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 ship the image to the user. Another option, render-local, is to send data to the user, where it is visualized rendered on the local workstation. With advances in networking technology and graphics technology, we can begin to focus on 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 currently under development at Lawrence Berkeley National Laboratory (LBNL) that strikes such a balance, achieving a blended, scalable visualization tool.

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.

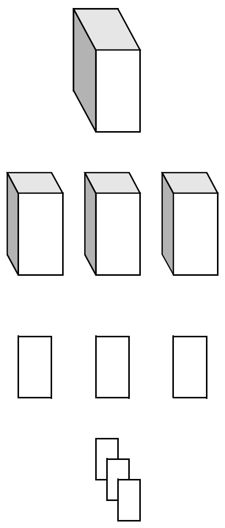
What if we could combine the best of both approaches? In such a model, we wouldn't have to move the potentially large data across the network, and we could take advantage of local workstation graphics. A blended model would facilitate the best use of resources; a large cluster, for example, could be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

3.0 A Prototype Implementation

The balance of this article describes a specific implementation of a distributed visualization prototype that is built upon a blended shared-rendering model. The title "Visualization Dot Com" reflects the importance of this type of emerging technology.

Our prototype is constructed from a distributed implementation of Meuller, et. al's Image Based Rendering Assisted Volume Rendering method [1], or just IBRAVR for short. The IBRAVR method leverages image-based rendering properties in order to achieve interactive volume rendering within a constrained interactivity framework. 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" (see 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.

3.1 Distributed IBRVR

The IBRAVR model maps nicely to an object-order decomposition for parallel rendering [2]. The primary difference between the IBRVR 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 partial images produced by each processor, each of which renders a subset of a volume, must be composited together in a particular order to produce a final image. Algorithms for image recombination in parallel software volume rendering have been the subject of much study [2].

The IBRAVR approach can be implemented with a pool of processors that perform object-order parallel software volume rendering. Rather than recombine the partial 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. The image warping 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.

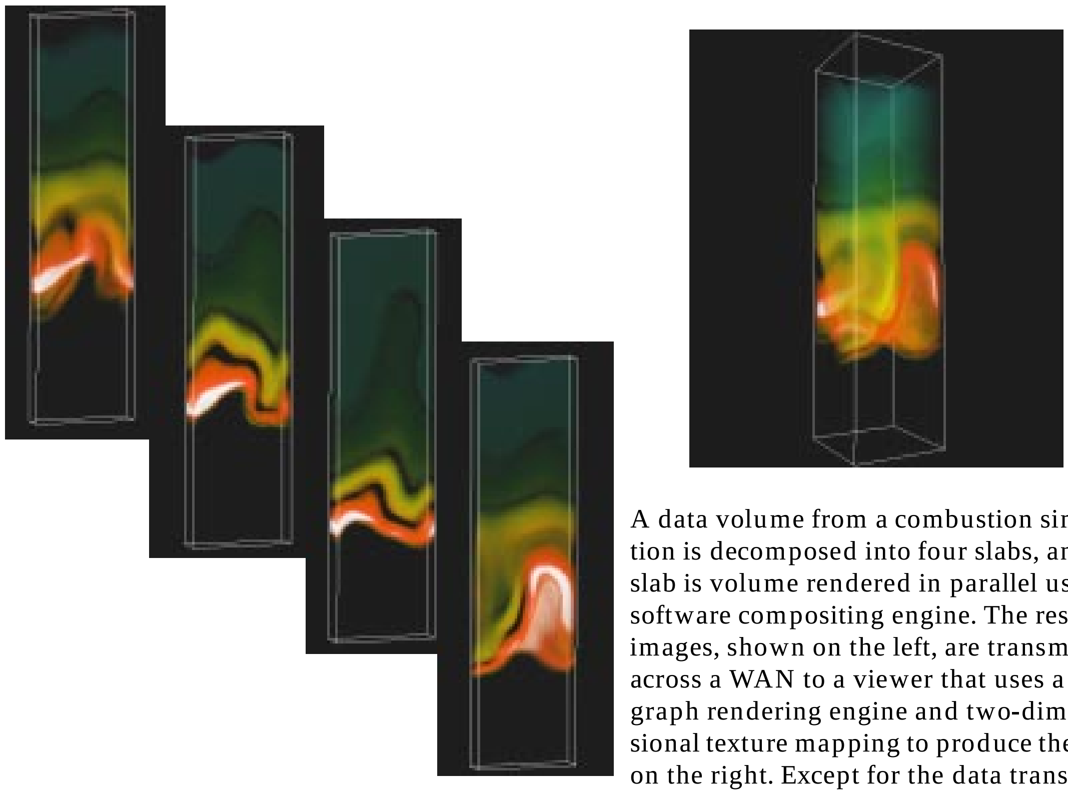
Our prototype is an application composed of two logical rendering components and one data component, all of which may be separated by a WAN. A "back-end" volume rendering engine performs the object-order parallel volume rendering in software. It is written using MPI [3], and runs on a variety of distributed 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. We'll address this component in more detail later in this article.

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].

The payload between viewer and software renderer consists primarily of the two-dimensional texture maps containing the results of partial volume rendering. Our present implementation is designed using 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 IRVAVR prototype.

In general, the payload from the back-end and the viewer consists of "visualization data." The texture maps and geoemtry in the current system combine on the viewer side to implement an IBRABR 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

The scene graph model plays an integral part in the design of the communication framework as well as viewer architecture: we 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.



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

3.2 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 in the LBNL and ASCI booths. 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 [7], 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. We used a private, high-speed network to move the large, time-varying data between Berkeley and Sandia-Livermore. Such demands may not be appropriate in congested, low-bandwidth networks.

Data logically travels in one direction only, from the data source (DPSS, Berkeley) to the parallel software volume rendering engine (SNL, Livermore) to the viewer (SC99, Portland). Control information, in contrast, travels in the "upstream" direction. For example, the back-end renderer needs viewing parameters from the viewer.

4.0 Discussion and Future Work

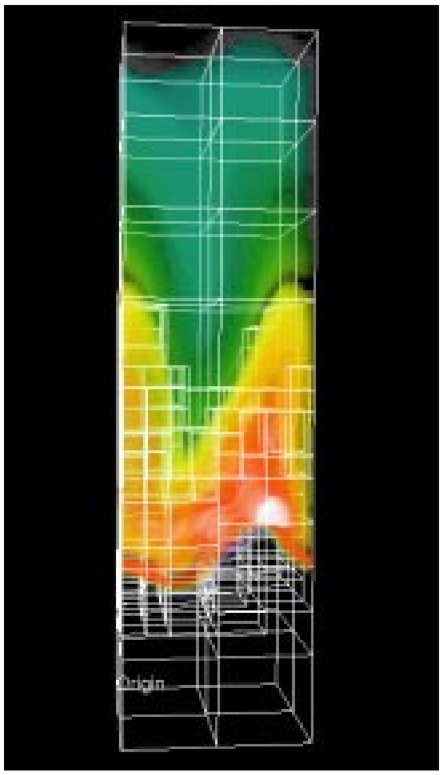
We believe that network-based, shared rendering and visualization is a fruitful avenue for future research. The application model we have presented uses a decomposition that leverages current trends in technology: graphics, networking and data storage and management.

Low-cost graphics hardware for the PC continues to become faster, and is presently matching the rendering rates of $100,000 machines of just a few years ago. Those visualization tools that are cross-platform and that scale from the lowly PC through the fully immersive environment are attractive from an economical standpoint.

Network technology improvements make possible a scenario that includes shared rendering. By network technology improvements, we mean not only evolution of the underlying grid fabric, but also advances in software and standards that make better use of the transport-centric shared visualization and rendering model.

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 still and dynamic video compression, as well as support for audio [8]. The benefits realized in MPEG-4 include 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 tradeoff 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. While reducing raw bandwidth through compression is a desireable goal, not all applications lend themselves to use of general compression. For example, when the content is static, but downloadable and viewable by many, the cost of compression is not only ammortized over the wide distribution, but may be performed once, off-line. Our prototype system is to be used for the visualization of large, time varying data that is remotely located. In the IBRAVR subsystem, the transmitted images are the results of visualization. Visualization is an iterative process - there is not a close match between the traditional compression ammortization model and a process of inspection, modification and parameter changing, followed by more inspection. However, general purpose image compression technology can reduce transmission costs in our system, at least for the IBRAVR subsystem.



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

FIGURE 3. IBRVR and Grid Visualization

Geometry compression is a related topic, but unlike image and audio compression, is still in embryonic stages of development. MPEG-4 refers to compression of geometry, where the geometry consists of meshes built from triangles, as well as a model for progressive refinement, or “level-of-detail” management, where the client may choose to display a lower-resolution model while a higher-resolution representation is downloading. Our approach is slightly different, where compression occurs, at least for the grid visualization, at the semantic level rather than at the level of vertices and faces. A “box” can be described simply with two vertices, and a sphere can be described with a center point and radius. The compression comes from using a higher-order representation of the underlying geometry.

Similarly, our “elevation maps” consist of two coordinates (minimum and maximum), two integers that define the resolution of the map in each parametric dimension, and a byte stream providing a positive or negative elevation at each implicit grid point of the elevation map.

Network technology improvments 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 from the back-end to the viewer. 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 combination of inexpensive, scalable storage with high capacity networking also promises to alleviate a long-standing requirement that the computing engine and data source are physically close. Bandwidth off the DPSS has been shown to be on the order of 100 megabytes per second. The bottlenecks we have encountered thus far tend to be clustered around the network interface of the parallel computer used for back-end data rendering.

In the future, we are interested in exploring two primary topic areas. The first is to expand the feature set of the underlying framework to support the needs of the scientific user community, the consumers of this technology. Second, we are tracking the changing network techologies in order to make best possible use of the network.

The use of the term “visualization dot com” is illustrative of an important point: with new technologies come new opportunities. We have discussed one such opportunity that capitalizes upon technology advances in networking and graphics.

5.0 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.

6.0 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 (beta), 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.