Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

End-to-End Data Reduction and Hardware Accelerated Rendering Techniques for Visualizing Time-Varying Non-uniform Grid Volume Data

Hiroshi Akiba[†], Kwan-Liu Ma[†] and John Clyne[‡]

[†]Institute for Data Analysis and Visualization, University of California at Davis, U.S.A
[‡]Scientific Computing Devision, National Center for Atmospheric Research, Colorado, U.S.A

Abstract

We present a systematic approach for direct volume rendering terascale-sized data that are time-varying, and possibly non-uniformly sampled, using only a single commodity graphics PC. Our method employs a data reduction scheme that combines lossless, wavelet-based progressive data access with a user-directed, hardware-accelerated data packing technique. Data packing is achieved by discarding data blocks with values outside the data interval of interest and encoding the remaining data in a structure that can be efficiently decoded in the GPU. The compressed data can be transferred between disk, main memory, and video memory more efficiently, leading to more effective data exploration in both spatial and temporal domains. Furthermore, our texture-map based volume rendering system is capable of correctly displaying data that are sampled on a stretched, Cartesian grid. To study the effectiveness of our technique we used data sets generated from a large solar convection simulation, computed on a non-uniform, 504×504×2048 grid.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications I.3.1 [Computer Graphics]: Graphics processors I.3.3 [Computer Graphics]: Viewing algorithms E.4 [Coding and Information Theory]: Data compaction and compression

1. Introduction

Studying time-evolving phenomena is critical for solving many scientific and engineering problems. The ability to interactively visualize and explore complex dynamic features contained within time-varying data is absolutely essential to ensure their correct interpretation and analysis, to provide insights, and to communicate those insights with others. However, a time-varying data set from a computational fluid dynamics simulation, for example, can contain an enormous amount of information in the spatial, temporal and variable domains; a single time step may contain hundreds of millions of grid points, while the temporal data set may occupy terabytes of storage in aggregate. Visualizing static volumes of this scale is challenging enough without the added difficulty imposed by the temporal dimension. The principal impediment to visualizing time-vary data arises from the need to manage and transfer time-steps between storage hierarchies from the potentially capacious, but slow, rotating disk arrays to the small, but high-performing, video memories.

This paper presents a systematic approach to direct volume rendering temporal data based on a combination of two user-directed data reduction strategies. First, a multiresolution data representation scheme, developed in our previous work, is employed to enable progressive access to the numerical simulation's raw floating point outputs [Cly03]. A low-overhead *packing* scheme is then engaged to provide a second level of data reduction. The combination of these two user-directed data reduction techniques allow the researcher to effectively make speed/quality tradeoffs, enabling the interactive visual exploration of terascale sized data using only a lowly desktop PC.

The reduced space requirements can be exploited to fit more time steps in computer memory, enabling interac-

† {akibalma}@cs.ucdavis.edu
‡ clyne@ncar.ucar.edu

tive animation of the data's temporal domain, for example, or reduce the bandwidth requirements for data transferred between memory hierarchies, further aiding interactive performance. Our multiresolution scheme reduces demands for disk storage, the CPU's memory, and the interconnect between these storage systems, while our texture packing scheme, which supports decoding (unpacking) on the GPU, further reduces demands on main memory, video memory, and the graphics bus that connects these devices.

In addition to our data reduction strategies, we also present a method to volume render data sampled on a non-uniformly spaced, *stretched* Cartesian grid. In an effort to reduce simulation time and storage requirements some numerical fluid flow models employ stretched grids, allowing higher sampling densities to be focused where they are needed. We have devised an efficient method for correctly volume rendering stretched grids by again exploiting the flexibility of programmable graphics hardware.

Thus the main contributions of our work are two fold. First, we demonstrate a comprehensive volume rendering system that incorporates end-to-end data reduction strategies to curtail the demands on numerous system storage resources, permitting the exploration of vast time-varying data sets using only a PC equipped with a current generation graphics card. Secondly, we present an efficient hardware-accelerated method for rendering data sets computed on stretched Cartesian grids.

2. Related Work

The problem of time-varying data visualization has received increasing attention. 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 [WV94] 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. Linsen et al. [LPD*02] 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 [WWS03] visualize 4D data by slicing or volume rendering in the 4D space. The resulting hyperplane and hyperprojection can display some unique space-time features.

Another class of techniques separates the time dimension from the spatial dimension. Shen and Johnson [SJ94] introduce differential volume rendering which exploits temporal coherence of the data and compresses 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. [MSSS98] integrate non-uniform quantization with octree and difference encoding and speed up rendering by sharing subtrees among consecutive time steps. Shen et al. [SCM99] refine the design deriving a hierarchical data structure called the Time-Space Partitioning (TSP) tree, which captures both the spatial 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 are also worth mentioning. Westermann [Wes95] encodes each time step separately using wavelet transforms. The result is a compressed multiscale tree structure also providing an underlying analysis model for characterizing 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. [WAA00] 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. [LMC01] use temporal encoding of indexed volume data that can be quickly decoded in graphics hardware. Sohn et al. [SBS02] compress time-varying isosurfaces and associated volumetric features with wavelet transforms to allow fast reconstruction and rendering. Schneider and Westermann [SW03] use vector quantization to compress time-varying data in the spatial domain with both decompression and rendering done in hardware.

Work closer to our own includes that of Li et al.'s [LMK03], Kraus and Ertl's [KE02], and Binotto et al.'s [BCF03]. Li et al.'s work is concerned with the packing of a single volume. They pack voxel data into texture blocks by merging similar voxels and skipping empty space through a growing algorithm. A BSP tree is then used to organize the packed texture blocks for efficient loading and rendering of the visible blocks in the correct order. Their approach results in lossless compression and faster rendering in graphics hardware. Kraus et al. introduce an adaptive representation of texture data that stores both indices to packed data blocks and scaling factors for specifying the resolutions of the data blocks. They show that this representation can be used to encode two, three, and four dimensional data. Decoding is done with programmable graphics hardware. Binotto et al. [BCF03] effectively pack time-varying data into a three-dimensional texture which adaptively stores indices to nonhomogeneous data blocks and values of homogeneous data blocks. This approach works well when the target volume data set is highly correlated both spatially and temporally, which is not the case for our target datasets.

In contrast, our packing approach, described in greater details in Section 5, is simpler. We adopt a regular partitioning of the volume, which leads to a lower per-texture-block overhead and thus better and more predictable interactivity. This simplification also allows us to perform much faster on-the-fly packing according to a user defined transfer function.

3. Driving Applications

Our design targets time-varying volume data generated by flow simulations employing high-resolution Cartesian grids

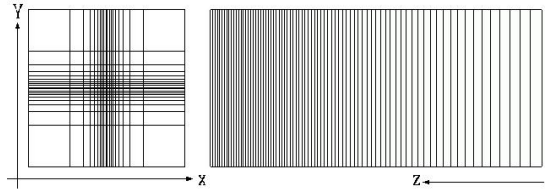

Figure 1: *A 2D view of the type of non-uniform mesh used by the simulation.*

that may possess non-uniform sample spacing. Sample coordinates along each dimension of these so called *stretched* grids are given by a monotonically increasing (or decreasing) function of the dimension's topological coordinates.

The flow features of interest in these simulations typically occupy a relatively small region of the computational domain rather than covering the entire domain. A representative example is the output of a simulation conducted by researchers at the National Center for Atmospheric Research (NCAR) to model 3D thermal starting plumes descending through a fully-compressible adiabatically-stratified fluid. Such plumes are produced by radiative cooling in the outer layers of the Sun. The simulations help scientists understand the stability of these plumes, so that they may better estimate how deeply the plumes penetrate into the solar interior after being generated in the upper surface layers. Better comprehension of the nature of these plumes in turn contributes to understanding of deep solar convection, its penetration into the stable layers of the solar interior, and the nature of the magnetic dynamo operating therein.

To capture the plumes secondary instabilities, a high-resolution $504\times504\times2048$ stretched grid (see Figure 1) is employed. The simulation required six-months of compute time on 112 processors of NCAR's IBM RS/6000 computer, and generated a total of nine terabytes of data. We obtained 400 time steps of the data for our study. Five variables including density, temperature, and the three velocity components, are stored at each grid point. There is also the need to visualize quantities derived from the model outputs such as the scalar components of the vorticity field which are computed from the velocity field, further driving up the size of this data set. Images and histograms for two of the variables at a selected time step are shown in Figure 2.

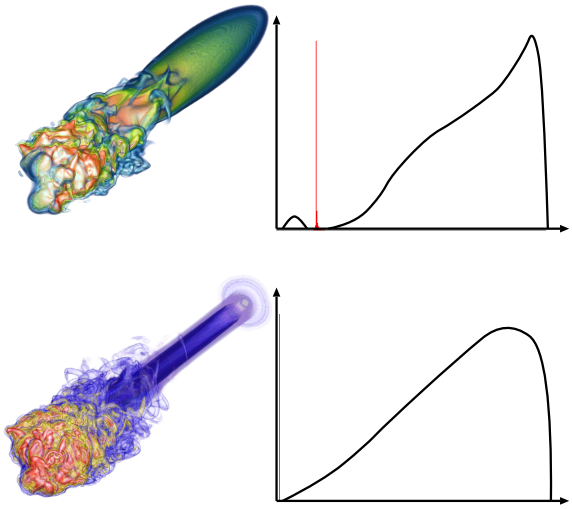

Figure 2: *Images and histograms of two variables at time step 380. Top: the z component of the velocity. Bottom: the square of the horizontal component of the vorticity. In each histogram, the curve represents the opacity transfer function used for the corresponding rendering. For both time steps, most data values lay within a very small range of values.*

4. Overview

Visualizing the plume simulation's raw floating point data directly, using a brute-force approach, would require a formidable graphics supercomputer. Maintaining a frame rate of only five Hz while animating through a time series would require a bandwidth of nearly 10.0GBs per second, for example! Even if the data are quantized to 8-bit quantities, the transfer rates required alone make desktop PC exploration prohibitive unless substantial, further data reduction is undertaken.

To make temporal data exploration possible using only a single PC with commodity graphics hardware, we employ a combination of aggressive data reduction strategies. Our three-stage, compression-based visualization process is depicted in Figure 3. The first phase of this process is the conversion of the raw floating point data into multiple resolution levels that may be accessed progressively at any desired power-of-two resolution. This is accomplished by using a lossless, wavelet-based multiresolution scheme developed by us previously [Cly03]. According to the visualization purpose and the computer used, a particular approximation level is chosen by the user.

In the second stage floating point data are quantized to integer quantities necessary to accommodate texture hardware volume rendering. Similar to the editing of an opacity transfer function, where uninteresting data values are assigned transparent opacities, the user then defines one or more intervals of interest from the domain of quantized values. Voxels possessing values within the user-defined domain of interest are packed in an efficient texture representation scheme for subsequent rendering. Voxel values falling outside the interval of interest are discarded. The resulting compact representation of the data can be transferred from disk, to main memory and to video memory more quickly then the uncompressed data. For example, the images shown in Figure 2

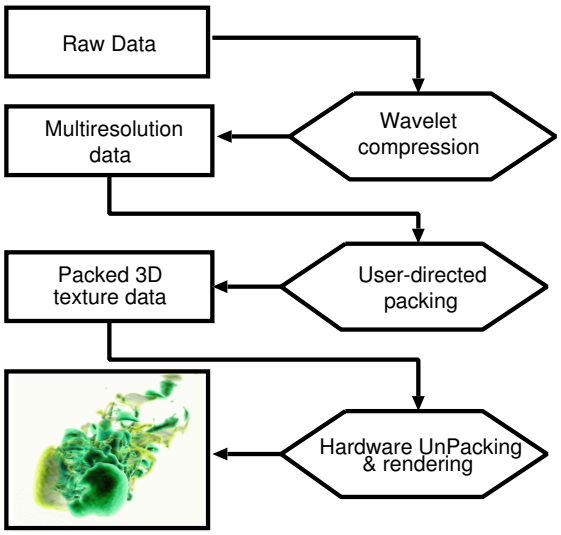

Figure 3: *A compression-based visualization process.*

correspond to an 80% reduction in data using our packing scheme.

Finally, unpacking and volume rendering of the compressed integer quantities are performed on the GPU to achieve interactive visualization. The user can freely explore in the spatial, temporal, and rendering parameter spaces of the data. Whenever desired, the user can switch to visualizing progressively finer approximations of the data, after first selecting rendering parameters (e.g., transfer functions and view) at a coarser, more interactive approximation level.

The remainder of this paper describes each of these three processes in detail, followed by test results acquired with the NCAR plume data set.

5. Data Reduction

Volume rendering has become increasingly attractive because of the real-time 3D texture support now found on commodity graphics hardware. The performance of texture hardware volume rendering, however, is constrained by fill rates, texture update rates, texture memory space, and the card's support for high-precision processing. Current card technology, for example, is capable of moving data between the main memory and texture memory at about 200-800 MB per second. The actual transfer rates may depend on the CPU speed and main memory speed. Presently, the maximum texture memory space available is 256MBs. Thus scalability of volume rendering on a single graphics card is limited. The NCAR dataset, for example, has $504 \times 504 \times 2048$ data points. The quantized version of the data would require over 500MBs to store a single variable, making volume rendering even a static volume a difficult task without careful texture management. Rendering of temporal data at interactive rates would be impossible without some decrease in the data size.

We must therefore rely on aggressive data reduction if we are to interactively visualize these data. We employ a combination of two data reduction strategies. The first is a multiresolution scheme that allows the selection of an appropriate resolution level from the raw floating point data. The second further reduces the data through a user-directed texture packing scheme. The combination of approaches allows the user to freely choose between interactivity and image quality during the data exploration process.

5.1. Multiresolution Representation

As depicted in Figure 3, the first phase of our visualization process is the conversion of the raw floating point simulation data into a hierarchical representation that permits the reconstruction of the sampled data at varying power of two resolutions. We provide a brief description of this process below. A detailed description may be found in our previous work [Cly03].

The multiresolution representation strategy we employ is based on Haar wavelet transformations that map sampled data into a space consisting of an overall coarse approximation of the original data together with the detail coefficients permitting the coarsened approximation to be refined at various scales. Thus it becomes possible to progressively access the data. That is, the data may be reconstructed at progressively finer resolutions.

The wavelet transformation process is lossless, save for floating point round off errors. Thus unlike many preprocessing strategies aimed at improving performance, multiple copies of the data are not required. The total number of wavelet coefficients are equal to that of the number of samples in the original data. Therefore no additional space is required by the wavelet representation. Other notable attributes of our data representation strategy include:

- Both the forward and inverse transform are highly efficient. This is an important consideration for very large data sets. An encoding scheme with long preprocessing requirements is not practical for terascale sized data.
- The implementation operates out-of-core for both forward and inverse transforms, permitting extremely large grids to be processed using only a modest memory footprint.
- Data approximations are produced by reconstructing and resampling the original data, not by simply subsampling.

For the NCAR data set, we have chosen to generate four approximation levels; the resolution of each level is shown in Table 1. Figure 15 shows images for the four resolution levels of the NCAR plume data set. Much as expected, the higher the data resolution, the finer the details of the flow structure that are revealed. Lastly, we note that because of the efficient computation and lossless nature of the wavelet

Table 1: *Volume data size from NCAR plume simulation for each resolution*

resolution level	dimensions	size
3	504x504x2048	2080MB
2	252x252x1024	260MB
1	126x126x512	33MB
0	63x63x256	4MB

transform, we apply the transform as a preprocess, discarding the original data and storing only the transformed data on disk.

5.2. Data Packing and Unpacking

The objective of data packing is to further reduce the size of the data obtained from our multiresolution scheme, lessening demands on the remainder of our visualization pipeline. Our basic approach is to first partition the volume uniformly along the x, y, and z directions into a set of equal-size subvolumes. The size of a subvolume should be chosen according to the data coherency and the capability of the graphics hardware. A good size to use is experimentally shown to be between 8^3 and 32^3 voxels. Even though packing small subvolumes can more effectively capture empty space by better fitting arbitrarily shaped subregions, small subvolumes can result in significant storage overhead. The overhead arises from the need to replicate voxels at the subvolume borders to avoid incorrect linear interpolation at the boundary, and also from the size of the auxiliary texture required to address valid subvolumes.

After partitioning, the subvolumes are packed into a sequence of 3D texture blocks, which we refer to as the *packed volume texture*. Subvolumes that contain values within the user-defined range of interest are packed, without loss, while those outside of the range are discarded. To address the limitations of the fragment shading language, which lacks provisions for a conditional statement, the first texture block is reserved as an empty volume to which all discarded regions refer. These empty volumes are subsequently rendered. But because they are empty they do not effect the final image. The coordinates of each subvolume are stored in a separate texture called the *address texture*. To unpack and render the volume on the GPU, the address texture must be also loaded into video memory. Unpacking is very straightforward because of the regular partitioning of the volume. Figure 4 illustrates a 2D exampling showing the relationships between a volume, its packed volume texture, and the address texture. Unpacking is performed during rendering using a fragment shading program. Unlike previous approaches [LMK03], for simplicity we reconstruct the whole volume with a one-time unpacking at the beginning of the rendering step. In this way,

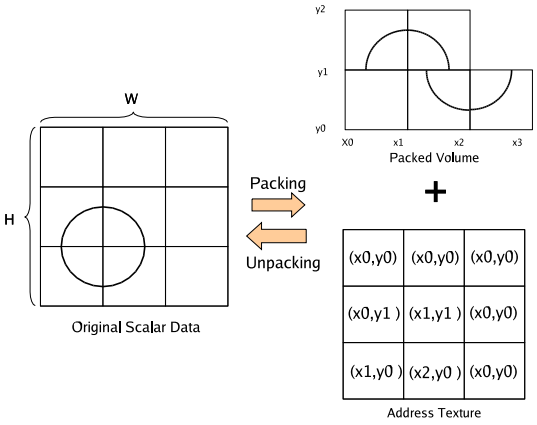

Figure 4: *Packing in the 2D case. The original data defined as a W by H image is first partitioned into sub-images. Each sub-image is stored in the packed texture starting from the lower left corner. The first sub-image is always an empty sub-image to which all the uninteresting sub-images refer. Addresses for sub-images are stored in a separate texture.*

the rendering cost is completely independent of the number of packed volume textures.

6. Stretched Grid Rendering

When 3D texture hardware support is available, sampling the volume data texture with a collection of view-aligned polygonal slices typically gives superior results over axis-aligned slicing necessitated when only 2D texture hardware is present. These slices, which are uniformly spaced, are composited using hardware alpha blending to derive the final image. When rendering data on a non-uniformly spaced grid a significantly large number of view-aligned slices may be needed to capture the fine details in a mesh if uniform spacing between the polygon slices is maintained. The number of slices may be reduced, while the image quality preserved, if appropriate adaptive spacing between slices is employed. However, determining the appropriate spacing interval is non-trivial, and may still lead to over-sampling, except for the case when viewing the volume along one of the primary coordinate axes. In this case the polygon slices become aligned with the grid and the best spacing may be inferred directly from the grid's sampling.

An alternative to view-aligned slicing is to use axis aligned slices, as is required by older graphics cards supporting only 2D textures. The advantage of this approach when rendering stretched grids is that the correct sample spacing between slices can be maintained for arbitrary viewing angles. Figure 6 compares axis-aligned rendering with view-aligned rendering using the same number of slices. Clearly, with non-uniformly positioned axis-aligned slices, the re-

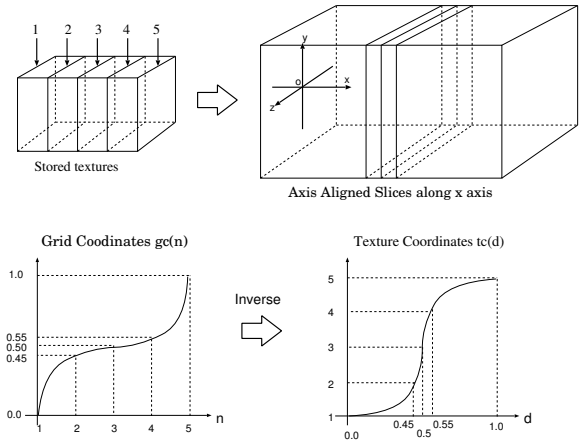

Figure 5: *A 5-slice example illustrating correct sampling of a stretched grid. Slice coordinates are obtained by 1D texture lookup. For each coordinate axis, a 1D texture stores the corresponding grid coordinates. The texture coordinates are derived from an inverse function*

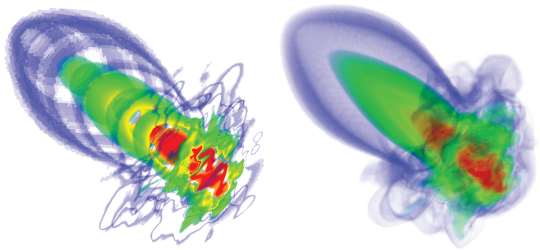

Figure 6: *Comparing view-aligned rendering (left) and non-uniformly axis-aligned rendering (right). The number of slices used for both images are the same.*

sulting images are much better as more slices are used in the region of higher sampling density in contrast to the placing of uniformly-spaced, view-aligned slices. Specifically, every grid point is rendered on a slice that correctly samples the volume along the viewing axis. In addition, axis-aligned slicing is computationally less expensive since intersection points between the volume and slices do not change. The top images in Figure 5 demonstrate non-uniform slicing for a simple 5-slice case.

When volume rendering with non-uniformly spaced samples along the viewing direction, care must be taken to correctly handle opacity integration. The conventional back-to-front compositing operation in volume rendering, suitable for uniform sampling, can be described as:

$$\alpha_{out} = \alpha_i + (1 - \alpha_i)\alpha_{in}$$

where α_{out} is the resulting opacity, α_i is the opacity at the sample position i, and α_{in} is accumulated opacity before

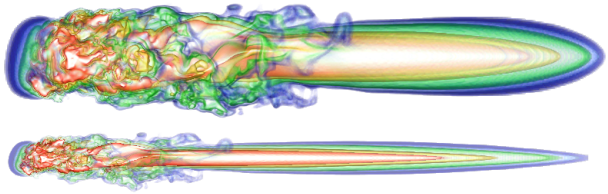

Figure 7: *Volume rendered images with uniform (top) and correct, non-uniform sampling (bottom).*

reaching the sample position i. For non-uniform slices, we must adjust α_i since attenuation depends on the distance between two neighboring samples. At each sample point i, α_i should be corrected as follows:

$$\alpha_{new} = 1 - (1 - \alpha_{original})^{d_i}$$

where d_i is the average distance between the current slice and the two immediate neighboring slices. Figure 12 compares with and without opacity correction in the rendering.

Adapting the spacing between polygonal slices as just described permits us to correctly sample the volume along a single axis. Correctly handling the non-uniform sampling along the remaining two axes requires additional measures. To address this issue, we again make use of programmable graphics hardware, mapping the uniformly stored textures to non-uniform locations on each of the slicing polygons by employing a fragment program. We accomplish this mapping by employing non-uniform texture coordinates, which are pre-calculated by computing the inverse of the function that defines the grid coordinates for each major axis, and then stored as three 1D textures. The bottom images in Figure 5 illustrate this inverse mapping. During rendering, the fragment program accesses the coordinate textures and uses their contents as the coordinates for indexing the original scalar data textures. Thus the stretched grid is correctly rendered through a combination of positioning slicing polygons appropriately in space to address the non-uniform sampling along the principal viewing axis, and by using a simple fragment program and 1D textures to accommodate the non-uniform sampling along the remaining two axes.

Figure 7 compares volume rendered images of stretched grid data with uniform and correct, non-uniform sampling. In the former case the non-uniform grid spacing is simply ignored. The volume aspect ratio changes from 1:1:4 to 1:1:2. Notice that image quality is maintained with the correct non-uniform sampling.

7. Test Results

We have implemented the compression-based volume visualization strategy and tested it using a uniformly-sampled turbulent jet data set and the non-uniform NCAR plume data set. Experiments were performed on a Pentium4 3.06GHz

Table 2: *The time in seconds that it takes to pack one time step for different subvolume sizes.*

Subvolume size	2^3	4^3	8^3	16^3
63×63×256	0.13	0.10	0.07	0.05
126×126×512	0.85	0.45	0.30	0.26
252×252×1024	6.6	3.4	2.3	1.8
504×504×2048	91	36	17	14

Table 3: *The sizes of the address texture for different subvolume sizes to pack a 252x252x1024 volume from the plume data set.*

subvolume size	2^3	4^3	8^3	16^3	32^3
lookup table size	100M	12.6 M	1.57M	196k	12k

PC with 2GB of memory and an NVIDIA 6800GT PCI Express graphics card, which has 256MB of video memory. All rendering was performed to a 512×512 pixel image.

Table 2 shows the time to pack one time step of the NCAR data for different resolution levels and different subvolume sizes. Packing the entire 400 time-step plume data set could therefore take tens of minutes. We view this cost as acceptable since packing is usually done for a selected subset of the mid-level data.

Figure 8 shows the compression rate achieved using varying subvolume sizes for each time step of the *omh* variable, the horizontal component of vorticity derived from the NCAR plume data set. The savings decreases as the flow fills more of the spatial domain in later time steps as can be easily seen in Figure 13. Later time steps also correspond to an increased dynamic range of the scalar values over time as shown in Figure 9. For most of the subvolume sizes significant savings are achieved. The optimal subvolume size for the plume data is 16^3, while using 2^3 results in a very significant storage overhead that outweighs any savings. Table 3 lists the storage overhead requirement for each subvolume size. For the 2^3 case, the lookup table alone consumes 100MBs of space. On the other hand, the overhead for 32^3 subvolumes drops to 12KBs.

We also studied the effectiveness of our packing scheme with the turbulent jet data set. Figure 14 shows selected time steps from this data set. Figure 10 displays the compression rate achieved. We observe that as with the NCAR plume data set, significant compression rates are achieved for each of the subvolume sizes except for the 2^3 case where overhead again outweighs any compression savings. The 8^3 subvolumes produces optimal results for all time steps. Lastly, we note that the compression rate remains more or less constant for the jet data. Unlike the NCAR plume data, the turbulent jet simulation has effectively reached steady state and the

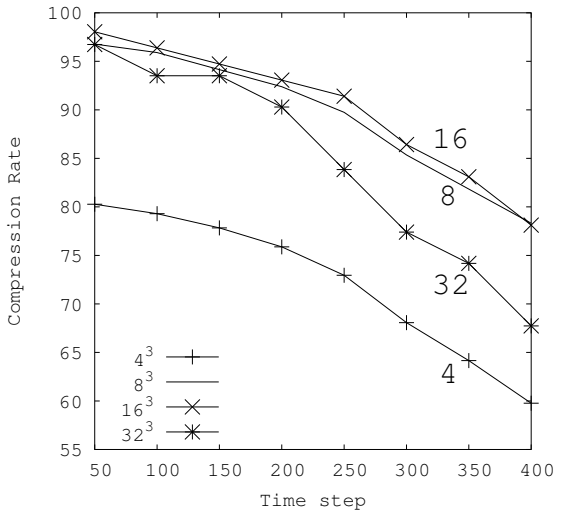

Figure 8: *compression rate for the 400 time-step NCAR data set for varying box sizes. Using a 2^3 box does not lead to any saving and is not shown.*

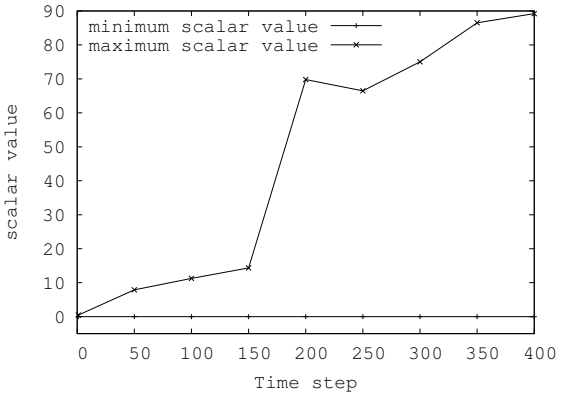

Figure 9: *Global minimum and maximum scalar value for each time step of the plume data set.*

volume of the region occupied by the turbulent feature of interest does not change as much as the plume data as can be seen in Figure 14.

In addition to evaluating compression, we also explored the rendering performance of the overall system for browsing in both spatial and temporal domains. In case of browsing in spatial domains, interactive browsing is possible for all four resolution levels by using fewer slices which can still reveal the overall structure of the volume. Once the spatial browsing is finished, the refinement process is performed to increase the picture quality.

Table 4 shows frame rates when browsing in the tempo-

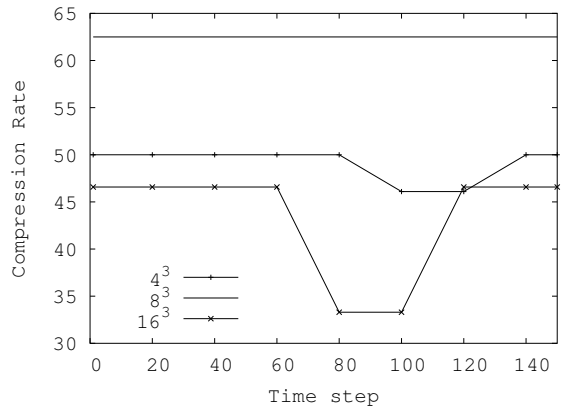

Figure 10: *compression rate for the 150 time-step turbulent jet data set for different subvolume sizes. The total size of packed data is the sum of the subvolume sizes and a lookup table. We do not obtain any savings by using either* 2^3 *or* 32^3 *subvolumes so their results are not shown.*

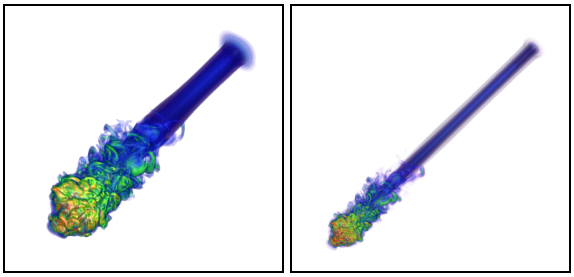

Figure 11: *Images showing a view used to obtain the performance numbers in Table 4. The left image shows the case when no warping is performed and the right image shows the case when warping is performed.*

ral domain. In this case, data must be constantly transferred from main memory or disk to the video memory. Timings were performed for both in-core rendering, where the data fit entirely into physical processor memory, and out-of-core rendering, where the data were processed from disk. In this set of tests axis-aligned slices were used for rendering and the number of slices was the same as that as the number of voxels. Using fewer slices can greatly increase interactivity, if desired. Level 3 numbers are not shown here because interactive viewing is no longer possible due to the data size. In level 1 we can see the benefits of packing since we can fit the entire time sequence in main memory and still maintain 1.8 frames per second, whereas without packing only 58 timesteps can be kept in main memory. In level 2, the benefits of packing is again obvious in terms of frame rate and memory bandwidth. We can also see from our test results that correctly sampling (warping) the stretched grid appears to slow down the frame rates somewhat. However, it

Table 4: *Frames per second for temporal browsing the NCAR dataset. The rendering modes correspond to whether the non-uniform spacing is considered or not, and whether data packing was used or not. The maximum number of timesteps that fit in main memory is shown inside the square brackets for each case. The starting timestep is 1. The viewpoint used is shown in Figure 11. Note that out-of-core numbers are not given when the entire 400 timesteps fit in the main memory.*

Level 0: 64×64×256

rendering mode \ data storage	in-core	out-of-core
No warping, no packing	9.0[400]	-
with packing	3.1[400]	-
warping & packing	2.5[400]	-

Level 1: 128×128×512

rendering mode \ data storage	in-core	out-of-core
No warping, no packing	4.0[58]	0.3
with packing	1.8[400]	-
warping & packing	1.7[400]	-

Level 2: 256×256×1024

rendering mode \ data storage	in-core	out-of-core
No warping, no packing	0.7[7]	0.3
with packing	1.3[197]	1.1
warping & packing	0.7[197]	0.7

is important to note that since the images with and without warping have very different screen coverages, comparing their timing results is generally not meaningful, as the fill requirements vary with pixel coverage.

Finally, we see from our test results that the cost of decompressing the data is relatively small, both for warped and the unwarped results. Responsive, if not interactive, frame rates can be maintained for most of the resolutions, regardless of whether compression or geometry correction (warping) is employed, or whether the data are in-core or reside on disk. We point out that the benefit of compression is the potential to store more data at various levels of the storage hierarchy.

8. Conclusion and Future Work

Achieving interactive visualization of large time-varying volume data requires efficient data transfer through a multi-level storage hierarchy from rotating disk to video memory, with each hierarchy component having varying bandwidth and capacity characteristics. To accommodate this hierarchy, we have developed a visualization system that relies on a combination of data reduction techniques that allow us to fit more data into lower capacity storage components and transfer data between hierarchy levels more quickly. These data reduction techniques, one wavelet-based, and the other a texture packing mechanism, are simple to implement and per-

mit the user to make effective speed/quality trade-offs. In the case of our texture-map based compression method, the unpacking (decompression) procedure is hardware acceleratable using today's programmable GPUs, while our wavelet-based multiresolution scheme is currently executed entirely on the CPU.

We have also extended the standard texture volume rendering technique to handle non-uniformed, *stretched* Cartesian grid data, and coupled the rendering with our GPU-accelerated unpacking step. Both the hardware data unpacking and the warping of the stretched grid have minimal impact on rendering performance. We have demonstrated the utility of this system by interactively browsing through multi-terabyte, time-varying volume data sets using only a single PC and a commodity graphics card.

9. Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE), ACI 0222991, ANI 0220147 (ITR), and ACI 0325934 (ITR), and the U.S. Department of Energy under Memorandum Agreement No. DE-FC02-01ER41202 (SciDAC) and DE_FG02-05ER54817 (SciDAC) and under Lawrence Livermore National Laboratory Agreement No.: B523578 (ASCI VIEWS), B537770 and B548210.

References

[BCF03] BINOTTO A. P. D., COMBA J., FREITAS C. M. D. S.: Real-time volume rendering of time-varying data using a fragment-shader compression approach. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 69–76.

[Cly03] CLYNE J.: The multiresolution toolkit: Progressive access for regular gridded data. In *Proceedings of Visualization, Imaging, and Image Processing 2003* (2003), pp. 152–157.

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), pp. 7–15.

[LMC01] LUM E. B., MA K.-L., CLYNE J.: Texture hardware assisted rendering of time-varying volume data. In *VIS '01: Proceedings of the conference on Visualization '01* (2001), pp. 263–270.

[LMK03] LI W., MUELLER K., KAUFMAN A.: Empty space skipping and occulusion clipping for texture-based volume rendering. *Proceedings of IEEE Visualization 2003 Conference* (2003), 317–324.

[LPD*02] LINSEN L., PASCUCCI V., DUCHAINEAU M. A., HAMANN B., 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), p. 346.

[MSSS98] MA K.-L., SMITH D., SHIH M.-Y., SHEN H.-W.: *Efficient Encoding and Rendering of Time-Varying Volume Data*. Tech. Rep. ICASE Reprot No. 98-22, Institute for Computer Applications in Science and Engineering, June 1998.

[SBS02] SOHN B.-S., BAJAJ C., SIDDAVANAHALLI V.: Feature based volumetric video compression for interactive playback. In *Proceedings of Volume Visualizationa nd Graphics Symposium 2002* (2002), pp. 89–96.

[SCM99] SHEN H., CHIANG L., MA K.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the IEEE Visualization Conference VIS 99* (1999), pp. 371–378.

[SJ94] SHEN H.-W., JOHNSON C. R.: Difference volume rendering: A fast volume visualization technique for flow animation. *IEEE Visualization* (1994).

[SW03] SCHNEIDER J., WESTERMANN R.: Compression domain volume rendering. In *Proceedings of the Visualization 2003 Conference* (2003), pp. 293–300.

[WAA00] WATERFALL A. E., ATHERTON T. J., ANAGNOSTOU K.: 4D volume rendering with the shear warp factorisation. In *Proceedings of the 2000 IEEE Symposium on Volume Visualization* (2000), pp. 129–137.

[Wes95] WESTERMANN R.: Compression domain rendering of time-resolved volume data. In *VIS '95: Proceedings of the 6th conference on Visualization '95* (1995), pp. 168–174.

[WV94] WILHELMS J., VAN GELDER A.: Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings of the 1994 Symposium on Volume Visualization* (October 1994).

[WWS03] WOODRING J., WANG C., SHEN H.-W.: High dimensional direct rendering of time-varying volumetric data. In *Proceedings of Visualization 2003 Conference* (October 2003), pp. 417–424.