UC Davis

IDAV Publications

Title

An Interactive Volume Visualization System for Transient Flow Analysis

Permalink

<https://escholarship.org/uc/item/3rf4936r>

Authors

Rosa, Gabriel
Lum, Eric
Ma, Kwan-Liu
et al.

Publication Date

2003

Peer reviewed

An Interactive Volume Visualization System for Transient Flow Analysis

Gabriel G. Rosa, Eric B. Lum, Kwan-Liu Ma, ¹ and Kenji Ono ²

¹ University of California at Davis
Email: {rosa, lume, ma}@cs.ucdavis.edu

² University of Tokyo
Email: keno@iml.u-tokyo.ac.jp

Abstract

This paper describes the design and performance of an interactive visualization system developed specifically for improved understanding of time-varying volume data from thermal flow simulations for vehicle cabin and ventilation design. The system uses compression to allow for better memory utilization and faster data transfer, hardware accelerated rendering to enable interactive exploration, and an intuitive user interface to support comparative visualization. In particular, the interactive exploration capability offered by the system raises scientists to a new level of insight and comprehension. Compared to a previous visualization solution, such a system helps scientists more quickly identify and correct design problems.

Keywords: 3D texture, compression, flow visualization, graphics hardware, interactive visualization, PC, scientific visualization, transient flow simulation, volume rendering, volume lighting

1. Introduction

High performance computing and interactive visualization techniques are changing the way we conduct scientific research and engineering design. Our ability to study and understand complex, transient phenomena is particularly critical to solving many scientific and engineering problems. A typical time-varying data set from a computational fluid dynamics (CFD) simulation can have from hundreds to thousands of time steps and each time step can contain millions or even billions of data points. Generally, multiple values are stored at each data point. As a result, a single data set can easily require from gigabytes to even terabytes of storage space which creates tremendous challenges to the subsequent visualization and data understanding tasks.

Scientists' ability to visualize time-varying phenomena is

absolutely essential to ensure correct interpretation and analysis, to provoke insights and to communicate their findings with others. The key is to allow them to freely and *interactively* explore in both spatial and temporal domains, as well as in the visualization parameter space. The resulting animation sequences can illustrate how the underlying physical structures evolve over time.

A starting point of the work reported in this paper is a low-cost interactive rendering technique developed for visualizing time-varying volume data on a PC or a cluster of PCs ⁴. The basis of this technique is an integrated hardware-accelerated decoding and volume rendering method. Based on this technique, we have then built a visualization system tailored to the visualization requirements of the thermal flow simulation for car cabin and ventilation design ⁷. In this paper, we describe the changes in the design and implementation of the system, as well as their impacts on the system performance and usability.

The main difference between the base system and the current work is the switch from using 2D texture to 3D texture. This difference has a significant implication to the overall system design. While the image quality is improved by the enlarged filtering space, maintaining fast data transfer be-

tween the main memory and video memory becomes a challenge due to varying differences in bus texture performance and scalability.

Additionally, realtime lighting is also implemented, using two different methods. One for standard hardware and one for the next generation of programmable graphics processors, which allows for very complex lighting to be applied in-realtime.

Our compression system allows for timestep consolidation, minimizing data transfer. However, amortizing this cost across consolidated frames is more difficult without frame interlacing, a feature of the previous system that is closely related to 2D texturing. Without careful planning, the system can experience significant performance degradation compared to previous work. We show here some of the specific issues regarding the system and how they can be overcome with some creative use of hardware functionality.

We also introduce features of a user interface designed to facilitate comparative visualization. Our ultimate goal is to couple the simulation code with the visualization code to allow for simulation-time tracking and steering.

2. Related Work

Time-varying data visualization has been an active research area. Various approaches have been proposed to reduce the storage, corresponding I/O, and rendering demands for visualizing time-varying data in a more efficient way.

Using proper encoding and exploiting temporal coherence or spatial coherence or both, the storage requirements and rendering cost of the data may be significantly reduced. Shen and Johnson¹⁰ take advantage of temporal coherence, using difference encoding to significantly diminish storage and rendering requirements. Westermann¹³ performs wavelet encoding of each time step separately to generate a compressed multiscale tree structure. Further compression can be obtained by examining the resulting tree structures and wavelet coefficients. Ma and Shen⁶ discuss how non-uniform quantization along with octree and difference encoding can be employed to speed up rendering of time-varying volume data. They show that the octrees for consecutive time steps can be merged to share subtrees. Consequently, during rendering, partial images built from subtrees that have not changed over time may be reused in later time steps.

Wilhelms and Van Gelder¹⁵ design hierarchical data structures for controlled compression and volume rendering. They extend octrees and a branch-on-need (BON) subdivision strategy¹⁴ to handle multi-dimensional data. Sutton and Hansen¹¹ propose a temporal branch-on-need tree (T-BON) as an extension to the 3-D BON tree for time-varying isosurface extraction.

Shen, Chiang, and Ma⁹ introduce a hierarchical data

structure, called Time-Space Partitioning (TSP) tree, for a better utilization of both spatial and temporal coherence. In essence, the skeleton of a TSP tree is a standard complete octree, which recursively subdivides the volume spatially until all subvolumes reach a predefined minimum size. To store the temporal information, each TSP tree node itself is a binary tree. Every node in the binary tree represents a different time span for the same subvolume in the spatial domain. Most importantly, TSP trees allow the renderer to use data from subvolumes of different spatial and temporal resolutions, which is not possible for 4-d octrees. The TSP tree data structure has been also used by Ellsworth, Chiang, and Shen² to facilitate large scale volume rendering using 3-D texture hardware.

Further speed up of rendering may be accomplished by utilizing parallel computers or graphics hardware. However, even though a parallel computer can render images at multiple frames per second, without high-speed network and parallel I/O support, two bottlenecks can still make it impossible to achieve interactive viewing; one bottleneck is the need to stream large volume files throughout the course of the visualization process; the other is the delay due to transferring the resulting images over a potentially non-dedicated network. Ma and Camp⁵ develop a post-processing parallel visualization strategy based on a pipelined rendering. They demonstrate remote visualization of time-varying volume data on a PC cluster over a wide-area network. Pipelining and careful grouping of processors are used to hide I/O time and to maximize processor utilization. Visually lossless compression is used to significantly cut down the cost of transferring output images from the PC cluster to a display device through a wide-area network. Clyne and Dennis¹ employ similar techniques, using double buffering to help mask both the costs of volume data I/O over a high-bandwidth channel and hide image transmission over a TCP/IP network.

The methods introduced in this paper can work as well, and provide an elegant way to efficiently render visually high-quality images. Furthermore, they have the added advantage of being scalable and running on low-cost, commodity hardware. This method makes full use of hardware features, making them far more accessible to a broad range of researchers.

3. Driving Applications

While in principle the reported system can be used to visualize any time-varying volume data on a Cartesian grid, the driving application for building the system is the thermal flow simulation for vehicle cabin and ventilation design⁷. The simulation is for studying how the flow that blows from a heater duct under the front seat of the car propagates to the back seat area and circulates around to the rear passenger. Figure 1 shows the surface geometry of the car cabin. The front seats were removed to allow unobscured view of the

Figure 1: The surface model of the car cabin with front seats removed for viewing clarity. The dark lines mark the computational domain.

rear seating area. Improved understanding of the flow pattern would help refine the design of the shape and location of the duct to maximize the efficiency of the heater.

A key requirement of this system is the ability to render the volumetric flow data integrated with the underlying geometry. While this reduces available memory for the volumetric rendering, and adds to the polygon count, thus reducing framerates in the spacial manipulation of the data, it adds depth to the scientists' understanding of the data and its application.

A typical data set generated by the simulation contains thousands of time steps. Visualization involves rendering of time-varying volume data defining the evolving flow field and the surface data defining the car cabin. The current spatial resolution of the data is $256 \times 256 \times 128$ but higher resolution calculations are underway. A total of five variables are stored in the data set. They are three velocity components, pressure, and temperature. In this work we focus on the visualization of temperature distribution.

Prior effort for the visualization of the simulation relied on RVSLIB which is a vectorized volume rendering code¹². By coupling RVSLIB with the simulation code, it is possible to perform flow simulation and visualization simultaneously, and generate sequences of rendered images with a pre-defined transfer function. For $256 \times 256 \times 128$ voxels, rendering time was roughly 18 hours for 1560 frames (640×480 pixels) on an NEC SX5. In addition, it took another 30 minutes to produce an MPEG1 movie from stored image files on an SGI Onyx2 Reality Monster⁸.

Thus, there was no interactive visualization capability to allow easy exploration of the flow field in the previous setting. It would take several rounds of batch-mode processing

to find an adequate transfer function and view point to derive a good animation revealing the specific flow behaviors that the scientists looked for. We have therefore developed an interactive visualization solution in which neither a large memory space nor multiple (or powerful) processors is required. Instead, this solution utilizes the commodity PC and graphics card that are highly available and low cost.

4. An Interactive Volume Visualization System

4.1. Basic algorithm

The basic algorithm is a derivative of the work presented by Lum, Ma and Clyne⁴. The algorithm is extended and altered to use 3D texturing hardware, both for the enhanced filtering and to accommodate the shift in newer hardware towards 3D texturing. Use of multiple texture units (multitexturing) is also included as an alternative to previous 2D-texture oriented techniques and to alleviate the polygon-render bottleneck.

The algorithm exploits features in current commodity PC hardware that resembles and supersedes the capabilities found in previous generation high-end graphics workstations. By using these features we eliminate some of the bottlenecks and limitation with hardware assisted volume rendering; in particular, we reduce the use of the low performance bus found in the PC architecture.

Compression

The core of the system is the compression and temporal encoding. Without compression it becomes unlikely to achieve interactive rendering speeds. Hardware assisted volume rendering requires that the volume be downloaded into the graphics memory before it can be rendered. For a single timestep this is not a problem, but interactive rendering of multiple timesteps requires multiple transitions over the slow graphics bus. By compressing the data before sending it across the bus, we increase the available bandwidth and manageability of the data, allowing for better use of hardware resources.

However, note that general purpose compression, while simple and effective, does not work well for this application. Instead, we must consider an alternate method of compression/decompression. While we could trivially compresses the data, decompression becomes a challenge considering the specialized GPU, with its limited set of instructions. As mentioned in previous work⁴, we could attempt to use the register combiners and other programmable features of modern GPUs, but those work in RGBA space, not in indexed mode. Storing the volume as index values instead of RGBA quads has numerous advantages, namely it reduces the required storage by a factor of four and allows the use a palette indexing scheme. This in turn produces a cheap solution to the problem of reclassifying the volume after a change in the transfer function is performed.

Palette based decoding

Instead of decoding the data, we make use of hardware support for indexed 3D textures, and reloading shared palettes to more render the volume. Our goal, interactive manipulation of the transfer function space and temporal domain, can be achieved more efficiently if the data does not have to be resent across the graphics bus after a change in classification. By retaining the data as index values instead of RGBA quads, we can consolidate the decompression step and the rendering step into one.

Further, by compressing various timesteps into one chunk of data, it is possible to create palettes in such a way that allows changing only the small palette to achieve a change in the temporal domain, not changing the entire 3D texture. By cycling the display palette we can achieve simple animations. Coupled with hardware tri-linear filtering of the RGBA values resulting from the index lookup, this becomes a cost-effective method of cycling through multiple time steps with reasonable fidelity.

Temporal Encoding

A Discrete Cosine Transform (DCT) can be applied to the data using as many timesteps as desired. The DCT does not actually compress the data, per se, but packs the data more efficiently, reducing multiple steps into a decodable sequence where more energy can be allocated towards temporal or spatial accuracy as desired. The larger the number of timesteps transformed, the higher the compression factor achieved. However, the higher the compression rate, the lower the fidelity of the rendering. Either the data will lose spatial fidelity or temporal fidelity, whichever one is sacrificed in the bit-allocation step. The reverse DCT could be used to decompresses the data once it is in video memory, however, an alternate method is presented that makes more efficient use of hardware features and virtually eliminates the need to expand the data. This same scheme works for both 2D and 3D texturing methods.

The data is quantized using Lloyd-Max quantization to reduce the mean square error. In the original work⁴ each slice could be quantized independently, due to the nature of 2D slicing. In the current work, the volume is quantized timestep by timestep as a whole. While quantizing each slice separately can result in better image quality, the current system already makes use of higher quality filtering. It follows that quantizing slices independently results in higher temporal fidelity, but tri-linear filtering allows for better spacial fidelity. In practice, it is difficult to compare the two methods, as they both differ from the original data in different ways.

Optimizations

The original work allowed the data to be temporally interlaced, reducing and amortizing data retrieval costs, as only one fraction of the data had to be sent across the bus. This

increased system frame rate and responsiveness. This was achieved by splitting up the data into slices. As the system used 2D texturing to render the volume, this was a trivial optimization.

However, with 3D texturing the volume is usually kept intact. When transferring data across the bus, the entire texture is sent. This, of course, is not a requirement. Using sub-volume operations, any chunk of the texture can be replaced. In effect, we can still slice up the volume much like if using 2D hardware. In the future, this approach should become more powerful than the 2D texture temporal interlacing, as the freedom of movement gained from using 3D textures allows the volume to be interlaced in a variety of manners. For example, an octree-like structure could be used to only interlace certain regions of the volume, while volume slabs could be oriented in any direction, not just axis-aligned. This could lead to multiple new ways to amortize data transfer costs.

Alternatively, multitexturing hardware can be used to amortize the data transfer cost by blending in future keyframes with the current display. For example, a high-cost step (a key frame) may be pre-fetched and blended in ahead of time to reduce the impact on framerate should that texture not be resident in video memory. The throughput of the system does not increase, but the displays becomes smoother to the user as the perception of a jump from frame to frame is reduced.

Multitexturing hardware can also be used to reduce the polygon count in the rendering. Rendering very large datasets may suffer from a texture fill rate problem. That is, the video card may not be able to draw and texture enough polygons to maintain an acceptable frame rate. By using multiple units, we can trim down the number of texels to render by reducing the number of polygons the system has to render. Instead of using one polygon to render each slice, multiple slices are combined on to a single polygon.

With a growing trend for longer pipelines and more texture units onboard video cards, using multiple textures units for volume rendering seems to be an effective way to trim down the rendering process, by consolidating steps and allowing the hardware to be used to its full extent.

4.2. Lighting

Hardware volume lighting is accomplished by using the pixel shading capabilities of newer programmable GPUs such as the GeForce2 and newer cards. Lighting, while greatly increasing the visual quality of volume rendered images by providing subtle depth cues and feature highlighting, also introduces the problem that normals must be either stored or calculated on-the-fly. For a traditional volume renderer this matter is of very little concern, as the normal map can be loaded prior to visualization and texture lookups

Figure 2: Left: Unshaded. Right: Shaded. Shading reveals a more natural outline of the volume.

within the shader program can be used to access the normals. Namely, the extra storage space and additional texture lookups are inconsequential to the performance of the system. This is not the case with a high performance time variant system. Several problems are immediately noticeable: storing a normal map for each timestep is prohibitively expensive, and cancels out the compression gains from using a palette based decoding system. Two solutions are presented that allow a more performance conscious application of shading without sacrificing quality.

The same palette compression method that is used for the volume data can be applied to the normal map. By creating an index into a hardware table, we can encode tri-value normal information into a single byte that is converted into a vector by the hardware. The texture lookup in the pixel shader will return an RGB value that represents the normal. Additionally, the same DCT compression system of having a key texture and sequential palette variations of that texture works just as well for normal information. Although, just as with the image compression, using a higher compression factor will result in greater deviation from the uncompressed version.

A second method is to calculate the normal information on the fly. On newer hardware, it is possible to make enough texture lookups in a single pass to extract gradient information based on neighbouring texels. However, a problem is immediately obvious. The pixel program does not have access to the index value of a texture, since the texture lookup is done in RGBA space. Thus, it is less trivial to extract the gradient information without knowledge of the data. The solution is dependant on the fact that the translation from index to color data is known to us. By using a linear grayscale color map for the normal map lookup, from 0 to 255, we can use either R, G, or B component of the colorspace to identify the index value of that particular texel. The index value will simply translate into a unified RGB value of it-

self. By sampling the 6 nearest neighbours, we can calculate the gradient of that voxel and apply lighting calculations accordingly. No normal map is required, and so no extra storage is needed. The performance impact is absorbed by the video card's ability to apply per-pixel effects, and not by the bottleneck of transferring data across the AGP bus. Note the necessity to perform at least 7 texture operations, 1 for each neighbour and 1 for the texel being rendered, limits the use of this method to cards with 7+ texture units. Fortunately, these are becoming increasingly common.

4.3. User interface

One of the requirements in visualizing large scale time-varying data is the ability to compare, side by side or in some effective manner, the data at various stages, both in the temporal and transfer function domains. Similarly, the user must be able to adjust the transfer function and spatial domain to allow for an optimal visualization as the temporal domain changes. The key is to allow the user to establish an initial visualization as quickly as possible. If the user would need to spend more than a few seconds to derive an approximate transfer function for the current timestep, then the system effectiveness is reduced.

Our system provides the user with an easy to use, but not necessarily completely accurate, way to sketch out a transfer function. Once the initial image visualization is established, the user is allowed to fine tune the transfer function, and other visualization parameters in both the temporal and spatial domains. The system also helps keep track of the transfer functions and visualization parameters the user choose to use for different time steps.

User intuitiveness and low learning curve are always primary considerations concerning user interfaces. By using a prototyping method, we have narrowed down interface issues and at the same time tailored the interface based on

specific user concerns for visualizing flow fields. In particular, having multiple view ports, each with its own associated set of controls that can be globally linked allows the user to effectively explore the transfer space and visualize the changes between timesteps and highlighting differences between transfer functions over time. On the other hand, having multiple viewports and sets of controls could be overwhelming to the user. As a result, it becomes a necessity to allow these to be disabled or hidden at the user's request.

4.4. Implementation

Our system was implemented and tested on an Athlon 1.53Ghz machine with 1GB of RAM, and a high speed (7200 RPM) IDE disk. The main concern with commodity PC hardware is eliminating the bottlenecks, most predominantly the AGP bus and the disk access times. Therefore, our implementation focuses on maintaining data in RAM as much as possible, using a background thread to bring in data from disk. By keeping data in memory, disk reads can be amortized by allowing the disk IO to happen as the system renders unrelated steps. As the user progresses temporally forwards or backwards, the memory cache can shift accordingly.

The spatial rendering is done with 3D texturing hardware. The development system contains a GeForce 4 4400 video card with 128MB of RAM. Since the volume is stored in RAM as index values, data can be sent directly to the video card ready for rendering. Typically, a set of N slices (where N is the greatest volume dimension) are created and textured in the view direction. While this approach leads to very fast, relatively good quality volume rendering, it can lead to a bottleneck in how fast the video card can display and texture polygons. Thus, we try to reduce the number of operations performed with the texture hardware; by making use of multitexturing hardware the pipeline is optimized resulting in more responsive spatial rendering of the data. Hardware blending is used, which offers a fast but not quite as high quality alternative to software methods. This is acceptable as the goal of this system is to allow interactive visualization.

The current geometry, a model of the cabin floor, seat and a pair of legs, contains in excess of 29,000 triangles. By making use of hardware features, the geometry can be displayed with little reduction in the frame rate. However, the memory used up by the polygons still reduces the amount of memory available for volumetric rendering.

Although the system can distribute the rendering across multiple CPUs (in a multiprocessor workstation) this method relies mostly on bus speeds and the graphics hardware, and does not offer significant advantages to splitting up the work across processors. Instead, the concurrency is used to allow the user to maintain interactivity with the interface while the data processing occurs in the background.

Finally, distributing the work across multiple nodes in a cluster, can be very effective, as the system can make good use of increased video memory. Although the current work has not been expanded to use a PC cluster, the previous work showed that the algorithm is easy expandable to and functions well in a multi-node configuration ⁴.

Lighting was implemented using both explicit precalculated normals and on-the-fly normals methods discussed earlier. The pixel shader program were written in the Cg language, using the Nvidia Cg Toolkit.

Using the stored-normal method, we still see a minor degradation in performance in the temporal domain, yet no performance degradation in the spatial domain. Lightning calculations are relatively simple, and the videocard can handle it in realtime. However, the visual quality of the rendered image is much improved. A greater sense of depth and features are visible even with a single point lightsource.

Using the on-the-fly method, performance testing was not possible with the current card. Using NV30 software emulation, it is possible to verify that the image is rendered, but at software speeds. With the next generation of cards, on-the-fly lighting should become very attractive, especially since lighting will come at virtually no extra cost.

Although the current system offers a single point lightsource as proof-of-concept, very advanced techniques with multiple finite geometric light sources or non-standard lighting models can be easily integrated into this system. In particular, non-photorealistic methods and high-fidelity methods are attractive, using the programmable hardware to its full potential.

5. Results

We have studied the performance of the current system and obtained encouraging results. All results presented are based on the use of 4 times compression, that is, our system relies on a keyframe every 4 frames, while using differently encoded palettes to extrapolate the next 3 frames from the base. Higher compression factors are obtainable, but this yields the best balance of visual results and performance. First was measured the time for animation in the temporal domain with a stationary view. Rendering that is purely based on palette switching results in over ten frames per second. Rendering data already in the video memory is slower but still reaches eight frames per second. It is clear that the system functions most optimally when engaged in such sequential animation, most noticeably so when between key-volumes. Note, however, that this is expected behavior. As per the algorithm, the system relies on amortizing this cost to maintain reasonably high frame rates.

If data must be transferred across the bus, an average three frames per second is achieved which does introduce an obvious jump in frame displaying. Rendering with or without

the cabin surface geometry data makes very little difference in frame rates; test results show that with the geometry data rendering slows down by no more than one frame. Figure 5 displays selected frames from a stationary-view animation. It is clear from the visualization that the high-temperature flow from the heater duct moves toward the left foot of the passenger. Figure 3 shows different views of three different time steps.

When animating in the spatial domain only, realtime rendering rates can be maintained. The user can thus freely change view angle, take a close-up view, and modifying color or opacity transfer functions, and immediately see the corresponding results. Figure 4 shows visualizations emphasizing on different ranges of values.

The timing data indicates significant coherence with the algorithm expectations. The spatial manipulation, rendering of the geometric data and the manipulation of the transfer function do very little to affect the frame rate. In practice, animating through a large volume, while not as fast as the 2D texturing method, is relatively smooth. The frame rate changes very little unless a non-linear jump in the timesteps is performed. As discussed earlier, biggest performance hit comes from transferring data across the bus. The system can suffer from a frame-rate reduction at certain points. The issue becomes apparent at each N th frame, where N is the compression factor. Since the compressed frames are consolidated into one, once the graphics memory is exhausted a new timestep must be brought in from cache. Thus large amounts of data may be sent across the bus, resulting in a decrease in the framerate. Presently, it takes anywhere from 0.3 seconds to 1.3 seconds to load a $256 \times 256 \times 128$ texture into the video memory. This issue was also apparent, under slightly different circumstances in the original work. Since the system makes heavy use of palette animation, moving between timesteps in a non-linear fashion results in reloading of the data, also causing a decrease in frame rate. However, frame interlacing removed this problem with linear movement. Since the data loading cost was amortized by loading $1/N$ th of the dataset at a time, the cost of sending data across the bus was not noticeable to the user. With the 3D texturing method, as previously discussed, this becomes more of a challenge. Our approach would become very attractive after this bandwidth or latency problem is fixed or perhaps a more effective amortization scheme is developed to counter it.

6. Conclusions

Through the use of high-performance computing and interactive visualization, it is possible to significantly decrease the turnaround time of both the simulation and the following visualization-based analysis task.

By providing an integrated, high performance (yet low cost) rendering system, we can enhance the visualization

process to allow comprehension of large scale time-varying data as a whole, not as timesteps. By allowing the user to explore the entire temporal space, we contribute to his ability to understand the data on a deeper level.

Our system allows the user to explore the flow dataset in an interactive fashion, while still maintaining good visual fidelity. Although not as high performance as the original 2D-texture oriented work, we feel that moving towards 3D texturing provides for a more elegantly designed system that can be geared to exploit new features in commodity hardware more easily and allows for a cleaner implementation, reducing the strain needed to maintain and add to the system.

7. Future Work

Future work will focus on exploiting hardware features that allow for partial volume replacement. In particular, volume coherence can be exploited to reduce the amount of data transferred. This should further extend the compression system, allowing for more interactivity for the user. In addition, further amortization schemes can be exploited from sub-volume coherence and optimizations. In particular, reducing the amount of data transferred will result in significant improvements in temporal animation performance.

Also, a key but somewhat overlooked problem with hardware volume rendering is the omission of lighting and shadows. Both of these elements allow for a deeper understanding of volume features, and allow the scientist to perceive the volume in a more familiar, more physically accurate way. Using hardware for volume shading is attainable with current GPUs that allow for pixel shader programs that make lighting lookups and adjustments on-the-fly³. As our current work incorporates basic lighting, future work will incorporate more extensive use of lighting to expose volume features. In particular, Non-photorealistic lighting methods are very attractive, as they allow very clear feature extraction and offer new insights into the data.

Finally, our ultimate goal is to combine both the data simulation with the interactive tools, so allow for real-time visualization of the simulation and allow the scientist to adjust simulation parameters interactively. By coupling both systems together, we can reduce the amount of data that needs to be stored, and allow for a much shorter turnaround time between data production and data analysis.

Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts ACI 9983641 (PECASE award) and ACI 9982251 (the LSSDSV program), and the U.S. Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC program) and No. B523578. The authors are grateful to their colleagues

Figure 3: Timesteps 302, 335 and 420 from different view points and zoom factors.

at the the Visualization and Graphics group at UC Davis for their assistance during this research.

References

1. CLYNE, J., AND DENNIS, J. Interactive direct volume rendering of time-varying data. In *Proceedings of Data Visualization '99* (May 1999), pp. b 109–120.
2. ELLSWORTH, D., CHIANG, L., AND SHEN, H.-W. Accelerating time-varying hardware volume rendering using TSP trees and color-based error metrics. In *Proceedings of 2000 Symposium on Volume Visualization* (2000), ACM SIGGRAPH.
3. LUM, E., AND MA, K.-L. Hardware-accelerated parallel non-photorealistic volume rendering. In *Proceedings of International Symposium on Nonphotorealistic Rendering and Animation* (June 2002).
4. LUM, E., MA, K.-L., AND CLYNE, J. A hardware-assisted scalable solution of interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics* 8, 3 (2002), 286–301.
5. MA, K.-L., AND CAMP, D. High performance visualization of time-varying volume data over a wide-area network. In *Proceedings of Supercomputing 2000 Conference* (November 2000).
6. MA, K.-L., AND SHEN, H.-W. Compression and accelerated rendering of time-varying volume data. In *Proceedings of the 2000 International Computer Symposium - Workshop on Computer Graphics and Virtual Reality* (December 2000), pp. 82–89.
7. ONO, K., FUJITANI, K., AND FUJITA, H. Applications of cfd using voxel modeling to vehicle development. In *Proceedings of the 3rd ASME/JSME Joint Fluids Engineering Conference* (1999). FEDSM99-7323.
8. ONO, K., MATSUMOTO, H., AND HIMENO, R. An application of volume rendering visualization to the thermal flow analysis in a vehicle cabin. In *Proceedings of the 3rd Pacific Symposium on Flow Visualization and Image Processing* (March 2001).
9. SHEN, H.-W., CHIANG, L.-J., AND MA, K.-L. A fast volume rendering algorithm for time-varying field using a time-space partitioning (TSP) tree. In *Proceedings of Visualization '99* (1999), IEEE Computer Society Press, Los Alamitos, CA.
10. SHEN, H.-W., AND JOHNSON, C. R. Differential volume rendering: A fast volume visualization technique for flow animation. In *Proceedings of the Visualization '94 Conference* (October 1994), pp. 180–187.
11. SUTTON, P., AND HANSEN, C. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of IEEE Visualization '99 Conference* (October 1999), pp. 147–153.
12. TAKEI, T., MATSUMOTO, H., MURAMATSU, K., AND DOI, S. Parallel vector performance of concurrent visualization system rvslib on sx-4. In *Proceedings of the 3rd Pacific Symposium on Flow Visualization and Image Processing* (March 2001).
13. WESTERMANN, R. Compression time rendering of time-resolved volume data. In *Proceedings of the Visualization '95 Conference* (1995), pp. 168–174.
14. WILHELMS, J., AND VAN GELDER, A. Octrees for faster isosurface generation. *ACM Transactions on Graphics* 11, 3 (July 1992).
15. 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).

Figure 4: Timestep 380. Left: rendered to show the low values. Middle: rendered to show the mid values. Right: rendered to show the high values.

Figure 5: Time steps 250, 300, 600, 650, 800 and 850 from a stationary-view animation. Rendering was done rendered without the geometry data and a transfer function highlighting the mid and high values.