlying analysis model to characterize time-varying data.

In contrast to Whilhelms and Van Gelder's work, our main focus is on time-varying volume data (i.e. four-dimensional data), even though we also use octree encoding and error evaluation for selective traversal. Furthermore, While their model treats all dimensions the same way, we apply difference encoding to the time domain. We especially pay attention to the quantization step and investigate how quantization may assist subsequent compression and rendering steps, and influence the visualization results. We develop a rendering strategy taking advantage of a tree representation of the time-varying data. Examination of the encoded data identifies partial images built from subtrees which have not changed and therefore may be reused in the following time steps. In contrast to Shen and Johnson's work, we use data sets with distinct properties which are not all highly spatially and temporally coherent in order to perform a more

Table 1: Four Test Data sets.

data set	time steps	spatial resolution
Turbutlent Vortex Flow	100	128^3 *float*
Thermal Convection	101	128^3 *float*
Turbulent Jets	150	128^3 *float*
Turbulent Shear Flow	81	128^3 *float*

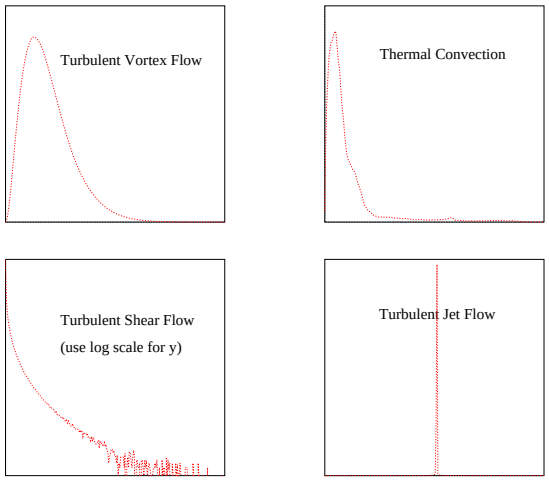

Figure 1: Histograms of the data sets. Each plot shows the distribution of data values in the whole data set.

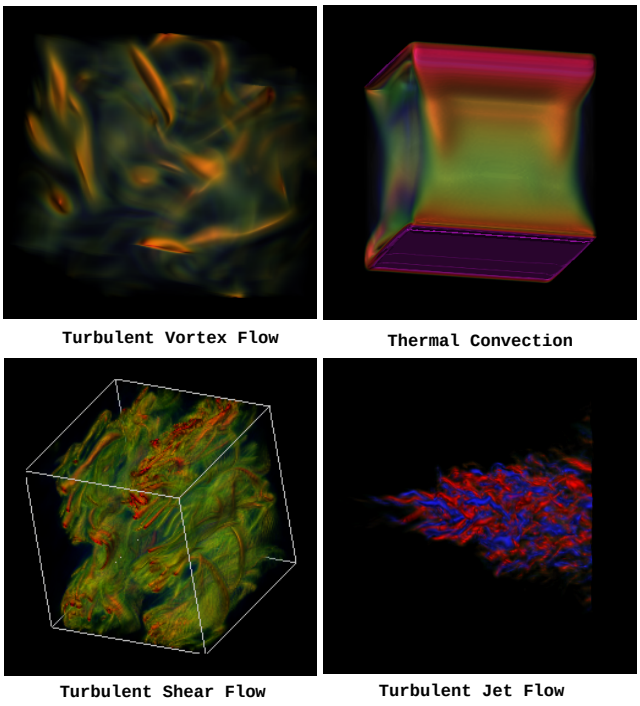

Figure 2: One selected frame for each data set.

general study. Finally, while Westermann's approach is theoretically sound, there are many computational cost issues needed to be further investigated before his approach can be utilized in practice.

3 Test Data Sets

Four data sets were used for our study. Table 1 lists the name and size of each data set. We chose small data sets due to the large number of tests we needed to perform. The vortex flow data set was obtained from pseudo-spectral simulations of coherent turbulent vortex structures. The second data set derived from a parallel three-dimensional thermal convective model and it represents the normalized temperature distribution in a closed environment when one side of the volume is heated by a constant heat source. The turbulent jets data set was generated from the modeling of naturally developing and forced jets with rectangular cross-section and different inlet conditions. The turbulent shear flow data set was obtained from a study of the generation and evolution of turbulent structures in shear flows.

Figure 1 presents histograms generated from the four data sets. In each plot, x axis is data value and y axis is the number of voxels. These plots showing the distribution of data values help the following discussions. Figure 2 shows one selected frame from each corresponding data set in Figure 1. Note that the use of different transfer functions would lead to very different visualization results.

4 Encoding and Rendering

How to efficiently encode a time-varying data set is an interesting research problem. We would like to compress the data set as much as possible while retaining the most relevant information in both the temporal and spatial domains of the data. To design a representation, we should also take into account the needed operations (i.e. visualization/rendering calculations) on the representation.

Data compression continues to be an important research topic because of its relevance to multimedia and web applications. Many compression techniques have been well studied and may be applied to new applications according to data characteristics and certain requirements. There are lossless and lossy compression methods. Popular compression techniques include huffman coding, scalar/vector quantization, differential encoding, subband coding, and transform coding [9].

Frequently, scientists demand lossless methods to preserve the accuracy of their original results. However, when performing data visualization, limited by the display technology and the implementation of rendering algorithms, degradation in image quality cannot be totally avoided. The questions are then: how lossy can the compressed data be to generate the highest possible accuracy in the visualization results with the given rendering and display technology; and how can the errors due to compression be quantified in the data and the resulting visualization?

Volume data generally come with 8-, 16-, or 32-bit voxels. Most volume renderer implementations use table lookup for color and opacity mapping. Color values are represented by red, green, and blue components, each of which is an 8-bit value. The color table thus typically consists of 256 entries of RGB values. For voxels represented with more than eight bits, quantization must be done which results in lossy compression. How quantization is done determines what in the data can be visualized.

4.1 Quantization

Quantization is the simplest lossy compression method. The idea of quantization is to use a limited number of bits to represent a much large number of distinct raw data values. The class of data sets we consider are typically generated from numerical simulations and quantization of the data results in a compression ratio of 4 : 1 by representing 32-bit data with only 8 bits. Quantization is a well studied area. However, the impact of data quantization to volume rendering has not been carefully studied.

There are uniform, non-uniform and adaptive quantizers designed according to the characteristics of the source data. For the simplest case, that is uniform quantization of uniformly distributed source data values x, the quantization error may be measured as the *mean squared error,* which is

$$\sigma^2 = \sum_{i=1}^{M} \int_{(i-1)\phi}^{i\phi} (x - \frac{2i-1}{2}\phi)^2 f(x) dx \qquad (1)$$

where M is the number of quantization levels, $\phi = (x_{max} - x_{min})/M$ and f(x) the probability density function which is $\frac{1}{x_{max} - x_{min}}$ for uniformly distributed source data. While the general principle of quantization is to reduce this data distortion error, for visualization tasks, an even more important criterion is to preserve and enhance particular features in the data. Data values outside the range of interest and the corresponding distortion error can be ignored. With a given number of quantization levels, enhancement can be achieved by allocating more levels to a particular range of the source data values. While most renderers use uniform quantization by default, non-uniform and adaptive quantization can more effectively minimize distortion error and enhance data for detecting features. For volume rendering to also include an error measure for the importance of data values, Equation 1 becomes

$$\sigma^2 = \sum_{i=1}^{M} \int_{(i-1)\phi}^{i\phi} (x - \frac{2i-1}{2}\phi)^2 f(x)\alpha(x) dx \qquad (2)$$

where $f(x)$ characterizes a general source data distribution and $\alpha(x)$ is the importance function which in this case is the opacity transfer function provided by the user.

For example, a simple non-uniform quantizer may use a logarithmic function for source data values spreading in a wide dynamic range. A more elaborate quantizer may take source data statistics (e.g. the probability density function) into consideration and set quantization levels adaptively. Figure 3 plots the maximum and minimum values for each time step of two data sets. The left one shows values of a turbulence flow data set that consists of 81 time steps. Such a data set must be quantized with care; otherwise, many important features in later time steps would become invisible due to the extremely wide dynamic range. The other data set shown on the right behaves very differently so it can be quantized in a straightforward manner.

4.2 Octree Encoding

After quantizing, each time step of the quantized data is then organized hierarchically in its spatial domain using octree encoding. Octrees are a family of data structures that represents spatial data using recursive subdivision. They have wide application to many graphics and visualization problems for faster searching, data packing, and algorithmic

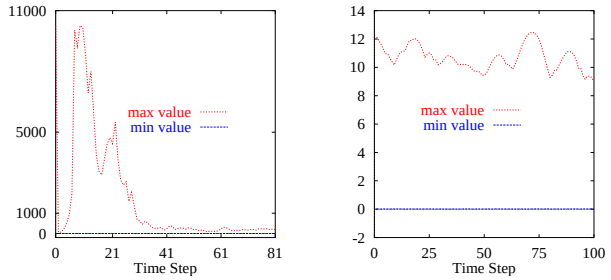

Figure 3: Left: maximum and minimum values at each time step of a data set from the study of the generation and evolution of turbulent structures in shear flows. Early time steps contain values in a very large dynamic range which makes quantization more difficult. Right: maximum and minimum values at each time step of a data set from the study of coherent turbulent vortex structures. This data set has a small dynamic range and the distribution of values is quite uniform which makes quantization straightforward.

optimization. Levoy [5] used a binary octree to skip transparent voxels for efficient volume ray casting. Laur and Hanrahan [4] implemented a hierarchical splatting renderer using octrees. Wilhelms and Van Gelder [13] used octrees with a branch-on-need (BON) strategy for faster isosurface generation, and later extended their octrees and BON strategy for k dimensions [14] of volume data for controlled compression and rendering, as we described previously in Section 2.

It is thus possible to use octree encoding for controlling compression, rendering, and image quality of time-varying volume data. With octree encoding, immediate neighboring voxels with identical values may be fused to form a macro voxel. This fusing process is performed recursively either in a top-down or a bottom-up manner until no more voxels or macro voxels can be merged. For an N-time-step volume data set, the results are N octrees. The amount of compression that can be achieved with octree encoding is data dependent. A data set containing many large, coherent structures usually can be effectively compressed. However, for 8-bit data, we found that further fusing of voxels based on some error tolerance produced images generally not acceptable for visualization. Some error control issues are discussed in [4, 14]

When using octree encoding, a bottom-up algorithm is more computationally efficient since it only visits each data value one time and avoids recalculating evaluation data. According to our test results, the bottom-up method is about two times faster than the top-down method. The space overhead of the octree encoding is generally acceptable as long as many large macro voxels are created. The maximum overhead is only about $\frac{vb}{7}$ where v is the total number of voxels in the data and b is the number of bytes used to store information about each internal tree node. Using a linear octree, it takes as few as 1 bit for each node to indicate if it is a leaf node or not. It is also needed to store values such as the minimum, maximum and mean data values which characterize the data and can be used to optimize rendering.

4.3 Difference Encoding

Like video and speech data, time-varying volume data are highly correlated from time step to time step. Difference encoding uses this fact to predict each sample based on its past, and to encode and transmit the differences between the pre-

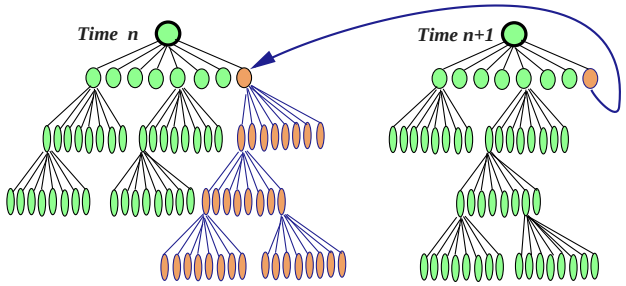

Figure 4: Merging Encoded Trees. Trees at consecutive time steps contain identical subtrees so the second time step only stores a pointer to the first time step for that subtree (red).

diction and the sample value. Further compression is built around this premise. In essence, each individually octree encoded volume may be partially merged with the one in the previous time step using difference encoding. The merging is incremental over the time dimension. Figure 4 shows how a subtree which has not changed may be represented by the one from the previous time step to save storage space.

The most interesting use of the tree structure is that when animating in the temporal domain we can waive the rendering of a subtree that has been rendered in previous time step. The image corresponding to the subtree is retrieved from the previous time step and composited into the final image of the current time step. The associativity of the *over* operation [8] for compositing guarantees the correctness of the composited results.

4.4 Encoding of the Test Data sets

Table 2 summarizes the encoded results due to different quantizations. The percentage of savings shown here is relative to the quantized data, not the raw data. The vortex data set does not include every time step of the simulation. In addition, the data values spread across the spatial domain quite uniformly. Uniform quantization brings out most features in the data. However, there is very little temporal and spatial coherence in the data set and consequently the compression rate is low. Enhancing a subset of the data values such as the high values with non-uniform quantization increases the compression rate.

In contrast, uniform quantization does not work very well for the thermal data set to discern fine features in the data. Two nonuniform quantizations focusing on different ranges of values lead to very different compression performance. We have also experimented with an adaptive quantization method which decomposes the spatial domain into subdomains and performs local quantization first to encourage voxel fusing based on local data statistics. We believe this approach will work well for some data sets, though no dramatic improvement on compression rates were obtained for our test data sets. For the shear flow data set, although the second nonuniform quantization method only achieves 40% saving, it helps bring out the most relevant structures in the data. Finally, the jets data set is best encoded with the uniform quantization which not only gives the highest compression rates but also brings out most features in the data.

We found that the quantization error as calculated by Equation 2 is less than 1% for all of our data sets. The corresponding computational cost for encoding is acceptable. For the test data sets, it takes on average about 0.5 seconds per time step to quantize and 3-5 seconds to perform octree-difference encoding on a SUN Ultra Sparc. For a data set containing 100 time steps, it takes about 5-10 minutes to encode the whole data set.

Table 2: Compression rates derived from different quantizations.

Data set	Quantization Method	Compression Percentage
Vortex	Uniform	18
	NonUniform I	71
	Adaptive	19
Thermal	Uniform	43
	NonUniform I	28
	NonUniform II	98
	Adaptive	50
Shear	Uniform	91
	NonUniform I	-7
	NonUniform II	40
Jets	Uniform	98

4.5 Rendering

The compression scheme leads naturally to a rendering strategy in which only modified data are rendered. To assess the performance, we have implemented a ray casting volume renderer, *tvvd-renderer*, which takes as input a sequence of trees, renders the first tree completely and then in subsequent timesteps renders only the modified subtrees. This requires that partial images representing the unmodified data must be retained and composited together with the partial images created from the modified data to create the final image at each time step. We do this by creating a compositing tree. The compositing tree is a pointer based octree which has the same structure as the compressed octree. Each leaf of the compositing tree contains a partial image rendered directly from the data represented by the corresponding leaf in the compressed tree. Each interior node contains a partial image which is the composite of all of its children's images. At each time step, modified subtrees in the compressed octree are identified. A new compositing branch is created to represent the data and spliced into the compositing tree, replacing the old branch. The image at the top of the new branch is composited with its siblings and all of the ancestors are recomposited to reflect the changes. The image at the root of the tree is the complete image.

Rendering only the modified data accounts for the largest savings in the time domain. Much less data (i.e. only the difference between consecutive time steps) is rendered as a result of tree merging which produces the most significant amount of savings in rendering cost. In addition, the time to read the encoded data is reduced in proportion to the compression rate.

However, rendering from the tree structure instead of directly the volume data incurs certain overhead. To offset this overhead, we use several optimizations, some of which have been discussed in [8]. First, we implemented front-to-back rendering to promote early ray termination. This optimization has been typically implemented for general ray-casting volume rendering, though the result is highly data and transfer function dependent. To reduce excessive matrix multiplication operations, we cache the coordinates of each ray in the object space. We also take advantage of the information provided by the octree structure to advance past transparent

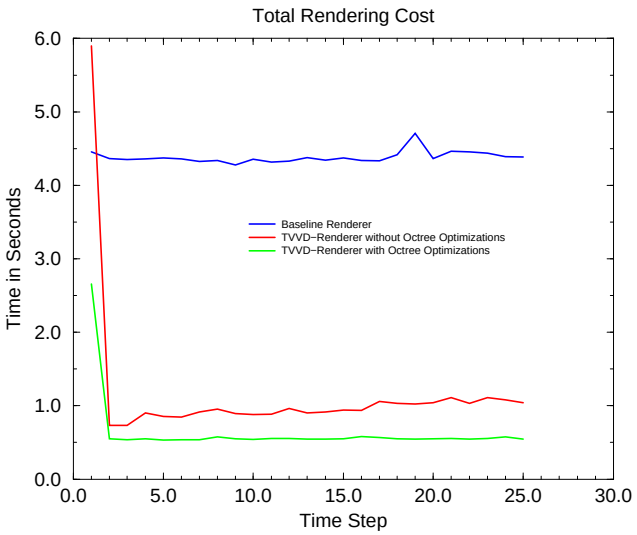

Figure 5: Rendering cost for turbulent jets data set. The time is the total time to process, including reading encoded data from disk, unencoding when necessary, calculating gradient, rendering and compositing. Overall performance gain is mostly due to tree merging while octree optimization does achieve some improvement.

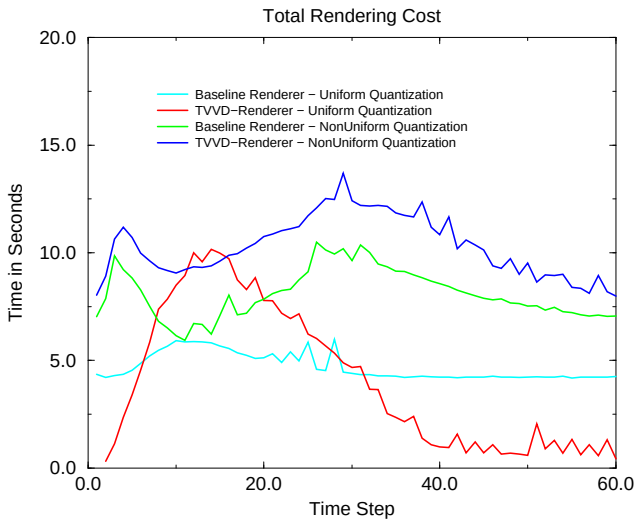

Figure 6: Rendering results for the turbulent shear flow data set.

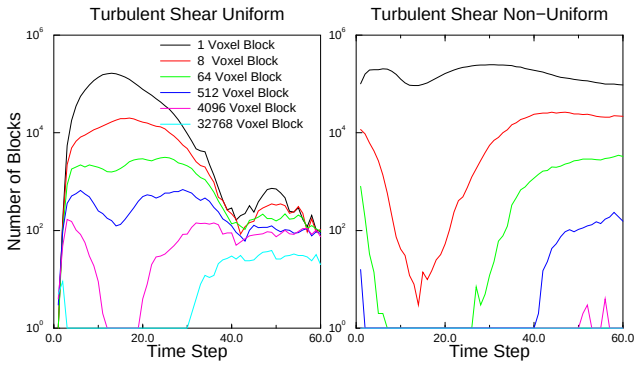

Figure 7: Number of blocks in turbulent shear flow data set.

space without rendering.

The rendering optimization is based on a fixed viewing position. Changing the viewing position requires that the entire compositing tree be regenerated. On the other hand, to allow the viewer to move randomly through the temporal domain of the data, a complete tree must be saved at regular intervals.

As expected, in many cases the rendering rate for a time-varying sequence can be greatly improved by using the compressed data. All of the timings presented are for an image size of 128×128. In this section, when we talk of rendering times, we are referring to the total cost of processing one image. That includes the time to read the data, to uncompress the data values when necessary, to calculate the gradient, update the compositing tree, render and composite.

The turbulent jet flow data set achieved the highest compression rate and the highest increase in rendering rate. For the turbulent jet flow data set, the tvvd-renderer renders the first image in 2.65 seconds and the subsequent images at an average of 0.55 seconds, which represents an increase of 80% in the rendering rate between the first and consecutive images and an 88% increase in the overall rendering rate.

Figure 5 shows three renderings of the turbulent jet flow data set. The baseline renderer renders the full data set from the volume data at each time step. The tvvd-renderer uses all of the aforementioned optimizations. The tvvd-renderer without octree optimizations uses the encoded data and builds the compositing tree, however it renders transparent space and uniform space as if they were nonuniform. Due to the transfer functions used, the turbulent jet flow data set has large regions of transparent space and also large blocks of non-transparent uniform space. This is the best case for octree optimization, but the figure shows that while some of the speedup is a result of using the octree optimizations, the majority of the speedup occurs because of the tree merging.

While the rendering rate increases dramatically when the compression rate is high, it is dependent upon the number of large blocks (4096 voxels or larger) which can be compressed. When a single voxel changes, the surrounding voxels are re-rendered. Thus, compression resulting from merging 1 voxel blocks or 8-voxel blocks is not useful at all in the rendering. Compression resulting from merging 64- and 512-voxel blocks has some effect, but the types of data sets which have many small matching blocks and few large matching blocks typically require more overhead to use the octree than can be gained by using the compression information.

An example of this is the turbulent shear data set. Figure 6 shows the rendering times for this data set using two different forms of quantization. Figure 7 shows the number of large matching blocks at each time step. Notice that at time step 30 in the uniform quantization method, the number of 32768-voxel blocks increases and there is an immediate response in the rendering time. The compression using the nonuniform quantization method is the result of a large number of small matching blocks, not a small number of large matching blocks. The renderer cannot take advantage of the compression, and the rendering rate is consistently lower. Generally, if the data are compressed by less than 50% in the time domain, unless many large subtrees were merged, little rendering performance gain can be obtained. This is consistent with the results reported in [10].

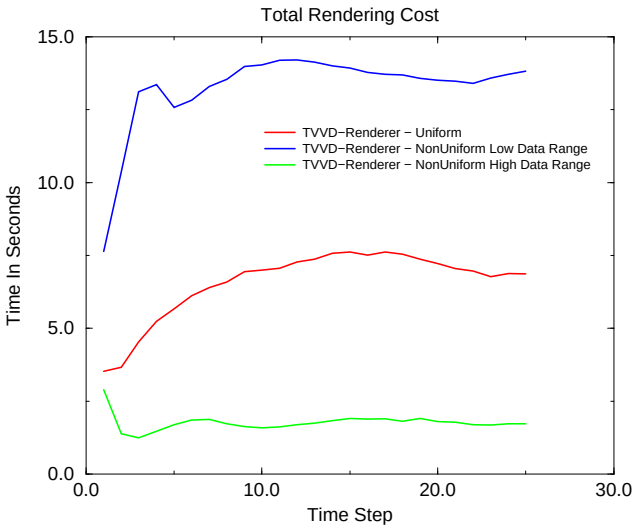

Figure 8: Rendering results Thermal Convection data set.

Quantization can be used effectively to focus on different features in the data and can affect the number of matching blocks at each time step. By choosing the area of interest carefully, a scientist is able to control not only the level of feature enhancement but also the compression and rendering times of the data. The thermal convection data set has interesting features which can be emphasized by nonuniform quantization. Figure 8 shows the effects of different methods of quantization on the rendering time.

The vortex data set can also be compressed well with non-uniform quantization, but the compression results from many small voxel blocks and not any larger blocks. Therefore, although the data set is compressed, the rendering time increases.

The core rendering code for the baseline volume renderer is the same as that used for the tvvd-renderer. It is a very basic renderer with few optimizations. Replacing the core code with a more optimized renderer will increase the rendering rate of both renderers. The tvvd-renderer can be configured to stop at any depth in the tree and render immediately. The minimum number of nodes which may be rendered is an 8-voxel block. Increasing the minimum number of nodes decreases the overhead associated with the octree but also decreases the number of matching blocks which do not have to be rerendered. The optimizations which we have incorporated into the octree renderer such as moving past transparent blocks without rendering and using front to back rendering to encourage early termination of rays are highly dependent upon the opacity maps. Using different opacity maps can dramatically change the rendering times. Rendering at 256×256 required approximately two to three times as long. For larger image size or higher interaction, the tree branches can be distributed to multiple processors to be rendered.

5 A Refined Design: Time-Space Partitioning Tree

Based on our findings, we determine that a better design ...

6 Conclusions

Visualization of time-varying data will continue to be important and challenging. We have investigated how time-varying volume data may be organized to facilitate direct volume rendering and demonstrated some promising results. In general, the selection of encoding and rendering strategies should depend very much on data resolution, statistics and visualization requirements.

We found that in many cases the amount of savings in storage space and rendering time can be tremendous while the resulting visualization results stay visually indistinguishable from high-resolution ones. This suggests that unless the display resolution and visualization requirements are high, we should take advantage of compression and multiresolution rendering to increase visualization efficiency. The savings in storage space also reduces the I/O required by the renderer. With large data sets over long intervals of time, this reduction can be a significant part of the overall savings.

Our goal is to increase the users interaction with the data. This requires that the images be presented to the user as rapidly as possible. Although we do not see large savings when the cost of quantization and rendering are combined, by preprocessing we can achieve near interactive viewing rates.

Future work includes the development of application-specific techniques and taking the grid structures (curvilinear, unstructured, etc.) into consideration. We will investigate how the order of encoding calculations would impact the overall compression and rendering performance. In addition, we will study the characteristics of time-varying computational fluid dynamics data sets and continue developing appropriate compression and rendering methods.

7 Acknowledgments

The authors would like to thank Peggy Li, John Shebalin, Deborah Silver and Robert Wilson for the test data sets.

References

[1] Chiueh, T. Z., Yang, C. K., He, T., Pfister, H., and Kaufman, A. Integrated Volume Compression and Visualization. In *Proceedings of the Visualization '97 Conference* (October 1997), pp. 329–336.

[2] Fowler, J. E., and Yagel, R. Lossless Compression of Volume Data. In *Proceedings of the 1994 Symposium on Volume Visualization* (October 1994).

[3] Freund, J., and Sloan, K. Accelerated volume rendering using homogeneous region encoding. In *Proceedings of the Visualization '97 Conference* (October 1997), pp. 191–196.

[4] Laur, D., and Hanrahan, P. Hierarchical Splatting: A Processive Refinement Algorithm for Volume Rendering. In *Proceedings of SIGGRAPH '91* (1991).

[5] Levoy, M. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics 9*, 3 (July 1990).

[6] Ma, K.-L., Cohen, M., and Painter, J. Volume Seeds: A Volume Exploration Technique. *The Journal of Visualization and Computer Animation 2* (1991), 135–140.

[7] NING, P., AND HESSELINK, L. Vector Quantization for Volume Rendering. In *Proceedings of the Visualization '93 Conference* (October 1993).

[8] PORTER, T., AND DUFF, T. Compositing Digital Images. *Proceedings of SIGGRAPH '84 18*, 3 (July 1984).

[9] SAYOOD, K. *Introduction to Data Compression.* Morgan Kaufmann Publishers, Inc., 1996.

[10] SHEN, H.-W., AND JOHNSON, C. Differential Volume Rendering: A Fast Volume Visualization Technique for Flow Animation. In *Proceedings of the Visualization '94 Conference* (October 1994), pp. 180–187.

[11] WESTERMANN, R. A Multiresolution Framework for Volume Rendering. In *Proceedings of the 1994 Symposium on Volume Visualization* (October 1994).

[12] WESTERMANN, R. Compression time rendering of time-resolved volume data. In *Proceedings of the Visualization '95 Conference* (1995), pp. 168–174.

[13] WILHELMS, J., AND VAN GELDER, A. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics 11*, 3 (July 1992).

[14] WILHELMS, J., AND VAN GELDER, A. Multi-Dimensional Trees for Controlled Volume Rendering and Compression. In *Proceedings of the 1994 Symposium on Volume Visualization* (October 1994).