Compression and Accelerated Rendering of Time-Varying Volume Data

Kwan-Liu Ma [*]
University of California, Davis

Han-Wei Shen [†]
Ohio State University

Abstract

Visualization of time-varying volumetric data sets, which may be obtained from numerical simulations or sensing instruments, provides scientists insights into the detailed dynamics of the phenomenon under study. This paper describes our study of a coherent solution based on quantization coupled with octree and difference encoding, and adaptive rendering for efficient visualization of time-varying volumetric data. Quantization is used to attain voxel-level compression and may have a significant influence on the performance of the subsequent encoding and visualization steps. Octree encoding is used for spatial domain compression, and difference encoding for temporal domain compression. In essence, neighboring voxels may be fused into macro voxels if they have similar values, and subtrees at consecutive time steps may be merged if they are identical.

The software rendering process is tailored according to the tree structures and the volume visualization process. With the tree representation, selective rendering may be performed very efficiently. Additionally, the I/O costs are reduced. With these combined savings, a higher level of user interactivity is achieved. We have studied a variety of time-varying volume data sets, performed encoding based on data statistics, and optimized the rendering calculations wherever possible. Preliminary tests on workstations have shown in many cases tremendous reduction by as high as 90% in both storage space and inter-frame delay when compared to direct rendering of the raw data.

1 Introduction

The ability to study time-varying phenomena helps scientists understand complex problems, but 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, the size of a single-variable time-varying volume data set can be easily in the range of hundreds of gigabytes.

Ideally, visualizing time-varying data should be done while data is being generated, so that users receive immediate feedback on the subject under study to achieve runtime tracking, and so the visualization results can be stored rather than the much larger raw data. However, runtime tracking is not always possible and desirable for certain applications. For example, one may want to explore the data set from different perspectives; or, the amount of computation power required for real-time rendering or a special visualization technique may not be readily available. As a result, postprocessing of pre-calculated data remains an important requirement.

This paper presents a strategy integrating compression and rendering techniques to achieve flexible and efficient rendering of time-varying volume data as a postprocess. Although volume data compression has been studied by many researchers [1, 13, 3, 4, 8], 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. Figure 1 shows the overall encoding and rendering process.

Data sets with quite different different characteristics were used for our study. We show how each data set 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.

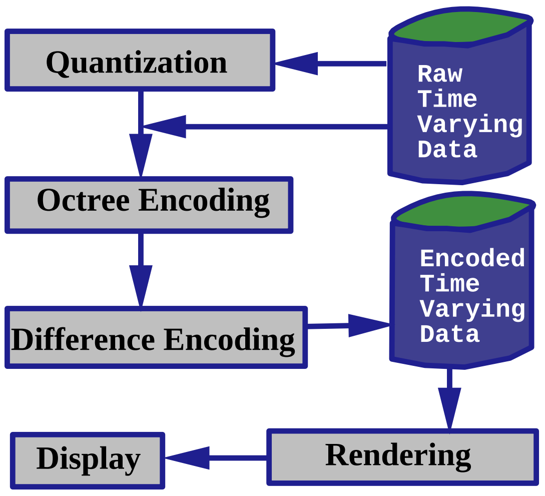

Figure 1: Overall Encoding and Rendering Process.

2 Related Work

The previous work most closely related to ours is the thorough study done by Wilhelms and Van Gelder [16] on the design of hierarchical data structures for controlled compression and volume rendering. They extend octrees and a branch-on-need (BON) subdivision strategy [15] 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 several evaluation metrics for performing selective traversal and visualization of the encoded data.

Another closely related work is the ray-cast rendering strategy introduced by Shen and Johnson [12] which they call *differential*

[*]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.

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.

Based on similar concepts, Westermann [14] 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.

More recently, Shen, Chiang, and Ma [11] introduced a hierarchical data structure, called Time-Space Partitioning (TSP) tree, for a better utilization of both spatial and temporal coherence. In essence, 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. To store the temporal information, each TSP tree node itself is a binary tree. Every node in the binary time tree represents a different time span for the same subvolume in the spatial domain. The focus of the algorithm is to reduce the amount of data required to complete the rendering task and to reduce the volume rendering time. In particular, TSP trees allow the renderer to use data from subvolumes of different spatial and temporal resolutions. The TSP tree data structure has been also used by Ellsworth, Chiang, and Shen [2] to facilitate large scale volume rendering using 3D texture hardware.

This paper focuses on how quantization might affect futher compression, rendering optimization, and image results of time-varying volume data. Similar to Wilhelms and Van Gelder's work, we use octree encoding and error evaluation for selective traversal. But in contrast to theirs, we apply difference encoding to the time domain while their model treats all dimensions the same way. We favor separating the temporal and spatial domains for encoding unless the disparity in resolution (and coherence) between the spatial domain and temporal domain is very small, which is unlikely in practice.

We have developed 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. This approach has also been applied by Shen, Chiang, and Ma for the TSP tree. 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 general study. Furthermore, we consider raw data sets (i.e. typically collections of floating-point values) directly rather than the quantized ones (i.e. byte data or integers). Finally, while Westermann's approach is theoretically sound, there are many computational cost issues remained to be investigated before his approach can be utilized in practice.

3 Test Datasets

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.

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	$128^3 float$

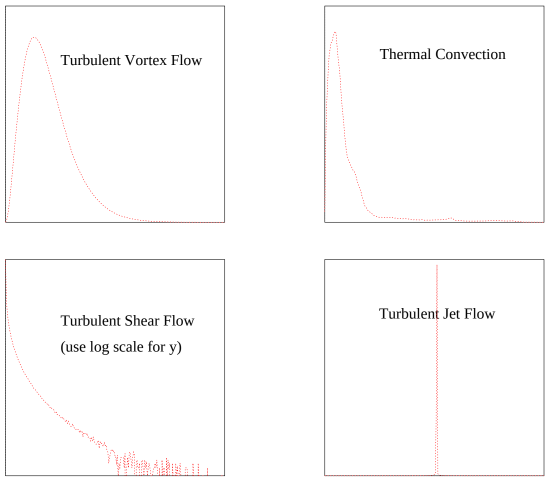

Figure 2: Histograms of the data sets. Each plot shows the distribution of data values in the whole data set.

Figure 2 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 3 shows one selected frame from each corresponding data set in Figure 2. Note that the use of different transfer functions would lead to very different visualization results.

4 Compression

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 [10].

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 de-

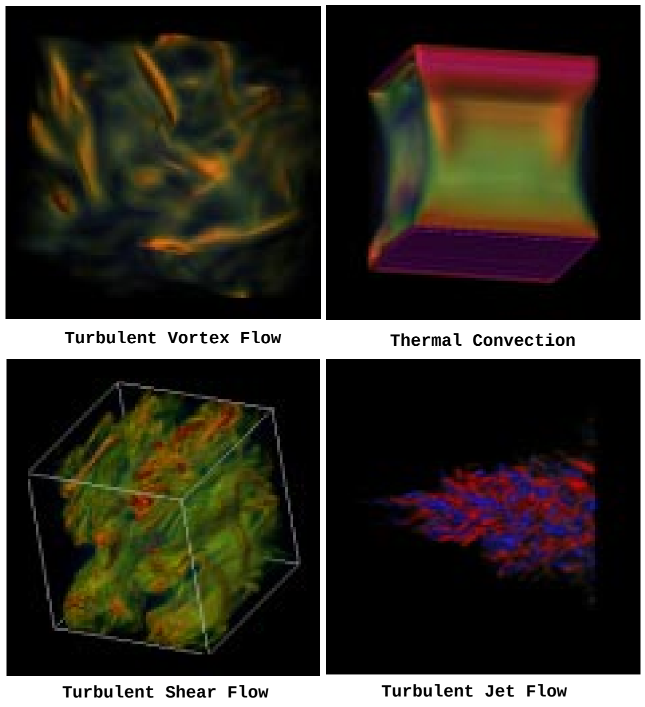

Figure 3: One selected frame for each data set.

termines 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

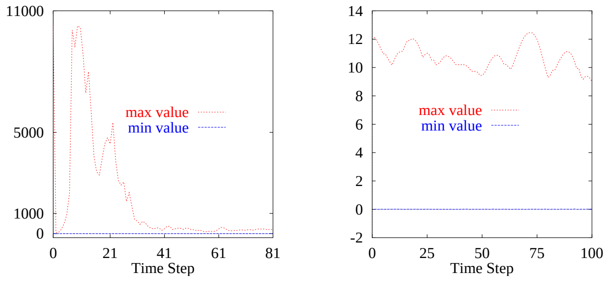

Figure 4: Left: maximum and minimum values at each time step of a data set from the study of the generation and evolution of turbulent structures in shear flows. Early time steps contain values in a very large dynamic range which makes quantization more difficult. Right: maximum and minimum values at 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.

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 4 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 optimization. Levoy [7] used a binary octree to skip transparent voxels for efficient volume ray casting. Laur and Hanrahan [6] implemented a hierarchical splatting renderer using octrees. Wilhelms and Van Gelder [15] used octrees with a branch-on-need (BON) strategy for faster isosurface generation, and later extended their octrees and BON strategy for k dimensions [16] of volume data for controlled compression and rendering, as we described previously in Section 2.

We use octree encoding to control 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

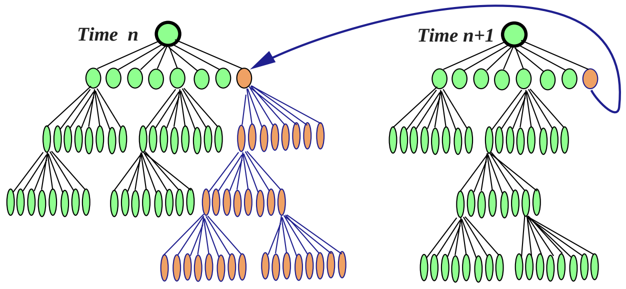

Figure 5: Merging Encoded Trees. Trees at consecutive time steps contain identical subtrees so the second time step only stores a pointer to the first time step for that subtree (red).

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 [6, 16]

Our octree encoding uses a bottom-up algorithm which only visits each data value one time and avoids recalculating evaluation data and is therefore more computationally efficient. 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. We also 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 prediction and the sample value. Our 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 5 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 [9] for compositing guarantees the correctness of the composited results. The details of the rendering step will be described in Section 5.

4.4 Optimization

For quantization, the choice of bit allocation significantly affect not only the subsequent encoding results but also the visualization results. That is, a particular quantization can result in more voxel fusing and thus higher compression and rendering rates. After seeing the corresponding visualization, if the scientist determines that quantization needs to be redone to emphasize a particular range of data, the octree and difference encoding must also be redone. Since data exploration is an inherently iterative process, we want to keep the cost of quantization and subsequent encoding as low as possible.

When the data for each time-step is very large and I/O cost becomes significant, a good strategy is to overlap encoding and I/O. We have also mentioned that certain algorithmic advantages such as using bottom-up tree construction can make a difference in the overall cost. Finally since most of the calculations for each time step is performed independently of other steps, multiple time-step data can be encoded concurrently by using a cluster of workstations.

Table 2: Compression rates (due to both octree and difference encoding) derived from different quantizations. Note that the percentage of savings shown here is relative to the quantized data, not the raw data.

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 Test Resutls

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 low-end SUN Ultra Sparc. For a data set containing 100 time steps, it takes about a few minutes to encode the whole data set.

5 Rendering

The compression scheme leads naturally to a rendering strategy in which only modified data are rendered. 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 space without rendering.

Additionally, when an octant representing a subvolume has a constant value everywhere in its domain, the rendering of the corresponding subvolume can be, though not waived, highly optimized. Discretizing the volume rendering integral equation, the accumulated color value up to n sample points on a ray is represented as:

$$C = \sum_{i=1}^{n} C(i)\alpha(i)\prod_{j=1}^{i-1}(1-\alpha(j)) \tag{3}$$

For a constant subvolume, since all sample points have an identical data value and therefore identical color and opacity values, the formulation for compositing can then be simplified to:

$$\begin{aligned} C &= \sum_{i=1}^{n} C\alpha\prod_{j=1}^{i-1}(1-\alpha) \\ &= \sum_{i=1}^{n} C\alpha(1-\alpha)^{i-1} \qquad (4) \end{aligned}$$

With this derivation, we only need to know the number of samples that should be collected along a ray. The calculations of the sample coordinates and trilinear interpolation of the sample values along each ray can be completely avoided. The resulting saving is tremendous for a data set containing many large, coherent structures.

In the octree, each leaf represents a uniform block of data which can be rendered efficiently as discussed above. However, the boundaries between the uniform blocks must be rendered more carefully. To avoid the overhead of traversing the tree to obtain boundary values, the data is initially uncompressed and the octree information is used as a map into the volume data.

Because of opacity accumulation fine details at the front parts of the volume often obscure the back. This means that when doing front-to-back rendering, subtrees which represent the back portion of the data may not be completely sampled. As an approximation, we do not re-render the subtrees which have not changed between time steps.

We can also improve performance by rendering data at different resolutions in different areas of the spatial domain. Figure 6 displays visualization results generated based on this strategy. Image (a) is a regular rendering result. Image (b) shows the result of skipping pixels in image space and the blocky pattern hampers normal perception of the image content. Images (c) and (d) show results from various degrees of coarsening in the spatial domain. Coarsening was done by fusing voxels with high tolerance values. Image (c) and (d) are the results of treating a block of voxels identically if the difference between the maximum and minimum voxel values is under some user-specified tolerance. The resulting savings in both storage space and rendering time are quite dramatic. We achieve 40% saving for (c) and 90% for (d) in storage space. Image (c) is almost visually indistinguishable from Image (a). Image (d) is less visually appealing but it is good for previewing of the data.

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.

5.1 Test Results

As expected, in many cases the rendering rate for a time-varying sequence can be greatly improved by using the compressed data. Because the large number of tests we needed to perform, all of the timings presented are for an image size of 128$\times$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.

Our ray-casting volume renderer is is reasonably optimized and capable of generating high quality images. There are other volume rendering algorithms such as the shear warp algorithm [5] that can deliver superior rendering rates. Since our task is to render time-varying data, the preprocessing calculations required by the shear warp algorithm must be done for every time step. This requirement makes the shear warp algorithm less attractive. Including the preprocessing time for each time step, a shear-warp image and a ray-cast image could take about the same amount of time to generate. In addition, due to the use of 2-d filtering, the quality of a shear warp image, in some case, could be less ideal.

The heart and turbulent jet flow data sets 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. For the heart data set, we saw a 93% increase in the overall rendering rate.

Figure 7 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 optimizations discussed in Section 5. The tvvd-renderer without octree optimiza-

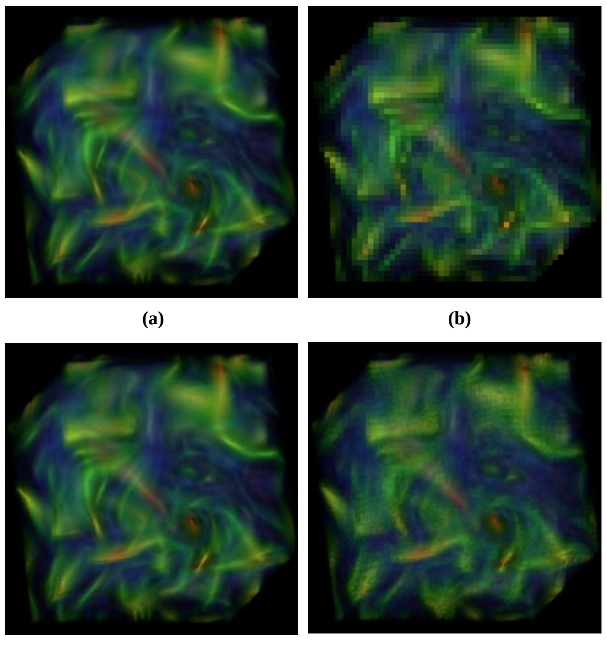

Figure 6: Rendering data at various resolution in various space. (a) regular rendering. (b) rendering at lower resolution in image space. (c) rendering at slightly lower resolution in the data domain which produces about 40% saving in storage space and 10% in rendering time. (d) rendering at much lower resolution in the data domain which produces about 90% saving in storage space and 30% in rendering time.

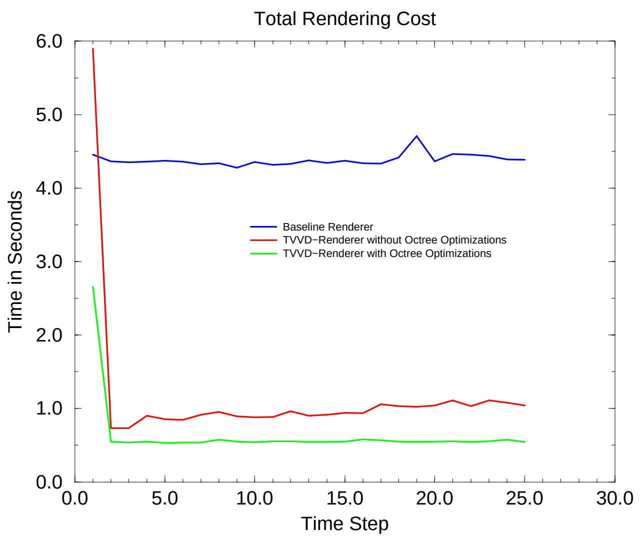

Figure 7: Rendering cost for turbulent jets data set. The time is the total time to process, including reading encoded data from disk, unencoding when necessary, calculating gradient, rendering and compositing. Overall performance gain is mostly due to tree merging while octree optimization does achieve some improvement.

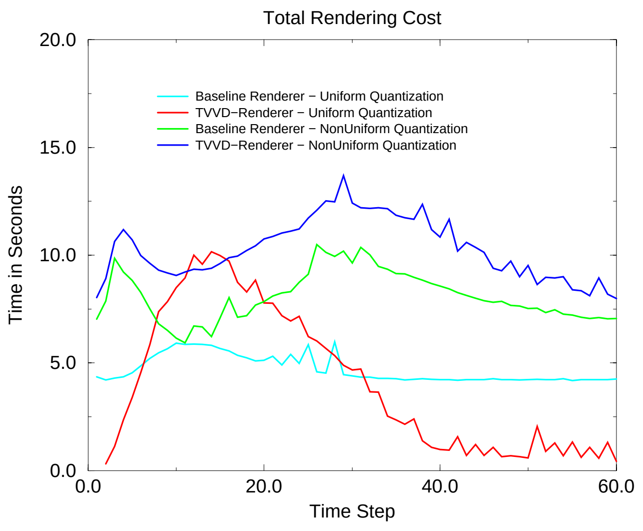

Figure 8: Rendering results for the turbulent shear flow data set.

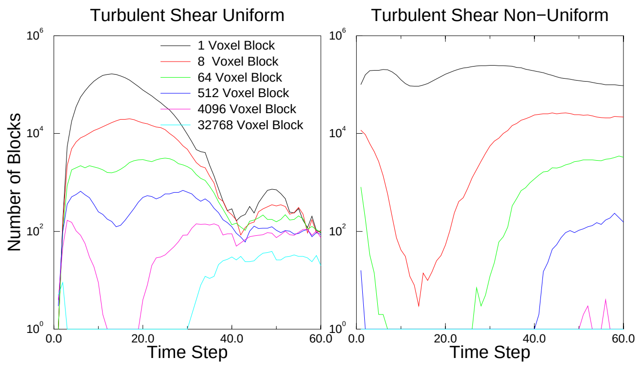

Figure 9: Number of blocks in turbulent shear flow data set.

tions 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 8 shows the rendering times for this data set using two different forms of quantization. Figure 9 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

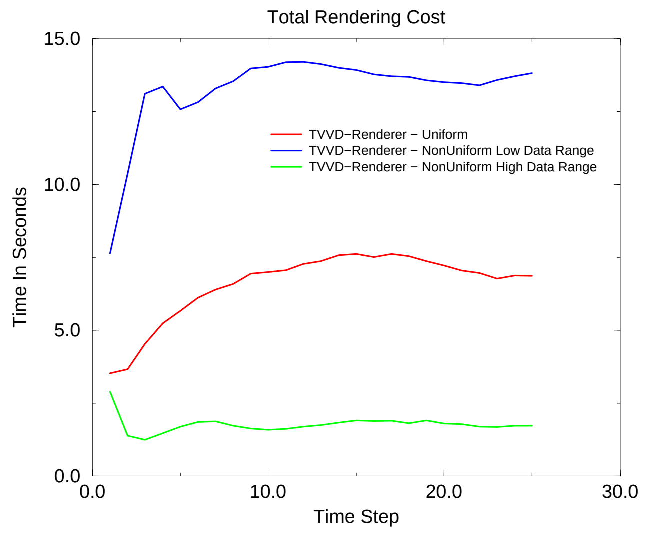

Figure 10: Rendering results Thermal Convection data set.

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 [12].

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 10 shows the effects of different methods of quantization on the rendering time.

The vortex data set can also be compressed well with nonuniform 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 our 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.

6 Conclusions

Visualizing 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 with a large number of time steps, this reduction can be a significant part of the overall savings.

An important goal of our study is to allow a more interactive user interrogation 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.

Our results also showed that utilizing both the spatial and temporal coherence existing in a time-varying dataset can be very effective in reducing the data size and rendering time. While the work presented in this paper is based on the use of a sequence of 3D octrees, we believe a more flexible data structure for 4D datasets is needed. We are currently investigating on the use of 4D octrees, or called 16-trees, and TSP trees for further optimizing the quantization and rendering process. Our preliminary study has showed that 16-trees is suboptimal in capturing the temporal coherence. This is because in 16-trees, the temporal domain is tightly coupled with other spatial dimensions during the hierarchical subdivision. For instance, for a $32 \times 32 \times 32$ volumetric dataset with 32 time steps, the first level of subdivision divides each dimension in half and thus subvolumes of this level have a spatial dimension of $16 \times 16 \times 16$ and the time interval of sixteen. This implies that for the $16 \times 16 \times 16$ subvolumes, temporal coherence can be detected only in the interval of sixteen time steps but not the others. In addition, subvolumes with only temporal coherence but no spatial coherence, can not be detected at all using 16-trees as the spatial and temporal coherence are always considered together. It appeared to us that the TSP tree data structure can be more effective, although the work presented in [11] is primarily focused on improving rendering speed but not data compression. We plan to further the study of uinsg TSP trees for effective quantization and data compression.

Other 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

This research was supported in part by the National Science Foundation under contract ACI 9983641 (CAREER Award), and through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251. It was also supported in part by the National Aeronautics and Space administration under NASA Contract No. NAS1-97046 while the authors in residence at the Institute for Computer Applications in Science and Engineering (ICASE). The authors would like to thank Peggy Li, John Shebalin, Deborah Silver and Robert Wilson for making test data sets available.

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] Ellsworth, D., Chiang, L., and Shen, H.-W. Accelerating time-varying hardware volume rendering using tsp trees and color-based error metrics. In *Proceedings of 2000 Symposium on Volume Visualization* (2000), ACM SIGGRAPH.

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

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

[5] Lacroute, P., and Levoy, M. Fast volume rendering using a shea-warp factorization of the viewing transformation. In *SIGGRAPH '94 Conference Proceedings* (1994), ACM SIGGRAPH, pp. 451–458.

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

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

[8] Ning, P., and Hesselink, L. Vector Quantization for Volume Rendering. In *Proceedings of the Visualization '93 Conference* (October 1993).

[9] Porter, T., and Duff, T. Compositing Digital Images. *Proceedings of SIGGRAPH '84 18*, 3 (July 1984).

[10] Sayood, K. *Introduction to Data Compression*. Morgan Kaufmann Publishers, Inc., 1996.

[11] Shen, H.-W., Chiang, L., and Ma, K. A fast volume rendering algorithm for time-varying field using a time-space partitioning (tsp) tree. In *Proceedings of Visualization '99* (1999), IEEE Computer Society Press, Los Alamitos, CA.

[12] 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.

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

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

[15] Wilhelms, J., and Van Gelder, A. Octrees for Faster Isosurface Generation. *ACM Transactions on Graphics 11*, 3 (July 1992).

[16] 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).