Visualization Dot Com

Wes Bethel
Lawrence Berkeley National Laboratory
University of California, Berkeley
Berkeley, CA 94720

1.0 Abstract

In this article, we explore the seemingly well-worn subject of distance-based, or remote visualization. Current practices in remote visualization tend to clump into two broad categories. One approach, which we'll call render-remote, is to render an image remotely, then transmit the image to the user. Another option, render-local, transfers raw data to the user, where it is then visualized and rendered on the local workstation. With advances in networking and graphics technology, we can explore a class of approaches from a new, third category. With this third category, which we'll call shared, or "dot com" visualization, we stand to reap the best of both worlds; minimized data transfers and workstation-accelerated rendering. We will describe a prototype system called *Visapult* currently under development at Lawrence Berkeley National Laboratory (LBNL) that strikes such a balance, achieving a blended, scalable visualization tool. "Dot com" visualization means that remote and local resources collaborate and negotiate, combining capabilities to produce a final product.

2.0 The Brute-Force Approaches

Consider the following common scenario: you and your workstation are on the West Coast, but your data is on the East Coast, and you need to look at the data. What do you do? One option is to perform the visualization and rendering on the East Coast, and send an image to your workstation. The other option is to move the data, either the whole thing or just a subset, to the West Coast.

In the render-remote approach, you win because only a single image is sent across the network. Presumably, one expects at least an order of magnitude reduction in traffic when sending only the final image as compared to the cost of sending the raw data. The usability cost of this approach is the loss of interaction on the local workstation due to the sacrifice of local rendering capabilities. The workstation plays the role of image viewer. In order to achieve interactivity using the remote rendering model, one would expect a minimum of ten frames per second, using potentially upwards of 30 megabytes per second of raw bandwidth (1024x1024x24 bit uncompressed images). We're making a generous assumption: on the remote host, it is possible to perform visualization and rendering ten times per second.

Using the render-local approach, data is transferred to the local workstation where it is subsequently visualized and rendered. We stand to gain the interactivity lost in the render-remote model, assuming a reasonable amount of local graphics and processing horsepower. Troublesome areas inherent in the render-local model include potentially long download times, the possibility that a large dataset simply cannot be stored on the local workstation, and related issues.

What if we could combine the best of both approaches? In such a model, we wouldn't have to move the potentially large data volumes across the network, and we could take advantage of local workstation graphics. A blended model would facilitate the best use of resources; a large cluster, for example, could be used for computationally expensive parallel software volume rendering while the local workstation is used for interactive graphics.

3.0 Visapult: A Prototype Implementation

Visapult, our prototype implementation, is a visualization tool that implements distributed, shared and parallel visualization and rendering of large, time-varying scientific data sets. The focus of ongoing research and development targets three broad topics. First, Visapult implements a framework that is an application testbed for shared and parallel visualization algorithmic development. The volume rendering engine, described in the next section, uses a parallel, shared rendering model that scales reasonably well to accommodate large and hierarchical data volumes. Second, Visapult provides a testbed suitable for use with emerging technologies that implement "network

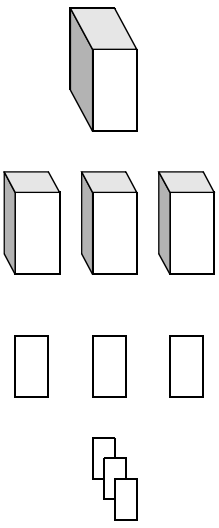
awareness." Finally, Visapult is used to address a "hard" but factual visualization problem - volume visualization of large and remotely located, time varying, adaptive and hierarchical scientific data that will not fit onto a workstation.

3.1 Distributed, Parallel, Shared Volume Rendering

The Visapult volume rendering engine is a parallel, distributed implementation of Mueller, et. al.'s Image Based Rendering Assisted Volume Rendering method [1], or just IBRAVR for short. The IBRAVR method leverages image-based rendering properties to achieve interactive, limited-range transformations of volume visualization on low-cost, commodity grade graphics hardware.

One of the many attractive properties of the IBRAVR model is that it will perform well on low-end workstation graphics, or even software, but will also run in high performance, immersive and stereo environments. What makes this possible is the decoupling of a computational back-end that performs software volume rendering from a front-end viewer that can run at interactive rates.

FIGURE 1. IBRAVR Task Decomposition



Volume data is first subdivided, with each processing element given the assignment of volume rendering it's subset of the volume.

The results, an image from each PE representing partial volume rendering, is sent to a viewer.

The viewer uses 2D texture mapping to render all partial images, and provides for interactive transformation.

In the IBRAVR model, a volume is decomposed into some number of "slabs" (Figure 1). Each of these slabs is separately volume rendered using whatever technique is handy to produce a single image. The resulting image is then used as raw texture data, and applied to either axis-aligned quads or quadmeshes. Quadmeshes are used to create a "terrain-style" elevation map for each of the textures, and provide more depth cues than flat quadrilaterals. Multiple texture maps are created from subsets of the volume so that the viewer may rotate the entire model for inspection. The IBRVR method works well, but within a limited range of rotation. Mueller et. al. claim a rotation range of about thirty-two degrees before visual degradation occurs [1], although this threshold may prove to be data-dependent. Increasing the number of texture maps may increase the threshold, while decreasing the number of texture maps will decrease the threshold.

3.2 Distributed IBRAVR

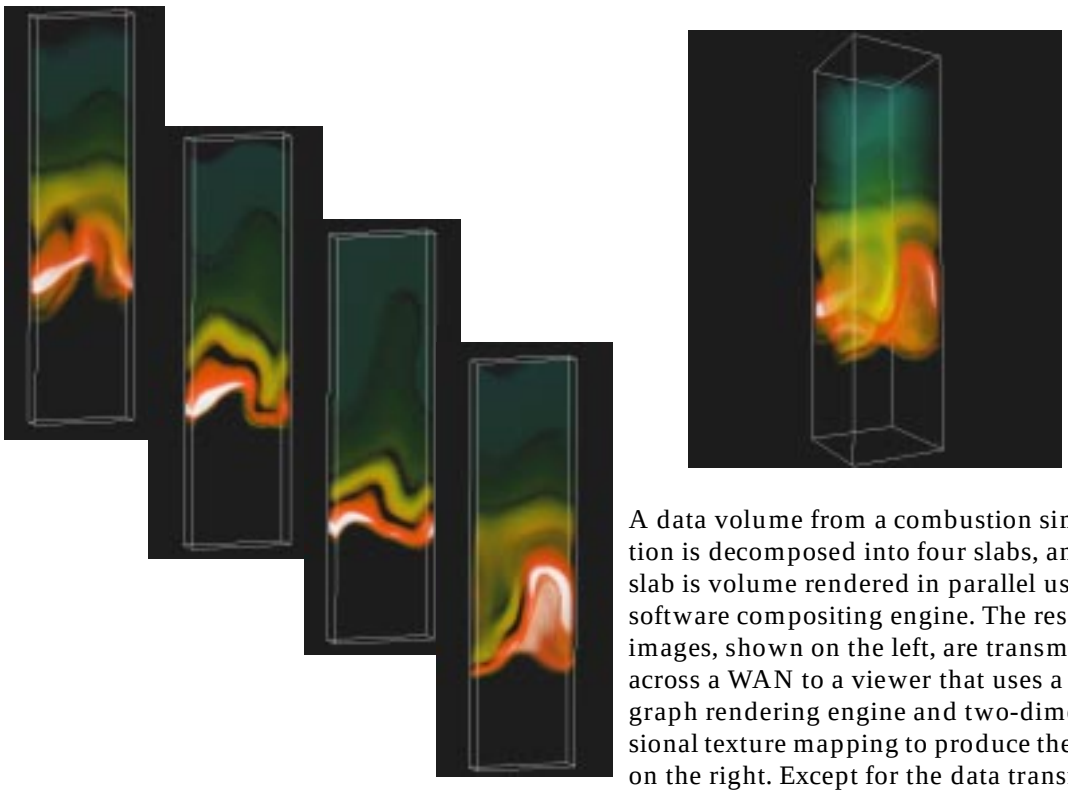
The IBRAVR model maps nicely to an object-order decomposition for parallel rendering [2]. The primary difference between the IBRAVR method and traditional object-order parallel software volume rendering lies in the design of the partial image recombination, or *gather* stage of the parallel rendering operation. The intermediate images produced by each processor, each of which renders a subset of a volume, must be composited together in a specific order to produce a final image. Algorithms for the image recombination stage of parallel software volume rendering have been the subject of much study [2].

Our IBRAVR implementation uses a pool of processors that perform object-order, parallel volume rendering in software. Rather than recombine the intermediate images in software, the partial images are "combined" using low-cost graphics hardware that supports two-dimensional texture mapping. By low-cost, we mean contemporary PC graphics cards that are in the $100-$250 price range. One of the fundamental ideas behind IBRAVR is that the image warping and depth-order compositing is performed using inexpensive graphics hardware. The image warping and inter-slice translation provided by texture mapping represents the image-based rendering aspect of the algorithm, while the depth-order rendering of semi-transparent 2D textures represents the image-gather and compositing stage of the traditional object order decomposition.

3.3 Visapult Framework

Our prototype is an application composed of two logical rendering components and one data component, all of which may be separated by a WAN.

A “back-end” volume rendering engine performs the object-order parallel volume rendering in software. It is written using MPI [3], and runs on a variety of distributed memory and shared memory machines. The second component is a viewer. The viewer is a lightweight interactive rendering application built from a OpenGL-based scene graph tool [4] that manages data and rendering services. The viewer is also a parallel application built using Pthreads[5]. The third component of the system is the scientific data and it’s management. In some cases, this might be as simple as a large disk farm connected directly to the volume rendering back-end, while in other cases, the data may be scattered across a WAN using a “network cache,” such as that implemented by the Distriubuted Parallel Storage System (DPSS) [7].



A data volume from a combustion simulation is decomposed into four slabs, and each slab is volume rendered in parallel using a software compositing engine. The resulting images, shown on the left, are transmitted across a WAN to a viewer that uses a scene graph rendering engine and two-dimensional texture mapping to produce the image on the right. Except for the data transfer, both viewer and back-end rendering execute asynchronously.

FIGURE 2. IBRAVR Applied to Combustion Simulation Results

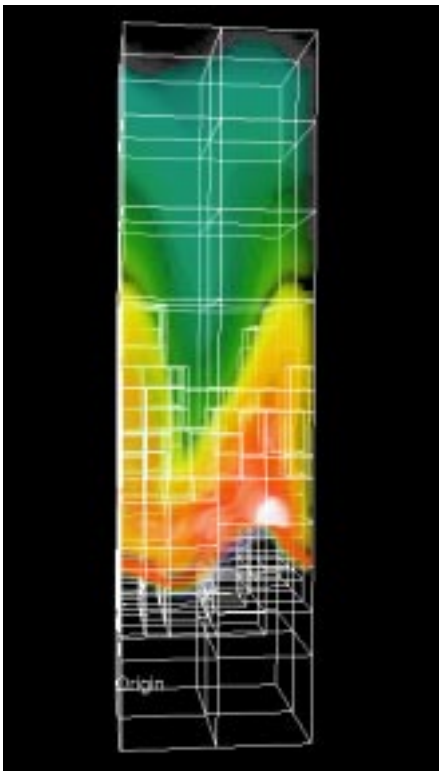
The volume renderer and the viewer communicate over a custom IPC layer built using TCP sockets. The protocol implemented by the prototype might be considered a *visualization communication protocol*, similar in some respects to the scene description and payload model described by the MPEG-4 specification [8].

For volume visualization, the payload between viewer and software renderer consists of two-dimensional texture maps containing the intermediate results of partial volume rendering. Our implementation uses a striped-socket model, where multiple back-end processing elements communicate with multiple threads in the viewer. Figure 2 presents an example created by our distributed IBRAVR prototype.

In general, the payload from the back-end and the viewer consists of “visualization data.” The texture maps and geometry in the current system combine on the viewer side to implement the IBRAVR algorithm. Arbitrary geometry can be used to represent the results of other types of visualization, such as the representation of grids in Boxlib [6], an Adaptive Mesh Refinement (AMR) multiresolution modeling framework. Figure 3 shows a set of

adaptive grids from a scientific simulation included with volume visualization.

The scene graph model plays an integral part in the design of the communication framework as well as viewer architecture: we can think of the scene graph model as a "data sink," and data arriving on a communication channel as a combination of scene layout and scene data, or content. Each of the viewer-side listener threads makes a contribution to the scene graph in the form of new texture data, or new geometry.



Our distributed IBRAVR prototype combines shared, parallel volume rendering with traditional visualization. In this case, the underlying grids are adaptive and hierarchical.

FIGURE 3. IBRAVR and Grid Visualization

3.4 Application of Shared Visualization

The prototype has proven useful in viewing large and time varying datasets produced by discipline scientists in the fields of Combustion and Cosmology, and was demonstrated at SC99 [9]. The prototype application defines a flexible framework centered around the communication protocol between the back-end and the viewer. As such, we have several types of back-end renderers. One of these back-end engines consumes data from a DPSS, a distributed parallel storage system.

The prototype shown at SC99 performed interactive rendering of a 50Gbyte time varying simulation, with data located in Berkeley, the back-end volume rendering engine located at Sandia National Laboratory in Livermore, and the viewer operating in Portland, Oregon.

A real and ongoing problem faced by scientific researchers is the sheer size of data produced by simulations and gathered by experiments. Datasets on the order of hundreds of gigabytes are not uncommon. Simply storing this much data can be problematic, and moving it across a WAN is often impractical. Our goal with Visapult is to make inroads into solutions for interactive visualization for problems of this scale. Three domains, data management, network and visualization technology all contribute to potential solutions.

4.0 Discussion and Future Work

We believe that network-based, shared rendering and visualization is a fruitful avenue for future research. The application model we have presented uses a decomposition that leverages current trends in technology: graphics, networking and data storage and management.

Low-cost graphics hardware for the PC continues to become faster and more usable. Current commodity-grade graphics accelerators match the rendering rates of $100,000 machines of just a few years ago. Those visualization tools that are cross-platform, and that perform well and scale through the continuum from the desktop to the fully immersive environment are economically and socially attractive.

Network technology improvements can enhance the basic "visualization dot com" model. Dynamic monitoring of Quality-of-Service parameters such as raw bandwidth, error rate, latency, reliability and priority can all have an impact on scheduling and performance of the system. Dynamic route discovery and modification could potentially result in shorter and more reliable data paths, either from the back-end to the viewer, or the data source to the back-end. Changes in bit rate can be taken into account to alter the resolution of data sent from the back-end to the viewer. Bandwidth reservation will assist in scheduling, so that "hero-

sized" problems may be smoothly executed. A "hero" problem would be one in which a researcher wishes to perform visualization of remote data that is tera-scale in size.

The prototype system discussed in this article is not unique in its design. MPEG-4, for example, provides for a scene description layer that is based upon scene graph technology, includes support for dynamic video compression, as well as support for audio [8]. One of the design goals of MPEG-4 is the possibility that the local viewer may interact with objects in a 3D scene, but with scene content being provided by a remote source.

Compression technology is integral to many network-based applications. The fundamental trade-off is one of time versus space. Compression algorithms can consume a substantial amount of time, but can produce highly compact and quickly-transmitted data objects. The cost of compression, which can be substantial for video streams, is typically amortized by many downloads of a single video or audio stream. Visualization tends to be an iterative process, hence the cost of video stream compression is difficult to justify.

An alternative, or supplement to payload compression is to approach the problem from a semantic, rather than syntactic perspective. We take this approach in Visapult by using "high level" descriptions of geometric elements when possible. The box geometries used in grid visualization are described with a minimum and maximum coordinate, rather than specifying eight box vertices and twelve box edges. Similarly, the quadmeshes in the IBRAVR implementation are specified with two coordinates, two integers defining the mesh resolution, then a stream of bytes defining offsets from the base plane for each grid point.

5.0 Conclusion

We have described a prototype application that explores a new approach to remote and large-scale visualization. Shared and parallel visualization and rendering lie at the center of the approach, with cooperative agents contributing to the finished product; interactive visualization on the desktop of remotely located data using remotely located resources, yet retaining the interactivity provided by desktop graphics engines.

While far from complete, Visapult provides glimpses of future research and development activities in parallel and remote visualization.

6.0 Acknowledgement

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

7.0 Bibliography

[1] Klaus Meuller, Naeem Shareef, Jian Huang and Roger Crawfis, IBR-Assisted Volume Rendering, Proceedings of IEEE Visualization 99, Late Breaking Hot Topics, pp. 5-8, 1999.

[2] Ulrich Neumann, Communication Costs for Parallel Volume-Rendering Algorithms, IEEE Computer Graphics and Applications, Volume 14, Number 4, pp 49-58, July 1994.

[3] The Message Passing Interface (MPI) Standard, http://www.mcs.anl.gov/mpi/.

[4] RM Scene Graph Programming Guide, http://www.r3vis.com/.

[5] David Butenhof, Programming with POSIX Threads, Addison-Wesley, 1997.

[6] C.A. Rendleman, V. E. Beckner, M. Lijewski, W.Y. Crutchfield, J. B. Bell, Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms, Computing and Visualization in Science, April 1999.

[7] Brian Tierney, Jason Lee, Brian Crowley, Mason Holding, J. Holding and F. Drake, A Network-Aware Distributed Storage Cache for Data Intensive Environments, Proceedings of IEEE High Performance Distributed Computing, August 1999. http://www-didc.lbl.gov/DPSS/.

[8] Overview of the MPEG-4 Standard, http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm#E40E1.

[9] SC99, Annual High Performance Networking and Computing Conference, http://www.sc99.org/