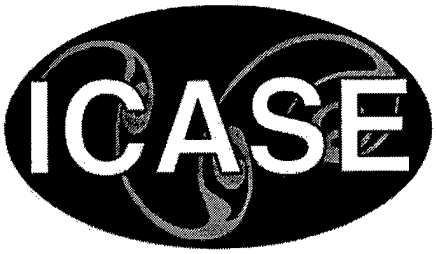
NASA/CR-1998-208424
ICASE Report No. 98-22



Efficient Encoding and Rendering of Time-Varying Volume Data

Kwan-Liu Ma, Diann Smith, and Ming-Yun Shih
ICASE, Hampton, Virginia

Han-Wei Shen
MRJ Technology Solutions, Moffett Field, California

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

June 1998

EFFICIENT ENCODING AND RENDERING OF TIME-VARYING VOLUME DATA*

KWAN-LIU MA†, DIANN SMITH†, MING-YUN SHIH†, AND HEN-WEI SHEN‡

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 a coherent solution based on quantization, coupled with octree and difference encoding for visualizing 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 datasets, 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.

Key words. time-varying data, data compression, hierarchical data structures, volume rendering, interactive visualization, distributed computing, scientific visualization

Subject classification. Computer Science

1. Introduction.

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

Ideally, visualizing time-varying data should be done while data is being generated, so that users receive immediate feedback on the subject under study, and so the visualization results can be stored rather than the much larger raw data. Rowlan [14] and Ma [9] demonstrate such tracking capability using direct volume rendering on a massively parallel computer. Some visualization software systems [4, 12] can support runtime tracking of three-dimensional numerical simulations and they may be operated in a distributed computing environment. 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 research was supported by the National Aeronautics and Space administration under NASA Contract No. NAS1-97046 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE).
kma@icase.edu, diann@icase.edu, ming@icase.edu, hwshen@nas.nasa.gov.

†Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA 23681-2199.

‡MRJ Technology Solutions at NASA Ames Research Center, Moffett Field, CA 94035-1000.

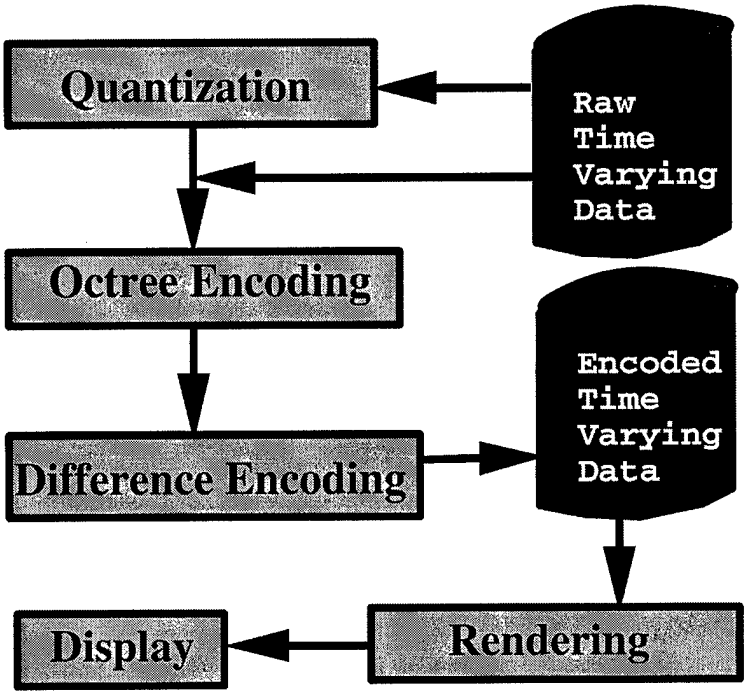


FIG. 1. *Overall Encoding and Rendering Process.*

Several techniques have been developed for visualizing time-varying data as a postprocess. Lane [6] developed a particle tracer for three-dimensional time-dependent flow data. Max and Becker [11] apply textures for visualizing both steady and unsteady flow fields. Silver and Wang [17] present a volume based feature tracking algorithm to help visualize and analyze large time-varying data sets. More recently, Jaswal demonstrates distributed real-time visualization of time-varying data using a CAVE [5]. He identifies I/O as the single most constraining factor in the level of interactivity and suggests performing various types of filtering to reduce the amount of data sent and rendered.

While parallel volume rendering algorithms are available for interactive visualization of large volume data, visualizing time-varying data on a parallel computer requires reading large files continuously or periodically throughout the course of the visualization process. Chiueh and Ma [1] developed a parallel pipelined renderer for time-varying volume data by partitioning processors into groups to render multiple volumes concurrently. In this way, the overall rendering time may be greatly reduced because the pipelined rendering tasks are overlapped with the I/O required to load each volume into a group of processors; moreover, parallelization overhead may be reduced as a result of partitioning the processors. They demonstrated that the ideal partitioning number leading to optimal performance can be determined.

This paper presents a strategy 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 [2, 18, 3], 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. Figure 1 shows the overall encoding and rendering process.

Five time-varying volume datasets were used for our study. We show how each dataset 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. Preliminary tests show that in general the amount of savings we can obtain in storage space as well as in rendering time justifies our approach.

2. Related Work. The previous work most closely related to ours is the thorough study done by Wilhelms and Van Gelder [20] on the design of hierarchical data structures for controlled compression and volume rendering. They extend octrees and a branch-on-need (BON) subdivision strategy [19] 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 datasets 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 [16] 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 [10] may be integrated into the renderer to avoid recalculations for visualizing irregular data.

Although we also use octree encoding and error evaluation for selective traversal, our main focus is on time-varying volume data (i.e. four-dimensional data.) While Wilhelms and Van Gelder's 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 favoring 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. Finally, in contrast to [16], we use datasets with distinct properties which are not all highly spatially and temporally coherent in order to perform a more general study.

3. 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 [15].

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.

3.1. 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 datasets 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

$$\sigma^2 = \sum_{i=1}^{M} \int_{(i-1)\phi}^{i\phi} (x - \frac{2i-1}{2}\phi)^2 f(x) dx \tag{3.1}$$

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 3.1 becomes

$$\sigma^2 = \sum_{i=1}^{M} \int_{(i-1)\phi}^{i\phi} (x - \frac{2i-1}{2}\phi)^2 f(x)\alpha(x) dx \tag{3.2}$$

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 2 plots the minimum and maximum values for each time step of two datasets. 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.

3.2. 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 [8] used a binary octree to skip transparent voxels for efficient volume ray casting. Laur and Hanrahan [7] implemented

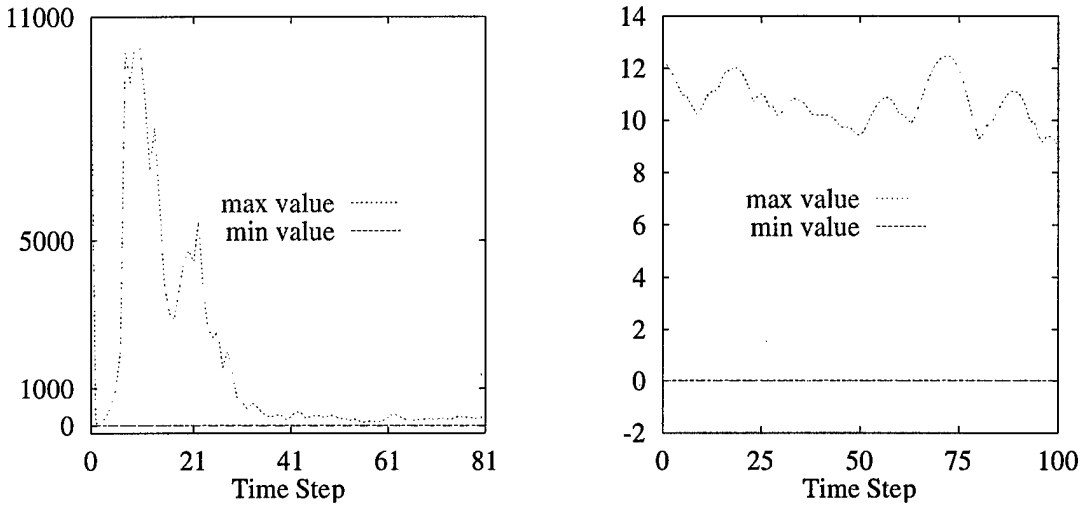


FIG. 2. *Left: maximum and minimum values at each time step of a data set from the study of the generation and evolution of turbulent structures in shear flows. Early time steps contain values in a very large dynamic range which makes quantization more difficult. Right: maximum and minimum values at 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.*

a hierarchical splatting renderer using octrees. Wilhelms and Van Gelder [19] used octrees with a branch-on-need (BON) strategy for faster isosurface generation, and later extended their octrees and BON strategy for k dimensions [20] 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 dataset, 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 [7, 20]

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 could take as few as 1 bit for each node to indicate if it is a leaf node or not. We could also store values which characterize the data such as the minimum, maximum and mean data values. This could be used to optimize rendering.

3.3. 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 individualy octree encoded volume may be par-

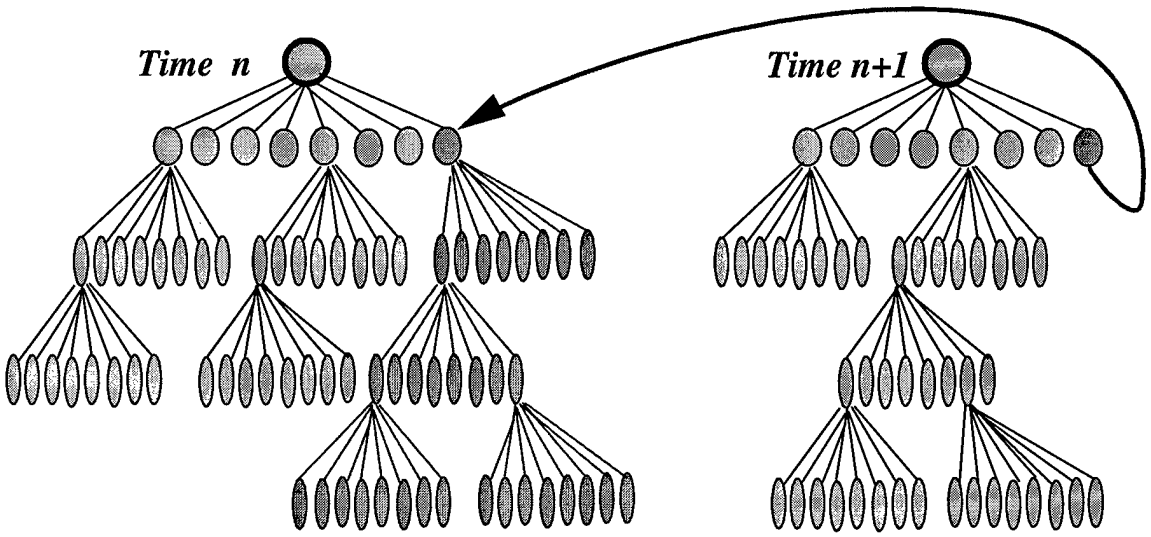


FIG. 3. *Merging Encoded Trees. Trees at consecutive time steps contain identical subtrees so the second time step only stores a pointer to the first time step for that subtree (red).*

tially merged with the one in the previous time step using difference encoding. The merging is incremental over the time dimension. Figure 3 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 [13] for compositing which is also the basis of many parallel volume rendering algorithm guarantees the correctness of the composited results. The details of the rendering step will be described in Section 4.

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

When the data for each time-step is very large, I/O cost can be significant and overlapping encoding and I/O should be implemented. 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 can be performed independently of other steps, multiple time-step data may be encoded concurrently by using a cluster of workstations.

4. 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:

$$C = \sum_{i=1}^{n} C(i)\alpha(i) \prod_{j=1}^{i-1} (1 - \alpha(j)) \tag{4.1}$$

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:

$$\begin{aligned} C &= \sum_{i=1}^{n} C\alpha \prod_{j=1}^{i-1} (1 - \alpha) \\ &= \sum_{i=1}^{n} C\alpha(1 - \alpha)^{i-1} \end{aligned} \tag{4.2}$$

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.

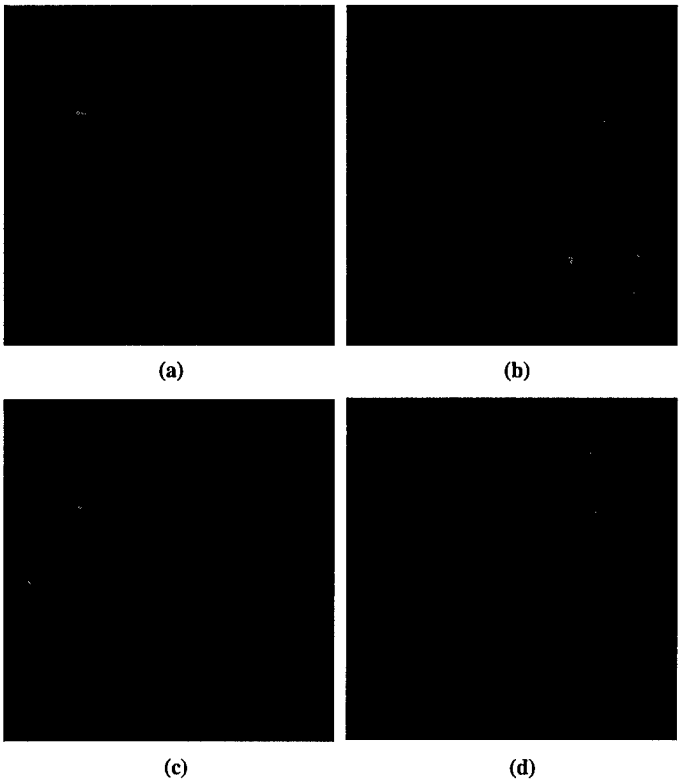


FIG. 4. *Rendering data at various resolution in various space. (a) regular rendering. (b) rendering at lower resolution in image space. (c) rendering at slightly lower resolution in the data domain which produces about 40% saving in storage space and 10% in rendering time. (d) rendering at much lower resolution in the data domain which produces about 90% saving in storage space and 30% in rendering time.*

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 4 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 tree be rerendered creating a new compositing tree. To allow the viewer to move randomly through the temporal domain of the data, a complete tree could be saved at regular intervals.

TABLE 1
Five Test Datasets.

dataset	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	$128^3 float$
Heart Modeling	100	$128^3 byte$

5. Test Results. Five datasets were used for our study. Table 1 lists the name and size of each dataset. The vortex flow dataset was obtained from pseudo-spectral simulations of coherent turbulent vortex structures. The second dataset 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 dataset was generated from the modeling of naturally developing and forced jets with rectangular cross-section and different inlet conditions. The turbulent shear flow dataset was obtained from a study of the generation and evolution of turbulent structures in shear flows. The heart dataset was obtained from simulating the electrical impulse conduction in the heart using an anatomically accurate cellular automation model. The purpose of including the heart dataset is to compare with previous results [16]. The data consist of the state histories of all the elements in the model over the duration of the simulation.

Figure 5 presents histograms generated from first four of the five datasets. 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 6 shows one selected frame from each corresponding dataset in Figure 5. Note that the use of different transfer functions would lead to very different visualization results. For example, the vortex image here looks quite different from the one in Figure 4.

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 dataset 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 dataset 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 dataset to discern fine features in the data. Two nonuniform quantizations focusing on different ranges of values lead to very different compression performances. We have also experimented with an adaptive quantization method which decomposes the spatial domain into subdomains and performs local quantization first to encourge voxel fusing based on local data statistics. We believe this approach will work well for some datasets, though no dramatic improvement on compression rates were obtained for our test datasets. For the shear flow dataset, although the second nonuniform quantization method only achieves 40% saving, it helps bring out the most relevant structures in the data. Finally, the jets dataset 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 datasets. The corresponding computational cost for encoding is acceptable. For the test datasets, it takes on average about 0.5 seconds per time step to quantize and 3-5 seconds to perform octree-difference encoding on a

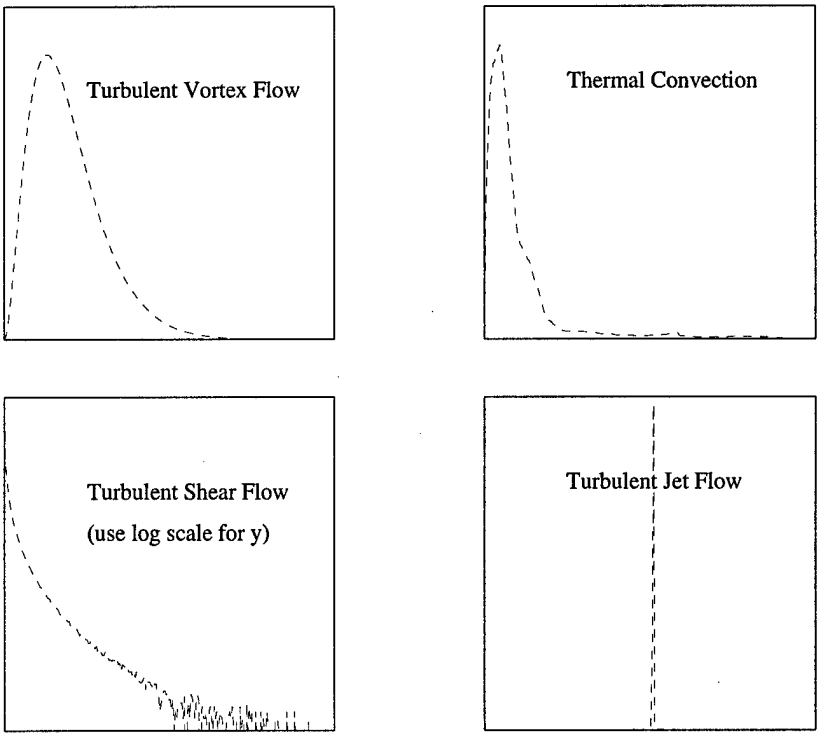


FIG. 5. *Histograms of the datasets.*

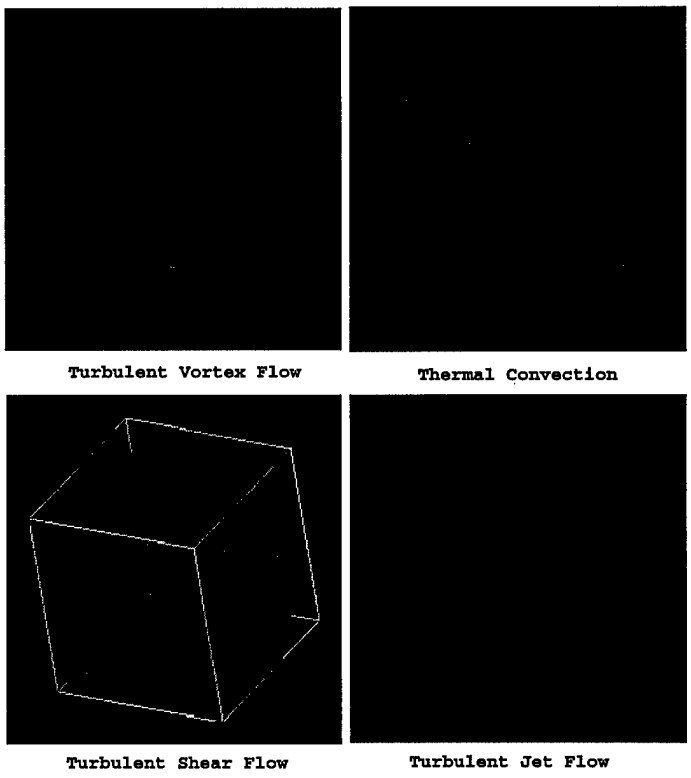


FIG. 6. *One selected frame for each dataset.*

TABLE 2
Compression rates derived from different quantizations.

Dataset	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

SUN Ultra Sparc. For a dataset containing 100 time steps, it takes about 5-10 minutes to encode the whole dataset.

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 heart and turbulent jet flow datasets achieved the highest compression rate and the highest increase in rendering rate. For the turbulent jet flow dataset, 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 dataset, we saw a 93% increase in the overall rendering rate.

Figure 7 shows three renderings of the turbulent jet flow dataset. The baseline renderer renders the full dataset from the volume data at each time step. The tvvd-renderer uses all of the optimizations discussed in Section 4. 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 dataset 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 datasets 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 dataset. Figure 8 shows the rendering times for this dataset

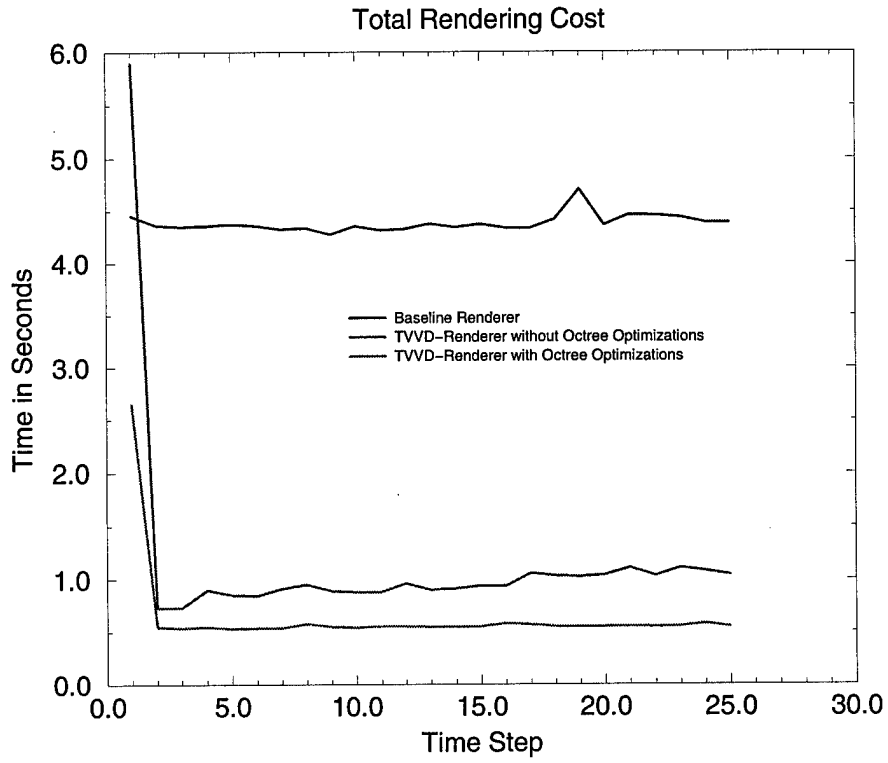


FIG. 7. *Rendering cost for turbulent jets dataset. The time is the total time to process, including reading encoded data from disk, unencoding when necessary, calculating gradient, rendering and compositing.*

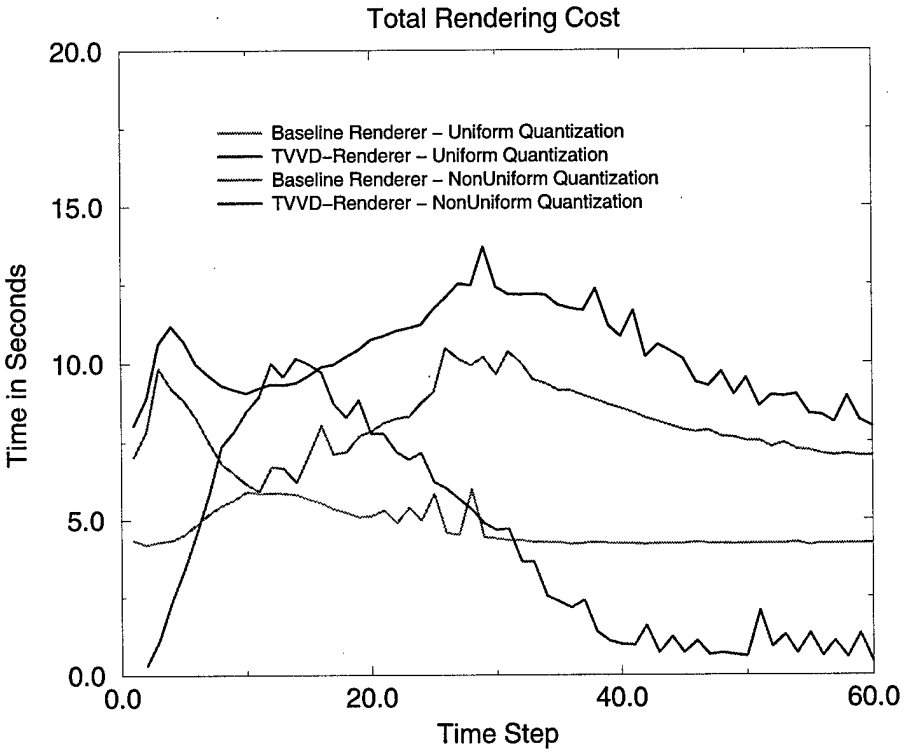


FIG. 8. *Rendering results for the turbulent shear flow dataset.*

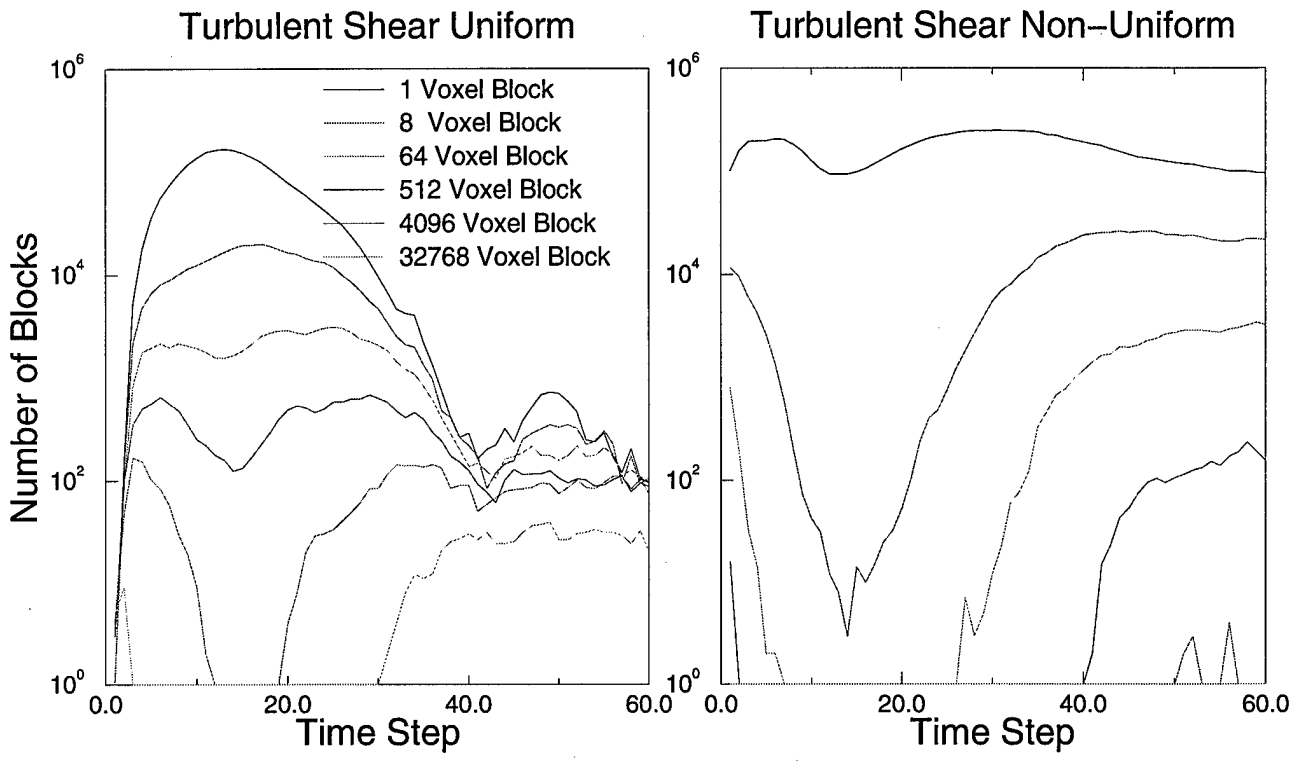


FIG. 9. *Number of blocks in turbulent shear flow dataset.*

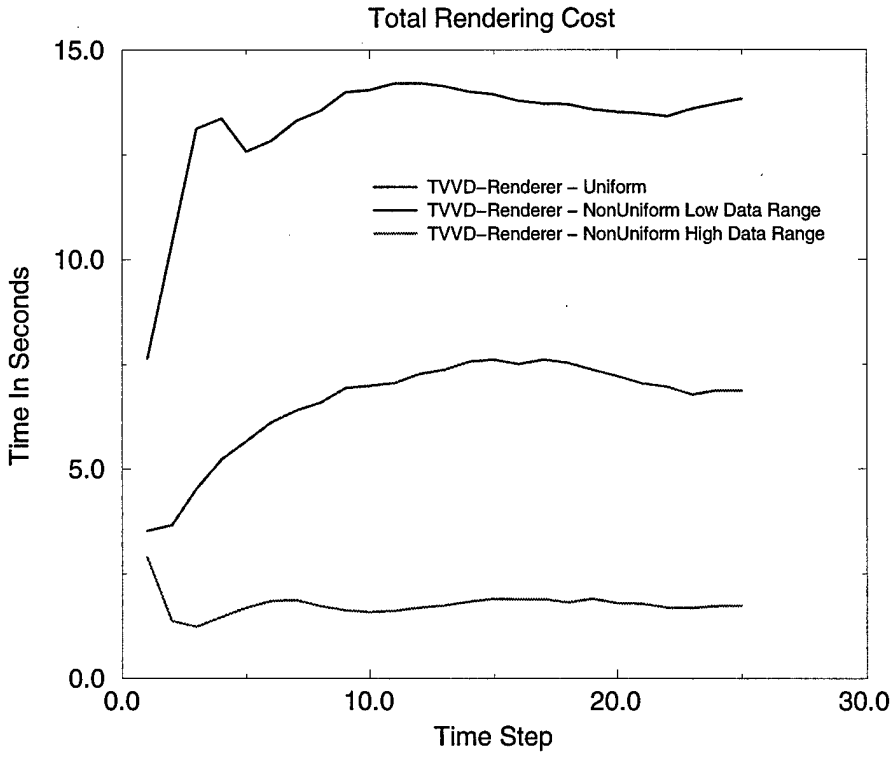


FIG. 10. *Rendering results Thermal Convection dataset.*

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 large number of small matching blocks, not a small number of large matching blocks. The renderer is not able to take advantage of the compression and the rendering rate is consistently higher. Generally, if the data are compressed by less than 50%, unless many large subtrees were merged, little rendering performance gain can be obtained. This is consistent with the results reported in [16].

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 dataset 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 dataset 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 dataset 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. 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 datasets 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 datasets 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] T.-C. CHIUEH AND K.-L. MA, *A Parallel Pipelined Renderer for Time-Varying Volume Data*, December 1997. ICASE Report No. 97-70.

[2] T. Z. CHIUEH, C. K. YANG, T. HE, H. PFISTER, AND A. KAUFMAN, *Integrated Volume Compression and Visualization*, in Proceedings of the Visualization '97 Conference, October 1997, pp. 329–336.

[3] J. E. FOWLER AND R. YAGEL, *Lossless Compression of Volume Data*, in Proceedings of the 1994 Symposium on Volume Visualization, October 1994.

[4] R. HAIMES, *Unsteady Visualization of Grand Challenge Size CFD Problems: Traditional Post-Processing vs. Co-Processing*, in Proceedings of the ICASE/LaRC Symposium on Visualizing Time-Varying Data, 1996, pp. 63–75. NASA Conference Publication 3321.

[5] V. S. JASWAL, *CAVEvis: Distributed Real-Time Visualization of Time-Varying Scalar and Vector Fields Using the CAVE Virtual Reality Theater*, in Proceedings of the Visualization '97 Conference, October 1997, pp. 301–308.

[6] D. LANE, *UFAT - A Particle Tracer for Time-Dependent Flow Fields*, in Proceedings of the Visualization '94 Conference, 1994, pp. 257–264.

[7] D. LAUR AND P. HANRAHAN, *Hierarchical Splatting: A Processive Refinement Algorithm for Volume Rendering*, in Proceedings of SIGGRAPH '91, 1991.

[8] M. LEVOY, *Efficient Ray Tracing of Volume Data*, ACM Transactions on Graphics, 9 (1990).

[9] K.-L. MA, *Runtime Volume Visualization for Parallel CFD*, in Proceedings of Parallel CFD '95 Conference, 1995. California Institute of Technology, Pasadena, CA, June 25-28.

[10] K.-L. MA, M. COHEN, AND J. PAINTER, *Volume Seeds: A Volume Exploration Technique*, The Journal of Visualization and Computer Animation, 2 (1991), pp. 135–140.

[11] N. MAX AND B. BECKER, *Flow Visualization using Moving Textures*, in Proceedings of the ICASE/LaRC Symposium on Visualizing Time-Varying Data, 1996, pp. 77–88. NASA Conference Publication 3321.

[12] S. G. PARKER AND C. R. JOHNSON, *SCIRun: A Scientific Programming Environment for Computational Steering*, in On-line Proceedings of the 1995 Supercomputing Conference, 1995. http://scxy.tc.cornell.edu/sc95/proceedings/.

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

[14] J. ROWLAN, E. LENT, N. GOKHALE, AND S. BRADSHAW, *A Distributed, Parallel, Interactive Volume Rendering Package*, in Proceedings of the Visualization '94 Conference, 1994, pp. 21–30.

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

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

[17] D. SILVER AND X. WANG, *Volume Tracking*, in Proceedings of the Visualization '96 Conference, 1996, pp. 157–164.

[18] R. WESTERMANN, *A Multiresolution Framework for Volume Rendering*, in Proceedings of the 1994 Symposium on Volume Visualization, October 1994.

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

[20] ——, *Multi-Dimensional Trees for Controlled Volume Rendering and Compression*, in Proceedings of the 1994 Symposium on Volume Visualization, October 1994.