Visualization Viewpoints

Editors: Lloyd Treinish and Theresa-Marie Rhyne

Visualization Dot Com

Wes Bethel

Lawrence Berkeley National Laboratory

Here 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, called 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 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 call shared or "dot com" visualization, we stand to reap the best of both worlds—minimized data transfers and workstation-accelerated rendering.

This article describes Visapult, a prototype system 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.

Brute-force approaches

Consider the following common scenario: you and your workstation on the West Coast need to look at your data on the East Coast. What do you do? You could perform the visualization and rendering on the East Coast and send an image to your workstation. Or, you could move the data, all or a subset, to the West Coast.

In the render-remote approach, you win because only a single image crosses the network. Presumably, you could expect at least an order of magnitude reduction in traffic when sending only the final image compared to the cost of sending the raw data. The usability cost is the loss of interaction on the local workstation due to the sacrifice of local rendering capabilities. Here the workstation plays the role of a passive image display device.

To achieve interactivity using the remote rendering model requires a minimum of 10 frames per second, potentially using upwards of 30 megabytes per second of raw bandwidth ($1024 \times 1024 \times 24$-bit uncompressed images). We're making the generous assumption that, on the remote host, it's possible to perform visualization and rendering 10 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 power. Troublesome areas inherent in the render-local model include potentially long download times, the possibility that a large data set 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 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. For example, a large cluster performs computationally expensive parallel software volume rendering while the local workstation provides interactive graphics.

Visapult

Visapult, the LBNL prototype implementation, is a visualization tool that implements distributed, shared, and parallel visualization and rendering of large, time-varying, scientific data sets. The ongoing research and development focuses on three broad topics.

First, Visapult implements a framework that serves as 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.

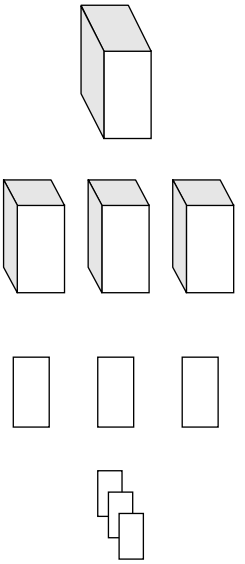
Second, Visapult provides a testbed suitable for use with emerging technologies that implement "network awareness."

Finally, Visapult helps address a vexing but commonly encountered problem—volume visualization of large, remotely located, time varying, adaptive, hierarchical scientific data that won't fit onto a workstation.

Distributed, parallel, and shared

The Visapult volume-rendering engine is a parallel, distributed implementation of Mueller et al.'s method, Image Based Rendering Assisted Volume Rendering[1]—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.

0272-1716/00/$10.00 © 2000 IEEE



1 IBRAVR task decomposition. (a) Volume data is first subdivided, with each processing element assigned volume rendering of its subset. (b) The results—an image from each processing element representing partial volume rendering—go to a viewer. (c) The viewer uses 2D texture mapping to render all partial images and provides for interactive transformation.

One of the many attractive properties of the IBRAVR model is that it performs well on low-end workstation graphics, or even software, but also runs 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.

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, used to create a "terrain-style" elevation map for each of the textures, provide more depth cues than flat quadrilaterals. The use of inexpensive 2D texture mapping enables interactive rotation with workstation-grade hardware.

The IBRAVR method works well, but within a limited range of rotation. Mueller et al. claim a rotation range of about 32 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.

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 2D texture mapping. By low cost I mean contemporary PC graphics cards in the $100 to $250 price range.

One of the fundamental ideas behind IBRAVR is that the image warping and depth-order compositing are performed using inexpensive graphics hardware. The image warping and interslice translation provided by texture mapping represent the image-based rendering aspect of the algorithm, while the depth-order rendering of semitransparent 2D textures represents the image-gather and compositing stage of the traditional object-order decomposition.

Visapult framework

Our prototype application consists of two logical rendering components and one data component, all of which may be separated by a wide-area network (WAN). A back-end volume-rendering engine performs the object-order parallel volume rendering in software. Written using MPI (the Message Passing Interface standard, http://www.mcs.anl.gov/mpi/), it runs on a variety of distributed memory and shared memory machines.

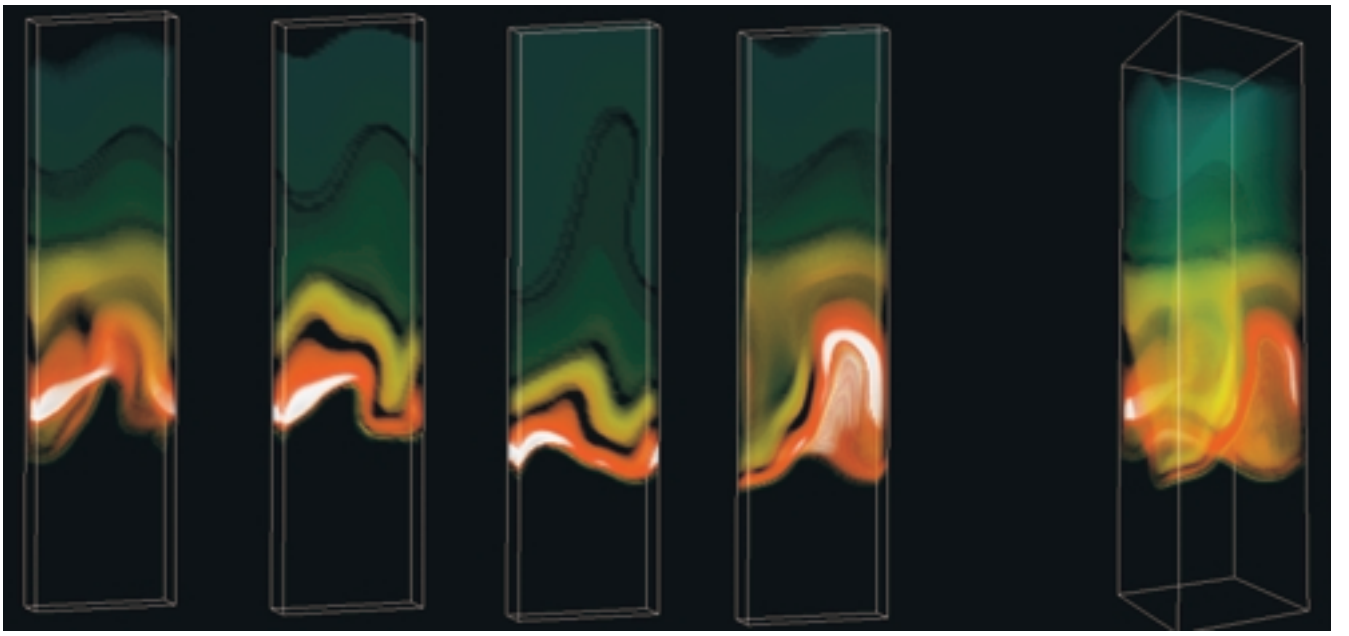
The second component—a viewer—is a lightweight, interactive rendering application built from an OpenGL-based scene graph tool (see the RM Scene Graph Programming Guide at http://www.r3vis.com/) that manages data and rendering services. The viewer is also a parallel application, built using Pthreads.[3]

The system's third component is the scientific data and its management. In some cases, this might be as simple as a large disk farm connected directly to the volume-rendering back-end. In other cases, the data may be scattered across a WAN using a network cache such as that implemented by a distriubuted parallel storage system (DPSS).[4]

The volume renderer and the viewer communicate over a custom interprocess communication (IPC) layer built using transmission control protocol (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 Moving Pictures Expert Group's MPEG-4 specification (http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm#E40E1).

For volume visualization, the payload between viewer and software renderer consists of 2D 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,[5] an Adaptive Mesh Refinement (AMR) multiresolution modeling framework. Figure 3



2 IBRAVR applied to combustion simulation results. 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 2D texture mapping to produce the image on the right. Except for the data transfer, both viewer and back-end rendering execute asynchronously.

shows a set of adaptive grids from a scientific simulation included with volume visualization.

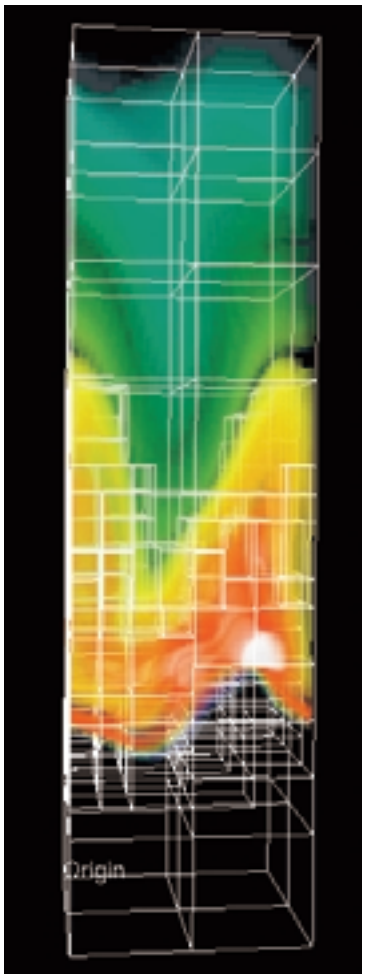
The scene graph model plays an integral part in the design of the communication framework as well as the viewer architecture. We can think of the scene graph model as a "data sink" and data arriving on a communication channel as a combination of scene layout and scene data, or content. Each of the viewer-side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

Application of shared visualization

The prototype has proven useful in viewing large and time-varying data sets produced by discipline scientists in the fields of combustion and cosmology. It was demonstrated at the Annual High-Performance Networking and Computing Conference, SC99 (http://www.sc99.org/). The prototype application defines a flexible framework centered on 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.

The prototype shown at SC99 performed interactive rendering of a 50-Gbyte time-varying simulation, with data located in Berkeley, California, the back-end volume rendering engine located at Sandia National Laboratory in Livermore, California, 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. Data sets on the order of hundreds of gigabytes are not uncommon. Simply storing this much data can be problematic, and



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

moving it across a WAN often proves impractical. Our goal with Visapult is to develop solutions for interactive visualization on this scale. Three domains—data management, networks, and visualization technology—all contribute to potential solutions.

Future work

I believe that network-based, shared rendering and visualization offers a fruitful avenue for future research. The application model 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 influence the system's scheduling and performance. 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 based on scene-graph technology, includes support for dynamic video compression, and supports audio (http://drogo.cselt.stet.it/mpeg/standards/mpeg-4/mpeg-4.htm#E40E1). One design goal of MPEG-4 is the possibility that the local viewer may interact with objects in a 3D scene, but with scene content 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 8 box vertices and 12 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.

Conclusion

The prototype application described here 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. Visipult provides interactive visualization on the desktop of remotely located data using remotely located resources, yet leverages the increasingly powerful desktop graphics engine to achieve interactive display rates.

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

Acknowledgment

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

References

1. K. Mueller et al., "IBR-Assisted Volume Rendering," *Proc. IEEE Visualization 99*, Late Breaking Hot Topics, ACM Press, New York, 1999, pp. 5-8.
2. U. Neumann, "Communication Costs for Parallel Volume-Rendering Algorithms," *IEEE Computer Graphics and Applications*, Vol. 14, No. 4, July 1994, pp. 49-58.
3. D. Butenhof, *Programming with Posix Threads*, Addison-Wesley, Reading, Mass., 1997.
4. B. Tierney et al., "A Network-Aware Distributed Storage Cache for Data Intensive Environments," *Proc. IEEE High-Performance Distributed Computing*, Aug. 1999, http://www-didc.lbl.gov/DPSS/.
5. C.A. Rendleman et al., "Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms," to appear in *Computing and Visualization in Science*, Vol. 3, No. 4, 2001.

Contact Bethel at Lawrence Berkeley National Laboratory, M/S 50F, University of California, Berkeley, Berkeley, CA 94720, e-mail ewbethel@lbl.gov.

Contact the department editors by e-mail: Treinish at lloydt@us.ibm.com and Rhyne at rhyne.theresa@epa.gov.