

Time-Varying, Multivariate Volume Data Reduction

Nathaniel Fout Kwan-Liu Ma
 Department of Computer Science
 University of California, Davis

James Ahrens
 Advanced Computing Laboratory
 Los Alamos National Laboratory

ABSTRACT

Large-scale supercomputing is revolutionizing the way science is conducted. A growing challenge, however, is understanding the massive quantities of data produced by large-scale simulations. The data, typically time-varying, multivariate, and volumetric, can occupy from hundreds of gigabytes to several terabytes of storage space. Transferring and processing volume data of such sizes is prohibitively expensive and resource intensive. Although it may not be possible to entirely alleviate these problems, data compression should be considered as part of a viable solution, especially when the primary means of data analysis is volume rendering. In this paper we present our study of multivariate compression, which exploits correlations among related variables, for volume rendering. Two configurations for multidimensional compression based on vector quantization are examined. We emphasize quality reconstruction and interactive rendering, which leads us to a solution using graphics hardware to perform on-the-fly decompression during rendering.

Categories and Subject Descriptors

I.3 [Computer Graphics]: Applications; E.4 [Coding and Information Theory]: Data compaction and compression

Keywords

Compression, hardware acceleration, multivariate data, time-varying data, interactive visualization, vector quantization, volume rendering

1. INTRODUCTION

The advent of terascale supercomputers enables advanced scientific simulations that probe deeply into complex physical phenomena and chemical processes at the heart of climate modeling, combustion engineering, fusion energy sciences, nuclear astrophysics, high-energy physics, life sciences, etc. While modeling at unprecedented scale and complexity, one pressing problem for the scientists, however, is that they

do not have convenient and efficient ways to store, transfer and understand the massive amount of data produced by their simulations. These datasets are typically time-varying and multivariate, consisting of both scalar and vector fields. One complete data set can take terabytes of storage space. Consequently, scientists generally leave their data on a mass storage system at the supercomputing facility where they run their simulations. Because of the high cost of transferring terabytes of data, the subsequent data analyses are done mostly to a small subset of the dataset and limited by what the supercomputing facility can provide. Even though one can always utilize faster (and more expensive) storage and network hardware to relieve this problem, a more effective solution would be to reduce the data.

Various data reduction techniques that use either value-based encoding or physically-based feature extraction have been investigated to make possible comprehension of the essential information in large volume data. This paper presents a study of volume data reduction with lossy compression in order to facilitate evaluation by visualization. Our study is unique in two ways.

First, in terms of encoding we study multivariate volume compression as opposed to scalar volume compression. The justification for this is the fact that in most simulations the variables represent certain physical properties of the system. These properties are rarely independent, although the relationships may be very complex. In our study, we attempt to take advantage of these relationships indirectly by relying on their correlation to provide good rate-distortion behavior during compression. The compression method, being a variant of vector quantization, is block-based, and so we investigate two possible partitioning strategies based on the dimensionality of the data. The first is spatial compression, which is analogous to typical image compression methods. The second is temporal compression, in which voxels with temporal proximity are grouped together.

Second, in terms of decoding we propose on-the-fly decompression during rendering by leveraging the programmability of modern graphics cards. While this topic has been explored by other researchers, a serious drawback common to all these efforts is the inability to interactively reconstruct a continuous representation of the volume, as well as the inability to compute gradients needed for volume shading. We propose the use of a novel decompression algorithm which allows high-quality volume rendering directly from the compressed volume, including continuous reconstruction and lighting, all at interactive frame rates.

Our use of lossy compression is based on the observation

©2005 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor or affiliate of the [U.S.] Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SAC '05, March 13-17, 2005, Santa Fe, New Mexico, USA.
 Copyright 2005 ACM 1-58113-964-0/05/0003 ...\$5.00.

that high data precision is important during the simulation to maintain numerical stability, but for visualization data integrity can be slightly compromised. While using multivariate compression has the potential to offer improved performance over independent compression of variables, an important advantage of multivariate compression is that it can facilitate comparative visualization, which usually requires fast access to multiple collocated variables.

Our results show an average storage savings of over 70%, which allows us to significantly lower the storage requirements for the entire data set, transfer data of comparable content much faster through the storage hierarchy, and fit more time steps (and variables) in memory. Our findings suggest that scientists should consider such a data reduction option in their terascale (and soon petascale) data understanding infrastructure.

2. BACKGROUND

In this section we give a brief background in the area of volumetric compression. We divide previous work into two categories, either spatial or temporal, based on the domain of the compression algorithm used. Methods that consider time-varying volumes as a single 4D data volume are also considered to be temporal for our classification purposes.

2.1 Volume Data Compression

Volumetric compression methods generally fall into one of two categories. The first type is based on the premise that volumetric compression should be tightly coupled with rendering. This in turn imposes special limitations on the compression algorithm; in particular, decompression must allow fast, direct, random voxel access, as the voxel access pattern in volume rendering is view dependent. Alternatively, a loose coupling allows more freedom in selection, with a potential disadvantage being decreased interactivity. On one extreme is the direct rendering of compressed data without decompression, and on the other is complete decompression prior to rendering. Both of these options suffer from serious drawbacks, however, and the best overall performance is achieved with methods that combine efficient decompression closely with rendering.

Much progress has been made in the application of compression to regular-grid volume data. Compression in general falls into two categories, lossless and lossy. Lossless compression usually results in lower compression ratios, but offers in return perfect reconstruction of the original data. In [6] differential encoding, a type of predictive encoding, is used in conjunction with Huffman coding to losslessly encode volumes, achieving better compression (approx. 50%) than common lossless compression utilities by taking advantage of the 3D structure in volume data. Another more recent work [4] provides lossless compression of smooth time-varying simulation data, also using predictive encoding. This technique targets scientific data stored in floating-point format, and offers a lossy alternative as well which is based on truncation of the least important bits in the compressed representation.

Lossy compression has the potential to offer the greatest storage savings in situations where some amount of error is tolerable. Popular lossy compression methods include transform encoding and subband/wavelet encoding. These techniques rely on quantization to achieve compression, but quantization can also be applied directly to the original data.

In transform encoding the Discrete Cosine Transform (DCT) is most suitable for compression. Direct application of the DCT to volume data using a 3D DCT is described in [25], where blocks are decompressed on demand in order to support coupling of rendering and decompression. In [2] the volume is also blocked and a 3D DCT is applied, but rendering is performed on the compressed data set using the Fourier Projection Theorem. Wavelet compression has become extremely popular for lossy compression, and is the method of choice for the JPEG-2000 compression standard. A 3D wavelet transform in combination with motion compensation (a technique used in video encoding) is used in [8] to achieve high-rate compression of time-varying volumes. In [9] wavelet compression is used with projective classification to enable decompression and rendering together at interactive frame rates using texture hardware. Subband encoding, closely related to wavelet encoding, has also been applied to volume data as described in [7]. In [17] high-pass coefficients from the Laplacian decomposition are encoded using vector quantization, and decompression is coupled tightly with rendering by performing on-the-fly decompression in graphics hardware. Vector quantization (VQ) is applied in a straightforward way to volume data in [15, 16] by grouping voxel attributes to form vectors. Finally, in [3] fractal compression is adapted to volumetric data with results indicating better performance than VQ but not quite as good as DCT. However, fractal decompression prohibits close coupling with rendering.

2.2 Time-Varying Volume Data Compression

The problem of time-varying data visualization has received increasing attention, and various data encoding, reduction, and rendering techniques have been developed. One class of techniques treats time-varying volume data as 4D data. For example, Wilhelms and Van Gelder [22] encode time-varying data with a 4D tree (an extension of octree) and use an associated error/importance model to control compression rate and image quality. Larsen et al. [12] introduce a more refined design based on a *4th-root-of-2* subdivision scheme coupled with a linear B-spline wavelet scheme for representing time-varying volume data at multiple levels of detail. Woodring et al [23] visualize 4D data by slicing or volume rendering in the 4D space. The resulting hyperplane and hyperprojection can display some unique space-time features.

The other class of techniques separates the time dimension from the spatial dimensions. Shen and Johnson [19] introduce differential volume rendering which exploits temporal coherence of the data and compress the data in a substantial way, but it is limited to a one-way, sequential browsing of the temporal aspect of the data. Ma et al. [14] integrate non-uniform quantization with octree and difference encoding and speed up rendering by sharing subtrees among consecutive time steps. Shen et al. [18] refine the design deriving a hierarchical data structure called the Time-Space Partitioning (TSP) tree, which captures both the spacial and temporal coherence from a time-varying field. It uses an octree for partitioning the volume spatially and a binary tree for storing temporal information.

Several other techniques also worth mentioning. Westermann [21] encodes each time step separately using wavelet transforms. The result is a compressed multiscale tree structure also providing an underlying analysis model for char-

acterizing the data. By examining the multiscale tree structures and wavelet coefficients, it is possible to perform feature extraction, tracking, and further compression more efficiently. Anagnostou et al. [1] exploit temporal coherence to render only the changed parts of each slice and use run-length encoding to compress the spatial domain of the data. Lum et al. [13] use temporal encoding of indexed volume data that can be quickly decoded in graphics hardware. Sohn et al. [20] compress time-varying isosurfaces and associated volumetric features with wavelet transforms to allow fast reconstruction and rendering.

3. MULTI-DIMENSIONAL ENCODING

The first step to choosing an appropriate compression scheme is to identify the problem domain. In our case we would like to achieve interactive visualization of multivariate time-varying data. Although a small loss of data is acceptable, a reasonable amount of fidelity is essential if scientific observations are to be made based on the rendering. On the other hand, high rates are required to handle such large data sets. If unconstrained we would probably use a wavelet compression scheme, but unfortunately it is difficult to achieve interactive rendering from wavelet-encoded data. With all of these requirements in mind we chose vector quantization as the tool of choice.

Vector quantization (VQ) extends scalar quantization by grouping the data into blocks or vectors to be compressed together. A relatively small set of representative vectors is then constructed from the data and placed in a codebook. Each vector from the data is represented by a codebook vector, as determined by searching the codebook for the closest vector. Vectors in the codebook are identified by a unique index, so that the compressed data will consist of a series of indices and a codebook containing the mapping from index to reconstruction vector. There are numerous variants of VQ, but in this work we use only one of these, called Mean-Removed Shape-Gain VQ (MRSG-VQ). The idea is to reduce the size of the codebook needed by removing aspects of the data which impede the quantization (such as the dynamic range) and to store them separately. With MRSG-VQ we remove the mean and magnitude and encode these separately using scalar quantization. The remaining vector, having zero mean and unit magnitude, represents the shape or trend of the vector. Encoding these shape vectors allows the VQ to better capture the structure present in the data.

In this section we describe how VQ can be applied to achieve data reduction in large multivariate time-varying data sets. One key advantage of VQ which makes it ideal for our situation is the speed of decompression; essentially a single table look-up is all that is required. We explore two encoding possibilities, describing both the algorithm for compression and the decompression process as mapped to programmable graphics hardware.

3.1 Compression Strategies

An important decision in VQ is how the data will be blocked into vectors. Several options exist, based on identification of the different dimensions within which the data is structured. As our data is multivariate, time-varying and in 3D space, we have three obvious choices. First, we could simply group all the variables in a single voxel together as one vector. This would be a direct way to perform multidimensional encoding, but the problem is that for many of

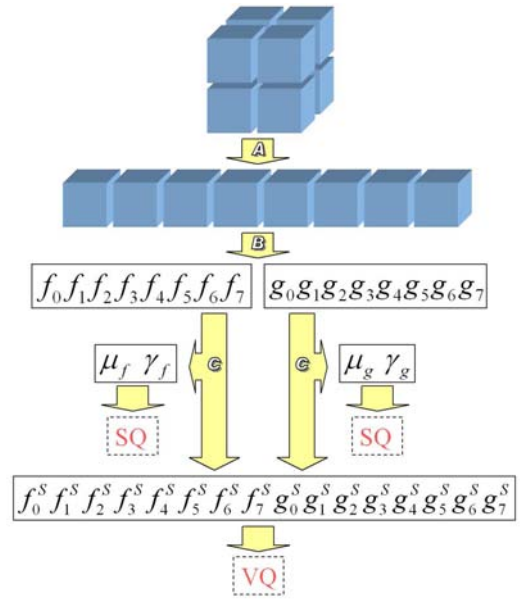


Figure 1: Multidimensional encoding of 2 variables f and g . Step A (for spatial blocking only) takes a $2 \times 2 \times 2$ block of voxels and rearranges them to be linear. In the case of temporal blocking we already have a linear arrangement of voxels. Step B forms separate vectors for each variable. Step C removes the mean and gain to be scalar quantized, and concatenates the shape vectors to be vector quantized.

the datasets there exists only a few variables. Encoding such small vectors does not provide high rates. Second, we could group the time series of a single voxel, achieving temporal compression. Of course the level of temporal coherency in the data set would be important for the success of this option. Finally, as in typical image compression algorithms we could group spatially, dividing the volume into small blocks to be compressed. In our study we explore the latter two options.

The first configuration we consider is temporal blocking, as shown in Figure 1. We first take the time series of each variable in a given voxel separately. We then process this series by removing the mean and the gain to produce a shape vector. The mean and gain for each variable are compressed using scalar quantization, but the shape vectors for all variables are concatenated to form one single vector. This composite vector is compressed using VQ. Therefore, if we have v variables and are encoding n time steps, the resulting compressed data will consist of a single volume in which each voxel will have $2v$ scalar values and 1 index into a codebook with size vn vectors. The compressed volume itself will have the same dimensions as the original volumes but will store n time steps instead of 1. While the number of time steps n is a user-defined parameter, in order to maintain quality reconstruction n should be chosen conservatively. The temporal coherency will largely determine the maximum possible value for a given desired quality.

For the second configuration, also shown in Figure 1, we consider spatial blocking by grouping voxels in a $2 \times 2 \times 2$ block together. We first group the 8 values for each variable together. Then we process each block in the same way as

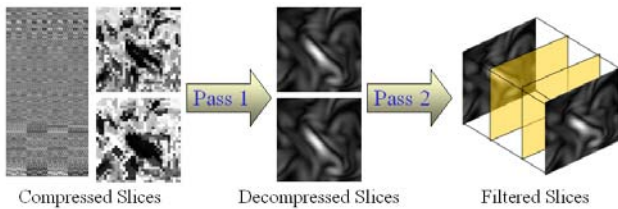


Figure 2: Two-pass algorithm for on-the-fly decompression. In the first pass two original slices are decompressed, and in the second pass sampling slices are rendered from the two decompressed slices.

in temporal compression by removing the mean and gain. The mean and gain are again quantized and stored separately, and the shape vectors are concatenated to form a single vector to encode using VQ. In this configuration, for each volume in the original data set of size $x \times y \times z$ we will have a corresponding volume in the compressed data of size $x/2 \times y/2 \times z/2$. Each voxel in the compressed volume will have $2v$ scalar values and 1 index into a codebook.

3.2 Decompression and Rendering

Ideally we would like to delay decompression until the data is actually needed in rendering. By exploiting programmable graphics hardware, decompression and rendering can be accomplished simultaneously to provide interactive visualization directly from the compressed volume data. Of the various compression techniques available, VQ is uniquely suited for this task due to its extremely fast decompression. When rendering directly from the compressed volume, decompression must precede sampling in the fragment program. This brings to light one of the key limitations of existing methods using on-the-fly decompression: interpolation of the index volume is incorrect and produces severe artifacts. The correct order is to decode each voxel prior to interpolation. This means that trilinear interpolation must be performed explicitly in the fragment program following eight decompressions. Because of the impact on frame rate, all methods to date forego a continuous reconstruction and use nearest-neighbor filtering. Computing gradients for lighting is thus out of the question, as it requires several more texture reads and 6 additional trilinear interpolations, not including the actual lighting instructions.

As our goal is quality rendering from the compressed data set, including continuous reconstruction and lighting, we use a new variant of on-the-fly decompression during rendering which is similar to the method used in [10] to render from 2D slices using a single stack of textures. The motivation for this algorithm is that with the method described above there are two major inefficiencies: first, a single voxel may be decompressed many times, and second, manual interpolation in the fragment program is slower than the built-in filtering supplied by the graphics card. Both of these problems can be addressed by decompressing entire slices at a time and sampling from these slices as shown in Figure 2. We begin by allocating an offscreen buffer, which will have both a front and back surface for rendering. The back buffer is used to decompress slices of the original volume, and the front buffer is used for volume rendering. The basic algorithm begins by first finding the axis with which the view is most closely

aligned. Slices are then rendered aligned with that axis.

This method achieves exactly what we want: no matter how many times it is needed, a particular voxel will be decompressed exactly once. Furthermore, when rendering the sampling slices we can read from the decoded slices using bilinear filtering supported by the native hardware. The only work we have to do in the fragment program is to take two samples from the adjacent slices and perform the final linear interpolation by hand to get a trilinearly filtered sample. So far we have only discussed how to get continuous reconstruction. If we want to perform lighting then we need to also compute gradients on-the-fly. In order to do this we simply keep a 4-slice buffer instead of 2-slice buffer. This enables us to calculate the partial derivatives using central differences in the fragment program when we are rendering sampling slices. The drawbacks of this technique are the overhead for rendering in two passes, and the restriction of using axis-aligned sampling planes for volume rendering. Nevertheless, the savings in computation is dramatic, thereby allowing us to achieve our goal of high-quality interactive rendering from the compressed data set.

As for the actual decompression (first pass of the algorithm), the mapping to graphics hardware is straightforward. The index volumes containing means, gains, and codebook indices are stored as 3D textures and accessed using nearest-neighbor filtering. The codebooks are stored as 2D textures with the s coordinate indexing the codes and the vector offset given by the r coordinate. If we have four or less variables then the shape vectors can be separated and stored in the RGBA channels of the texture. In this way we can retrieve the value of all variables with only one texture read. Although in general codebook accesses should use nearest-neighbor filtering, in the case of temporal compression we can use linear interpolation between consecutive vector elements, resulting in smooth interpolation in the time domain; that is, we can animate smoothly through time instead of discretely from time step to time step. However, this requires us to duplicate vectors due to the fact that the hardware allows only bilinear interpolation of 2D textures.

In order to decompress a single voxel and variable, we must first access the index volume to find the mean, gain, and codebook index. The codebook index gives us the s coordinate of the codebook texture. The value of the r coordinate depends on the type of compression (spatial vs. temporal) and the variable required. For the case of temporal compression, simply knowing the current time and the variable is enough to compute r . For spatial compression we need a mapping from the voxel position within the $2 \times 2 \times 2$ block to a 1D vector offset. We use the same approach as in [17] and use an address texture to quickly find this mapping. The address texture is a 3D texture of size $2 \times 2 \times 2$ which is repeated to cover the entire volume. Each texel in this texture holds the necessary offset which, when combined with the variable, gives a unique r coordinate. Finally the codebook is accessed and the gain and mean are added back in to produce the reconstructed value. In summary, the decompression process consists of a single index volume read, followed by a single codebook texture read. For spatial compression we use an additional address texture read.

4. RESULTS AND DISCUSSION

Our implementation of multidimensional encoding first

encodes the volumes in software as pre-processing. For code-book training we use Pairwise Nearest Neighbor [5] to initialize and then perform a relaxation with a few iterations of LBG [11]. Although this training process is slow there are alternative algorithms available which sacrifice quality of encoding for speed. As previously stated the spatial compression uses $2 \times 2 \times 2$ blocks of data, so for purposes of comparison we chose to encode 8 time steps at once for the temporal compression. The decompression and rendering are performed in graphics hardware using hand-tuned assembly fragment programs.

To test our implementation we chose three time-varying multivariate datasets. The first two are compressible fluid flow simulations of jet streams mixing and were run for 2000 time steps. The first consists of five jets, is sampled on a $128 \times 128 \times 128$ grid, and stores every time step. The second consists of four jets, is sampled on a $256 \times 256 \times 256$ grid, and stores every 10th time step. Each volume has 3 variables: energy, density, and velocity magnitude. The third data set is from a supernova simulation with 200 time steps and three variables: pressure, density, and velocity magnitude. The grid size is $256 \times 256 \times 256$.

The results of the compression tests are shown in Tables 1-4. All tests were performed using a single PC with a P4 3.2 GHz and 2 GB of RAM. Encoding times for both spatial and temporal compression ranged from several minutes to tens of minutes. In Figures 3-5 selected results are displayed showing the three variables for the five-jet data set and the supernova data set at a range of compression rates. These images were generated by our on-the-fly decompression implementation, which achieves 17 fps when rendering to a 512×512 window.

We see from these tests that temporal encoding results in higher quality than spatial encoding for the jet data sets, but the opposite is true for the supernova data set. Thus it is specific to the simulation itself as to whether it would be better to encode in space or in time. It would be nice to be able to predict which is better a priori; however, to our knowledge no investigation into this topic has been made. For the five-jet data set we also stored every time step in order to study the effect of temporal coherence in the temporal compression. As can be seen the error is dramatically reduced at the same rate when every time step is present. However, the error for using every 10th time step is already low enough that this further reduction does not noticeably improve the quality in terms of volume rendering; thus, there is no real advantage in using every time step for purposes of temporal compression of 8 time steps.

As expected, the lower-rate multivariate compression results in larger errors, but for most cases the error increase is not noticeable. For these cases multivariate compression is a viable option for achieving greater compression while making available all variables for other tasks, such as comparative/multivariate visualization. While the errors listed in Tables 1,2 and 4 may seem very small, we have intentionally set the rate so that high fidelity reconstruction is achieved. The reason for this becomes clear by examining the artifacts present in Figure 3d. While for image compression the error observed for a given pixel is just the error associated with that pixel in the reconstructed image, in volume rendering the error for a given pixel is the product of many errors accumulated along the ray. In addition, high frequency features such as specular highlights and isosurfaces (which are

Table 1: Compression results for five-jet data set ($128 \times 128 \times 128$ grid, 3 variables, 2000 time steps).

Type		Space	Time
Multivariate	Com. Rate (%)	70.05	70.05
	PSNR-e (dB)	47.17	55.73
	PSNR-r (dB)	52.79	60.96
	PSNR-v (dB)	42.11	57.25
Univariate	Com. Rate (%)	61.72	61.72
	PSNR-e (dB)	51.72	66.92
	PSNR-r (dB)	56.96	65.94
	PSNR-v (dB)	49.40	67.70

Table 2: Compression results for four-Jet data set ($256 \times 256 \times 256$ grid, 3 variables, 2000 time steps).

Type		Space	Time
Multivariate	Com. Rate (%)	70.05	70.05
	PSNR-e (dB)	56.24	57.16
	PSNR-r (dB)	56.97	65.72
	PSNR-v (dB)	49.17	58.28
Univariate	Com. Rate (%)	61.72	61.72
	PSNR-e (dB)	60.33	60.65
	PSNR-r (dB)	60.38	70.71
	PSNR-v (dB)	52.60	61.81

Table 3: Compression results for supernova data set ($256 \times 256 \times 256$ grid, 3 variables, 200 time steps).

Type		Space	Time
Multivariate	Com. Rate (%)	70.05	70.05
	PSNR-d (dB)	34.58	32.31
	PSNR-p (dB)	39.58	37.71
	PSNR-v (dB)	41.70	36.45
Univariate	Com. Rate (%)	61.72	61.72
	PSNR-d (dB)	36.34	32.62
	PSNR-p (dB)	39.89	38.00
	PSNR-v (dB)	45.68	38.27

Table 4: Temporal compression of five-jet data set encoding every time step vs. every tenth time step.

Type		Time-1	Time-10
Multivariate	Com. Rate (%)	70.05	70.05
	PSNR-e (dB)	65.20	55.73
	PSNR-r (dB)	69.01	60.96
	PSNR-v (dB)	69.05	57.25
Univariate	Com. Rate (%)	61.72	61.72
	PSNR-e (dB)	70.92	66.92
	PSNR-r (dB)	71.13	65.94
	PSNR-v (dB)	78.37	67.70

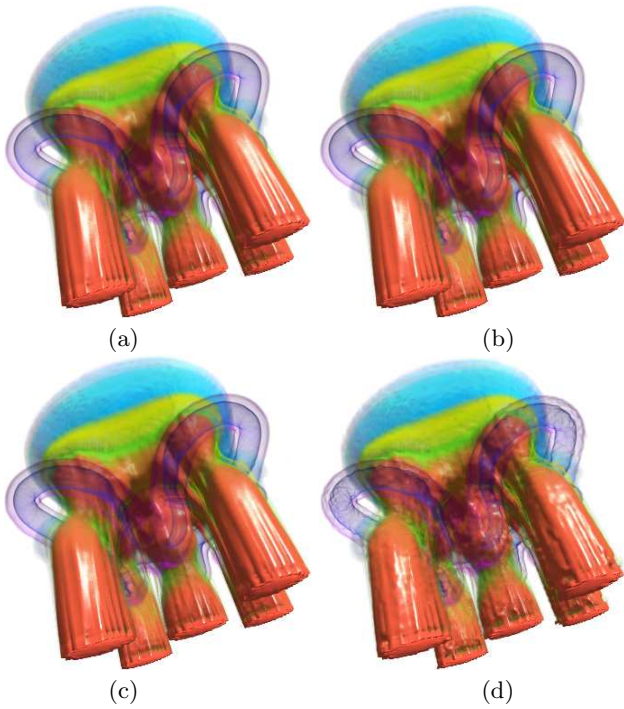


Figure 3: Rendering of the energy field (variable e) of the five-jet data set: (a) original (b) compressed, PSNR=70.92 (c) compressed, PSNR=55.73 (d) compressed, PSNR=47.17

produced from transfer functions containing high frequencies) will expose even the slightest errors. All these factors make it necessary to use high quality compression; in fact, we see in the figures that in most cases a minimum of 50 dB is required to avoid artifacts, but this is dependent of the data set. For the supernova data set even our high-rate encoding results in noticeable artifacts; this fact highlights the need for mapping better, more sophisticated compression techniques to the graphics hardware.

Our compression methods are able to reduce the five-jet data set from 12 GB to 3.5 GB, and the four-jet data set from 94 GB to 28 GB. The supernova data set which takes up 9 GB is reduced to 2.8 GB. While in terms of rate-distortion our method cannot compare to other techniques such as wavelet encoding, we are able to take advantage of the coherency among variables and our method is simple enough to allow on-the-fly decompression during rendering. This makes our method more appropriate for the task of volume rendering, whereas other methods are primarily for archiving. An important direction for future work is the development of better on-the-fly compression schemes, as current VQ-based methods are sometimes unable to achieve high quality reconstruction.

5. CONCLUSIONS

In this paper we present a solution which addresses the need for data reduction in large supercomputing environments where data resulting from simulations occupies tremendous amounts of storage. Our solution employs a lossy encoding scheme to achieve data reduction with several options in terms of rate-distortion behavior. We focus on encoding

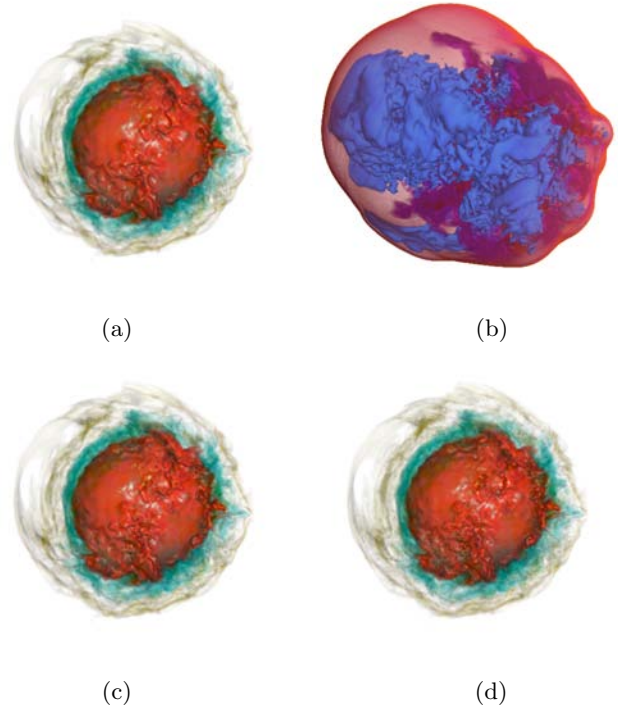


Figure 4: Rendering of the supernova data set: (a) original density (b) original velocity magnitude (c) compressed density, PSNR=36.34 (d) compressed density, PSNR=32.31

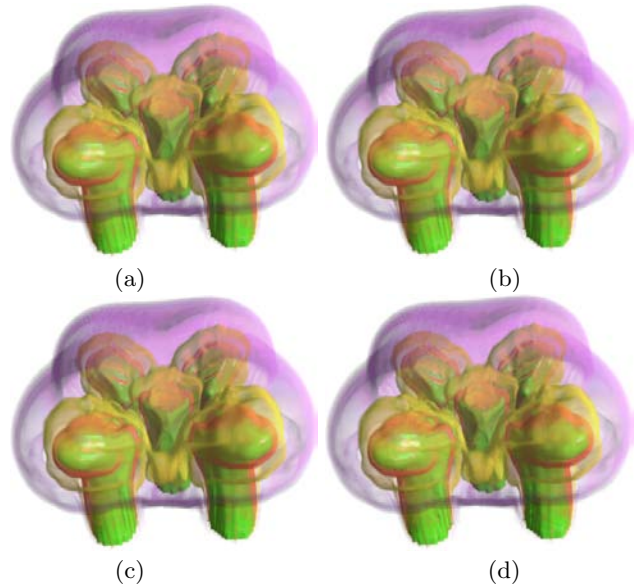


Figure 5: Rendering of the velocity magnitude field (variable v) of the five-jet data set: (a) original (b) compressed, PSNR=78.37 (c) compressed, PSNR=67.66 (d) compressed, PSNR=49.40

of multiple variables together, with optional compression in space and time. The compressed volumes can be rendered directly with commodity graphics cards at interactive frame rates and rendering quality similar to that of static volume renderers. Compression results using a multivariate time-varying data set indicate that encoding multiple variables results in acceptable performance in the case of spatial and temporal encoding as compared to independent compression of variables. The relative performance of spatial vs. temporal compression is data dependent, although temporal compression has the advantage of offering smooth animations, while spatial compression can handle volumes of larger dimensions.

Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE award), ACI 0325934 (ITR), ACI 0222991, and CMS-9980063; and Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC) and No. B523578 (ASCI VIEWS). The authors would like to thank Jens Schneider for his helpful comments.

6. REFERENCES

- [1] ANAGNOSTOU, K., ATHERTON, T., AND WATERFALL, A. 4D volume rendering with the shear warp factorization. In *Proceedings of Volume Visualization and Graphics Symposium 2000* (2000), pp. 129–137.
- [2] CHIUH, T.-C., YANG, C.-K., HE, T., PFISTER, H., AND KAUFMAN, A. Integrated volume compression and visualization. In *Proceedings of IEEE Visualization '97 Conference* (1997).
- [3] COCHRAN, W. O., HART, J. C., AND FLYNN, P. J. Fractal volume compression. *IEEE Transactions on Visualization and Computer Graphics* 2, 4 (1996), 313–322.
- [4] ENGELSON, V., FRITZSON, D., AND FRITZSON, P. Lossless compression of high-volume numerical data from simulations. In *Data Compression Conference* (2000).
- [5] EQUITZ, W. H. A new vector quantization clustering algorithm. *IEEE Trans. Acoust., Speech, Signal Processing* 37 (1989), 1568–1575.
- [6] FOWLER, J., AND YAGEL, R. Lossless compression of volume data. In *Proceedings of 1994 Symposium on Volume Visualization* (1994).
- [7] GHAVAMNIA, M. H., AND YANG, X. D. Direct rendering of laplacian pyramid compressed volume data. In *Proceedings of Visualization '95 Conference* (1995), pp. 192–199.
- [8] GUTHE, S., AND STRABER, W. Real-time decompression and visualization of animated volume data. In *Proceedings of the IEEE Visualization 2001 Conference* (2001), pp. 349–356.
- [9] GUTHE, S., WAND, M., GONSER, J., AND STRABER, W. Interactive rendering of large volume data sets. In *Proceedings of the IEEE Visualization 2002 Conference* (2002), pp. 53–60.
- [10] LEFOHN, A. E., KNISS, J. M., HANSEN, C. D., AND WHITAKER, R. T. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of IEEE Visualization 2003 Conference* (2003), pp. 75–82.
- [11] LINDE, Y., BUZO, A., AND GRAY, R. M. An algorithm for vector quantizer design. *IEEE Transactions on Communication* (1980), 84–95.
- [12] LINSEN, L., PASCUCCI, V., DUCHAINEAU, M., HAMANN, B., AND JOY, K. Hierarchical representation of time-varying volume data with '4th-root-of-2' subdivision and quadrilinear B-spline wavelets. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications - Pacific Graphics 2002* (2002), IEEE Computer Society Press.
- [13] LUM, E., MA, K.-L., AND CLYNE, J. Texture hardware assisted rendering of time-varying volume data. In *Proceedings of IEEE Visualization 2001 Conference* (October 2001).
- [14] MA, K.-L., SMITH, D., SHIH, M.-Y., AND SHEN, H.-W. Efficient encoding and rendering of time-varying volume data. Tech. Rep. ICASE Reprint No. 98-22, Institute for Computer Applications in Science and Engineering, June 1998.
- [15] NING, P., AND HESSELINK, L. Vector quantization for volume rendering. In *Proceedings of 1992 Workshop on Volume Visualization* (1992), pp. 67–74.
- [16] NING, P., AND HESSELINK, L. Fast volume rendering of compressed data. In *Proceedings of Visualization '93 Conference* (1993), pp. 11–18.
- [17] SCHNEIDER, J., AND WESTERMANN, R. Compression domain volume rendering. In *Proceedings of the Visualization 2003 Conference* (2003), pp. 293–300.
- [18] SHEN, H.-W., CHIANG, L., AND MA, K.-L. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the IEEE Visualization 1999 Conference* (1999), pp. 371–378.
- [19] SHEN, H.-W., AND JOHNSON, C. R. Difference volume rendering: A fast volume visualization technique for flow animation. In *Proceedings of IEEE Visualization 1994 Conference* (1994).
- [20] SOHN, B.-S., BAJAJ, C., AND SIDDAVANAHALLI, V. Feature based volumetric video compression for interactive playback. In *Proceedings of Volume Visualization and Graphics Symposium 2002* (2002), pp. 89–96.
- [21] WESTERMANN, R. Compression time rendering of time-resolved volume data. In *Proceedings of the Visualization '95 Conference* (1995), pp. 168–174.
- [22] 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).
- [23] WOODRING, J., WANG, C., AND SHEN, H.-W. High dimensional direct rendering of time-varying volumetric data. In *Proceedings of Visualization 2003 Conference* (October 2003), pp. 417–424.
- [24] YANG, C.-K. Integration of volume visualization and compression: A survey, September 2000. Research Proficiency Exam Report.
- [25] YEO, B.-L., AND LIU, B. Volume rendering of dct-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics* 1, 1 (1995), 29–43.