Deploying Web-Based Visual Exploration Tools on the Grid

T.J. Jankun-Kelly, Oliver Kreylos, Kwan-Liu Ma, Bernd Hamann, and Kenneth I. Joy
University of California, Davis

John Shalf and E. Wes Bethel
Lawrence Berkeley National Laboratory

Our Web-based portal lets users explore, encapsulate, and disseminate visualization results over the grid. This portal integrates an interface client, a visualization Web application, and a centralized application server.

The easy access to low-cost, high-performance, network-aware computers has impacted the way scientists conduct their research. While productivity has improved, scientists are burdened by the increasing size of the data they generate. Visualization is an effective and economical means to explore the data and the insights obtained through scientific study. However, due to the size of the generated data, the scientists, their data, and the visualization software are often located on different machines at geographically distributed locations.

Grid-based computing solves some of these problems by facilitating access to these different resources. But management in a grid-based environment isn't centralized. To use grid resources effectively, researchers need a central access point to manage the resources, provide a visual means to explore the data, and record these explorations for further investigation and dissemination. This article describes such a system that's being developed jointly by the University of California, Davis, and the Lawrence Berkeley National Laboratory (LBNL).

The centralized system acts as a portal into grid-enabled visualization systems. Scientists using the portal can focus on the important task of extracting insights from their data through visualization instead of having to worry about process management. Because scientists at LBNL and their collaborators require access to the portal from around the world, the portal's interface is entirely Web-based. Authenticated users only need a standards-compliant Web browser to explore their data from anywhere in the world.

The portal provides a Web-based interface not just for exploring but also for encapsulating visualization data. Encapsulating the process lets users reproduce the visualization results for validation or extend those results by continuing data exploration. In this article, we discuss the integration of our grid-enabled visualization server, the visualization Web application that performs the visualization session management, and the Web-based interface.

Grid-based portals

A portal is a single point of presence (typically hosted on the Web) that provides centralized access to widely distributed collections of information or services. The portal organizes this information to hide its complexity and location from the user. Portal technology also provides location-independent access to state information. When you log in to the URL of the portal's interface, you can access the same view of your personalized environment and data. Yahoo and Hotmail are typical consumer-oriented examples of this kind of capability. Grid portals extend the portal paradigm to organize and manage widely distributed computing resources, software components, and services that support collaboration among the people that form a virtual organization.

Many virtual collaboration and distributed application developers have turned to grid portals as the primary way to hide the complexity of distributed applications under a single interface. (See the sidebar "Related Web- and Grid-Based Visualization Work" for information about related research.) Consequently, several portal development toolkits have emerged, including the San Diego Supercomputing Center (SDSC) GridPort (http://gridport.npaci.edu/), the Grid Portal Development Kit (http://www.itg.lbl.gov/grid/projects/GPDK/), and GridSphere (http://www.ascportal.org/).[1]

Portal interfaces need not be Web-based, but Web portals have been widely adopted by the grid community, in part because of the ability to leverage the wide variety of robust development tools, components, and platforms that have already been developed for e-commerce servers. Also, given the ubiquitous availability of the Web and the comparatively uniform cross-platform programming model it offers for the client interface, the Web makes an attractive platform for a widely deployed

 Published by the IEEE Computer Society

Related Web- and Grid-Based Visualization Work

Research on controlling visualizations using the grid typically falls into two areas: Web-based visualization systems and distributed visualization in grid-like environments. Although there has been limited interaction between these fields, most previous research has focused on only one or the other. Ang and colleagues[1] described one of the first Web-based visualization systems. Ang's system displays visualization results within a Web page using an embedded application (an applet). The results are controlled using a launched application on the client side. This launched application communicates with the visualization server to request rendering. The visualization server then communicates with the applet to display the result.

Wood and colleagues[2] generalized Ang's approach into four different compositions of Web-based visualization. In the first scenario, the system sends only images to the client with no user interaction with the visualization. The second scenario lets a user manipulate the result, but a user can't change the visualization parameters. The next scenario supports full control of the visualization, including the type of visualization performed, but requires significant resources on the client side. The last scenario, and the one implemented by both Wood and Ang, supports Web-based interaction for controlling the parameters and the visualization type performed without requiring a significant client installation. The majority of subsequent Web-based visualizations follow this last approach, using an applet to allow interaction with the visualization. Our Web interface combines aspects of the first and last approach.

W. Lefer[3] and C. Bajaj and S. Cutchin[4] implemented a Web-based interface to distributed or grid-based visualization systems. Lefer's visualization system dynamically and transparently shares the processing load on a local area network. Another interesting property of Lefer's approach is that interaction with the visualization system happens entirely through HTML-based forms. We use an all-HTML approach as well, but augment it with JavaScript. This approach eliminates the need for HTML forms by letting the JavaScript events invoke actions on the visualization server. Bajaj's work also coordinates the users, distributed resources, and the use of those resources. Our Web interface doesn't manage the grid resources; the visualization portal as a whole handles this task.

Researchers have also investigated visualization using grid resources that don't involve Web-based interfaces. GridMapper[5] addresses the problem of determining the performance of grid computations by collating and visualizing distributed information. This information is dynamically gathered from the sites performing the computation on the grid. TeraVision[6] lets users seamlessly present and interact with graphics, such as visualizations over the AccessGrid. Finally, Cactus[7] is a grid-based, computational astrophysics framework that incorporates various visualization methods: a Web-based slice viewer of the simulation volumes created at each node, a remote isosurfacer (with the isosurface calculated locally at each compute source and rendered elsewhere), and Visapult, an image-based volume renderer.

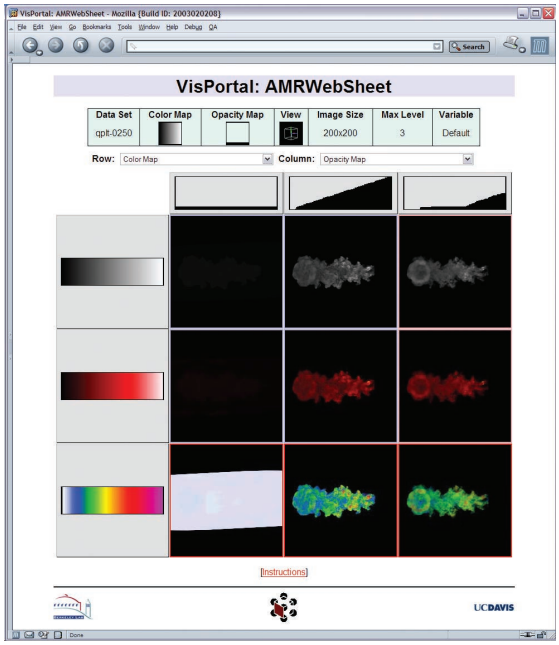
Each of these approaches (except the Web interface for Cactus) requires a thick-client installation to perform the visualization. In other words, for remote users, their platform and system capabilities must be determined before the appropriate grid visualization client can be installed and launched. Our approach eliminates the need for a thick client by requiring only a standard Web browser.

References

1. C.S. Ang, D.C. Martin, and M.D. Doyle, "Integrated Control of Distributed Volume Visualization Through the World Wide Web," *Proc. IEEE Conf. Visualization 1994*, IEEE CS Press, Oct. 1994, pp 13-20.
2. J. Wood, K. Brodlie, and H. Wright, "Visualization over the World Wide Web and Its Application to Environmental Data," *Proc. IEEE Conf. Visualization 1996*, IEEE CS Press, 1996, pp. 81-86.
3. W. Lefer, "A Distributed Architecture for a Web-Based Visualization Service," *Proc. Ninth Eurographics Workshop on Visualization in Scientific Computing*, Springer-Verlag, 1998.
4. C. Bajaj and S. Cutchin, "Web-Based Collaborative Visualization of Distributed and Parallel Simulation," *Proc. 1999 IEEE Parallel Visualization and Graphics Symposium*, IEEE CS Press, 1999, pp. 47-54.
5. W. Allcock et al., "GridMapper: A Tool for Visualizing the Behavior of Large-Scale Distributed Systems," *Proc. High Performance Distributed Computing*. IEEE CS Press, 2002, pp. 179-87.
6. J. Leigh et al., "TeraVision: A Platform and Software-Independent Solution for Real-Time Display Distribution in Advanced Collaborative Environments," *Proc. Access Grid Retreat*; http://www-fp.mcs.anl.gov/fl/accessgrid/agretreat2002/Proc./leightera.pdf.
7. G. Allen et al., "Solving Einstein's Equations on Supercomputers," *Computer*, vol. 32, no. 12, Dec. 1999, pp. 52-58.

client interface to grid services. Furthermore, the Web requires essentially no custom software installation for the scientists who use the service.

We regard the spreadsheet approach as an excellent match for the Web's interaction modality. Spreadsheets can capitalize on the HTML table display paradigm as well as the use of hyperlinks to drill down into information content. We can easily integrate such an interface



1 The AMRWebSheet interface, an example of our Web interface to grid-based visualizations. The interface consists of three major areas: the default parameter bar that displays and allows the modification of the default parameter values; the displayed row and column parameter drop-down lists; and the tabular result display. The first two components can change the location of the tabular window in visualization parameter space, while the last component can request new results.

with other ongoing portal development efforts throughout the grid community.

Web-based visualization

The system we're developing for Web-based visual data exploration over the grid consists of three major components:

- a Web-based user interface to grid-enabled visualization services,
- a visualization Web application that tracks the exploration of visualization results, and
- the portal application server that manages and coordinates the authentication for and use of grid resources.

The application server (the VisPortal) uses established grid technologies to handle user and resource management. Once the system authenticates the user, the Web application initializes a new visualization exploration session.

The Web application (also called a servlet) is a program on the Web server that communicates with the client via HTTP. The servlet maintains the visualization session state. After the application initializes the visualization session, the Web-based visualization interface loads in the client's Web browser. As the visualization session progresses, the Web application stores the visualization results and the relationships between those results. Finally, when the user finishes visualizing the data, the application closes the session. The user can then initialize another session or reexamine previous explorations.

Sheet-like interface

Our Web interface (the WebSheet) implements an entirely Web-based version of the visualization exploration sheet-like interface discussed elsewhere.[2] The original spreadsheet-like interface (the VisSheet) was designed to assist visualization exploration by providing context for where users are, where they have been, and where they might go. The VisSheet handles these tasks by providing a movable, scalable window into the visualization parameter space. The application displays only two visualization parameters at a time: one along the rows and another along the columns. For the nondisplayed parameters, the application maintains a set of default values that can be updated at runtime. The application renders parameter values as glyphs. Cells representing a combination of the row, column, and default parameter values display the visualization results. By changing the default values for nondisplayed parameters, users can move the window in the visualization space. Thus, the data exploration process becomes the process of manipulating the spreadsheet window through visualization space.

Our sheet-like Web interface shares many characteristics with the VisSheet. The interface refines the initial VisSheet design to let a user easily modify default and displayed parameters through the default parameter bar and drop-down row and column parameter lists (Figure 1). The default bar assists in identifying parameter values and their corresponding results. The parameters are always the parameters belonging to a cell's row and column, combined with the default values for the other parameters in the default bar.

Interaction with the tabular display remains essentially unchanged. Users can add, edit, or remove parameter values; render or view a cell's image; and apply parameter and value operators to generate new rows, columns, or cells. The implementations of VisSheet and WebSheet, however, differ significantly. Design considerations for our Web interface required several modifications to the original VisSheet. Due to the wide range of platforms scientists can use to access the Web interface, we wanted to create a platform-independent solution. However, the software environment of each user is unlikely to be the same, and the difficulty in remotely installing or the unavailability of plug-ins for certain environments meant that we couldn't use Macromedia's Flash or Sun's Java.

For example, the incompatibilities of Java on different platforms or between different versions make it difficult to use in a robust client setting. The only assumption we made was that a user possesses a standards-compliant Web browser with ECMAScript/JavaScript and cookie support. We designed the system so that no permanent state would be stored on the client machine. By keeping the state in a centralized, Web-accessible location, users can reexamine visualization sessions from different locations without losing any

Visualization Exploration Process Model

The visualization process for information and scientific visualization is an iterative sequence of user-applied transformations from data to view.[1-3] The fundamental operation that occurs during the visualization process is the formation of parameter value sets to derive visualization results. These parameter value sets (p-sets) possess a parameter value for each parameter in a visualization transform—the function that performs the mapping of data to visual primitives. When applied to a visualization transform, a p-set corresponds to a rendered result. Other research describes a model of the visualization process based on a parameter derivation calculus.[4] The calculus defines how p-sets and the results rendered from them are derived from previous p-sets.

User interaction with the visualization system creates new p-sets in one of three ways:

- *Parameter application.* Parameter values from a p-set are applied by the user to another p-set to generate a new p-set. For example, a new color map replaces an old color map in a previously generated p-set to render a new result from the new p-set.
- *Parameter range sweep.* Users can interactively manipulate a single parameter value over a range between an initial and final p-set. For example, users can generate a range of view positions by dragging a mouse pointer in the render window.
- *Function parameter generation.* A function/operator generates a set of parameter values to be used in a p-set. For example, applying a set union operator to two previously used opacity maps creates a new opacity map.

In the AMRWebSheet, we use only two types of parameter derivations: parameter application and function parameter generation. When the system renders a cell, it collects the parameters for that cell in a p-set; this process corresponds to a parameter application of the new parameter values to the p-set from the last generated result. Function parameter generation occurs when a user applies an operator in the AMRWebSheet to generate new parameter values.

The parameter derivation calculus is the basis for recording a visualization exploration session. Formally, a visualization session consists of a set of visualization session results. A visualization session result contains a p-set, the visualization result derived from the p-set, a time stamp to place the result in temporal context, and a parameter derivation calculus instance detailing how the result was derived. Each session result represents the generation of a single visualization result.

However, as more than one result can be generated in a single user action, such as when applying a parameter operator, multiple session results can share the same time stamp. For each visualization result (an image), the AMRWebSheet stores its corresponding visualization session result (information about that image). This approach differs from previous Web-based collaborative visualization work[5] where only parameters for particular visualizations and their position in a tree of parameter snapshots are stored for future collaboration. In our system, the application stores the complete exploration and derivation information (encapsulated in session results) as an XML document on the portal for later access and reexploration.

References

1. S.K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in Information Visualization: Using Vision to Think*, Morgan Kaufmann Publishers, 1999.
2. C. Upson et al., "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, July 1989, pp. 30-42.
3. R.B. Haber and D.A. McNabb, "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems," *Visualization in Scientific Computing*, G. Nielson and B. Shriver, eds., IEEE CS Press, 1990, pp. 74-93.
4. T.J. Jankun-Kelly, K.-L. Ma, and M. Gertz, "A Model for the Visualization Exploration Process," *Proc. IEEE Conf. Visualization 2002*, IEEE CS Press, 2002, pp. 323-330.
5. K. Brodlie, S. Lovegrove, and J. Wood, "Harnessing the Web for Scientific Visualization," *Computer Graphics*, vol. 34, no. 1, Feb. 2000, pp. 10-12.

information. Because of these constraints, our interface is Web-based and not a single-user, network-unaware Java application like the VisSheet.

Several consequences, however, result from our Web interface design. Interactivity partially suffers in comparison to the VisSheet. The VisSheet uses the Java Foundation Classes (JFC) for its user interface. JFC supports a rich set of user-interface elements and customizability. In contrast, the interface elements offered by HTML are limited. HTML supports only checkboxes, radio buttons, push buttons, lists, menus, and text fields. JavaScript can help overcome many of these limitations by letting different portions of the HTML page react to mouse events. Even so, HTML can't render glyph icons, for example, that represent the parameter values in a drop-down list for the default parameter bar.

Furthermore, JFC provides more functionality than raw JavaScript. Our Web interface can't query any significant information about a client's machine. Methods exist to extract the client's browser information or to download a single file from the client machine, but such transfers aren't optimized for the large data sizes common in visualization applications.

Web-based encapsulation

The Web-based visualization interface structures the visualization-exploration process, which the application server captures. By capturing the process, we can ensure that the visualization results generated, and the relationships between those results, aren't lost when the visualization session ends. To record the visualization process, we use a formal model of the visualization exploration process (see the "Visualization Exploration Model" sidebar). As the application renders each

The Web interface and visualization Web application can help with several kinds of scientific visualization problems.

requested image, the server stores the corresponding visualization session result. Thus, at the end of a session, all the rendered images—including the parameter value sets (p-sets) used for creating those images—the time the image was generated, and that image's relation to previous images, are available for later use.

The visualization Web application is the entry point to our Web interface. When loaded from the portal, the servlet provides a user with two options: starting a new visualization session or viewing previous sessions. When the user chooses to start a new session, another servlet, the user-interface servlet, handles interactions with the visualization interface. As the user requests images or adds, edits, or removes parameter values, the underlying JavaScript sends HTTP requests to this servlet. The servlet then processes the requests, contacting the rendering server if needed, and updates the visualization session and the client. The interface servlet represents the interface's state. If a user chooses to examine previous sessions, the application loads the session servlet.

Initially, the application presents a list of all the previous explorations, sorted by date, to the user. A user can reload a previous session in the Web interface by clicking on its corresponding link. Users can add new results to this session. When the session terminates, the application will store these results along with the old session information. This capability is crucial to the VisPortal because scientists must be able to distribute their work over time as well as over space.

The session servlet also supports the ability to view previous sessions. By selecting the "view as HTML" option, the user initiates the generation of an HTML page that summarizes the corresponding visualization session. Each result, the parameters corresponding to that result, and the parent and child results are all part of the HTML page. The HTML page serves as an overview of a previous visualization session and as documentation of that session. First, the Web page fully documents the visualization process as it completely describes the information captured by the visualization process model. Second, users can add or edit annotations of results. The application stores these annotations on the server for others to access. Scientists can use these annotations to flag certain results as interesting.

The session servlet also lets users view an overview graph. While the HTML session document describes the visualization in detail, it's difficult to obtain a sense of the visualization at a glance. By selecting the "view overview graph" option from the session list, the servlet generates a graph depicting the results and various relationships between the results. The user chooses a "visualization metric" that determines how the application displays the graph.

All the graphs use a new radial focus-context visualization technique, in which the radial distance from the center node to another node represents the distance of that node's p-set from the center result's p-set according to the chosen metric. As the distance increases, the size of the node and its radial separation decreases to let the system display all results simultaneously. Example metrics include those that measure how a result was derived from another result (a directed edge only exists if the first result derives the second) and those that measure the temporal distance of the result (a directed edge only exists if the first result was rendered immediately before the second result). Different metrics and the HTML session view provide the means to understand what occurred during a visualization session.

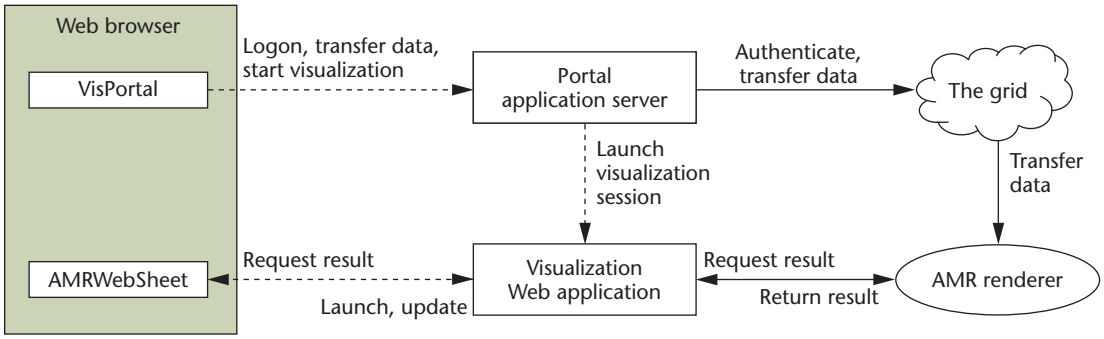
Application domain

The Web interface and visualization Web application can help with several kinds of scientific visualization problems. Scientists at LBNL are particularly interested in visualizing Adaptive Mesh Refinement (AMR) data. Many of the most challenging problems in numerical modeling involve meshes with huge ratios of scale. For example, when modeling the fuel-injection system of an automobile, you must model the fluid dynamics of the injector's orifice as well as the dynamics of fuel–air mixing in a cylinder chamber. A typical finite-difference or finite-element simulation covers the entire domain with a uniform mesh of cells, the smallest of which must be less than half the size of the smallest structure being modeled.[1] Given the huge ratio of scale in these structures, it would be impossible to span this range of spatial scales without using impractically large meshes.

AMR techniques for finite-difference codes use refinement criteria that create higher-resolution meshes only in areas in which they're needed. For example, large-scale structures of superclusters in cosmology are relatively compact; it would be a waste to model uninteresting events in the voids using the same mesh resolution as that used for the events that occur in the dense regions. With current methods, the refined meshes must be an even multiple of the size of the parent meshes on which they're placed. Furthermore, regions can be refined in a recursive process that can descend through many levels of resolution.[1]

For second- or higher-order methods, the cell size must be even smaller. The cosmology simulations done by Norman, Bryan, and Abel[3] that modeled the formation of the first stars in the universe require 27 levels of refinement, covering an eight-billion-to-one ratio of scale using a fraction of the memory required for a uniform mesh. Starting with a 128^3 base mesh for the AMR simulation, an equivalent simulation on a uniform mesh would require at least a 10^{36} element uniform mesh. AMR makes these extreme problems tractable for today's supercomputers. These problems also pose significant challenges for visualization researchers.

AMR data structures don't fit into any of the traditional data structures used in modern visualization techniques and systems.[4] Sampling AMR data onto uniform meshes results in the same data-handling problems that motivated the development of AMR in the first place.



2 The VisPortal/AMRWebSheet architecture. Elements within the blue box represent Web pages with which the user interacts; green boxes represent Web servers; and the other nodes represent grid-accessible resources. Dashed lines represent HTTP/HTTPS connections for HTML and images (for the client) and connection requests (for the server), while solid lines represent TCP connections. A line's label indicates the action performed over that link. For bidirectional connections, the top label corresponds to actions initiated from the left entity, while the bottom label corresponds to actions initiated from the right entity. The AMR renderer can reside anywhere on the grid. The VisPortal initializes the connection between the renderer and the visualization Web application when a user first requests a visualization.

Naive conversion of AMR data to finite-element data structures composed of hexahedral cells requires us to use memory-inefficient data structures with comparatively inefficient visualization algorithms. Furthermore, dangling nodes at the coarse–fine mesh boundaries can cause cracks. Direct application of finite-difference techniques to an AMR hierarchy leads to visual artifacts at the coarse-fine boundaries as well as to significant data-management issues. A typical desktop system can't process these deep hierarchies interactively. Consequently, few visualization algorithms can be directly applied to hierarchical meshes, and essentially no off-the-shelf commercial software is available for visualizing AMR data. It's important to develop the tools and techniques necessary to navigate data sets with huge ratios of scale in a simple and widely accessible manner.

Given the growing interest in AMR simulation and the need for scalable systems supporting remote data visualization, we developed a parallel multiprocessor volume renderer for AMR data. (For more information about this, see the "AMR Volume Rendering" sidebar on the next page.) Because the majority of LBNL visualization users are off-site and have comparatively smaller resources at their disposal, we created a client–server architecture so the entire system can be accessible over the grid using a traditional client interface. The AMRWebSheet project—an implementation of the WebSheet—aims to extend access to this parallel-rendering back end using an entirely Web-based interface suitable for emerging Web-based collaborations.

Architecture

Figure 2 summarizes the VisPortal/AMRWebSheet architecture. We implemented the AMRWebSheet interface and Web application in a flexible visualization exploration and encapsulation framework. The framework, implemented in Python, consists of a series of objects that manage visualization sessions and the interface's interactions with the sessions. Visualization session, transform, parameter, result, and derivation objects exist within the framework to capture the information described in the visualization exploration process model.

A view object represents the interactions between visualization interfaces and the visualization sessions in a platform-independent manner. Other interface objects representing general visualization exploration spreadsheet views and state also exist within the framework. The system uses these classes as the basis for different implementations of the VisSheet. One implementation recreates the original VisSheet as a Java application using Jython (the Java version of Python) to communicate between the framework and the Java classes. The servlet uses the framework to implement the AMRWebSheet.

We implemented the Web application servlets that manage visualization sessions in Python using the Webware application environment (http://webware.sourceforge.net/). We use Apache running under Linux to serve Web pages. A group of servlets creates, processes, and stores sessions. When a client connects, the application creates a new session identified by a temporary cookie in addition to servlet-persistent objects. Whenever a user interacts with the generated HTML interface, the application communicates the AMRWebSheet HTTP requests to the interface servlet. This request in turn modifies the visualization session state. When the client needs to update, the server performs a refresh to display the new information. When a session terminates or expires due to inactivity, the application encodes the session results as an XML document on the application server for later retrieval.

The application server handles all communication between the AMR volume renderer and the AMRWebSheet. When the AMRWebSheet requests a result, a visualization transformation class instance on the Web application requests the corresponding result from the volume-rendering server. When the volume renderer returns the corresponding result, the Web application stores a copy with the visualization session results. The

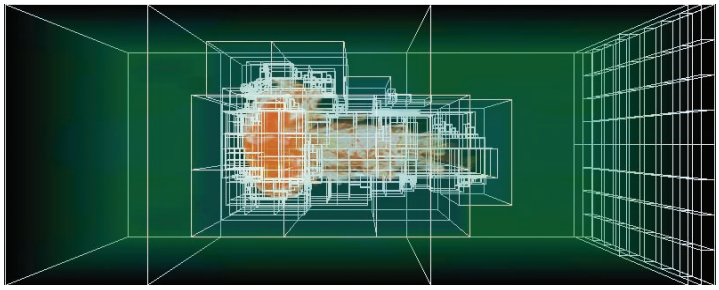
AMR Volume Rendering

The current rendering back end for AMR data is a hardware-assisted 3D texture-based parallel volume renderer.[1] AMR hierarchies are typically highly irregular and can't be rendered directly using graphics hardware's texture-mapping capabilities (Figure A). Instead, a given AMR hierarchy must be homogenized, which means that it has to be transformed into a set of nonoverlapping rectangular grids with acyclic visibility order for any viewing direction. This process generally involves removing parts of lower-resolution grids overlayed by higher-resolution grids, and then splitting the resulting nonconvex grid regions into rectangular grid patches.

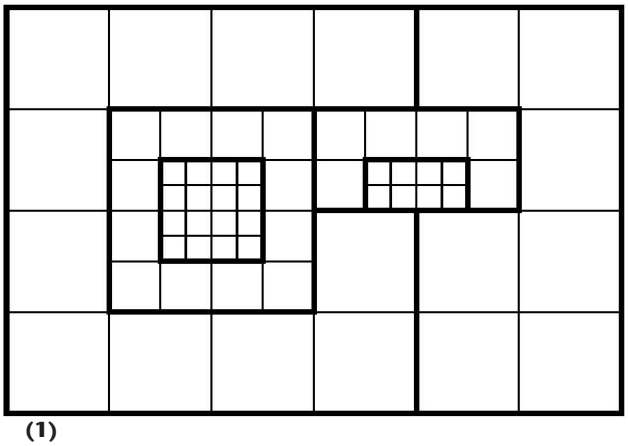
Our method uses a tree-based approach, in a *k*-d tree covering the entire domain of an AMR data set. AMR grids are inserted one at a time, starting with lowest-resolution grids. Figure B shows a 2D AMR hierarchy and its homogenizing *k*-d tree. Once an AMR hierarchy is homogenized, it can be rendered from arbitrary viewpoints by sorting all grid patches on-the-fly in back-to-front visibility order. All grid patches are rendered independently into the same color buffer using alpha blending, performing implicit compositing of partial-rendering results.

For parallel rendering on *n* nodes, the sorted list of grid patches is chopped into *n* sequences of approximately equal rendering cost. We estimate rendering cost during *k*-d tree traversal. The sequences are then assigned to rendering nodes. This step is performed on all nodes in parallel and doesn't require communication between nodes. Each node renders its sequence of patches into its own color buffer. When rendering is done, nodes exchange color buffers to composite a complete rendering. Because the rendering bottleneck is grid-patch rendering, and compositing is performed in hardware, a simple binary tree compositing strategy suffices. It could be replaced with a binary-swap compositing strategy should the need arise, but the rendering algorithm is independent from the choice of compositing strategy.

The portal version of the parallel renderer currently runs on an SGI Onyx3400 with two IR4 graphics pipes. A software parallel renderer can also be used on cluster or distributed-memory architectures. At the University of California, Davis, the renderer runs on two Linux clusters (with 4 and 16 nodes, respectively) using NVidia GeForce3 graphics cards for rendering and 100 BaseT Ethernet for internode communication. You can read more information about this setup elsewhere.[1]



A Volume rendering of the argon bubble with superimposed AMR grid hierarchy.



B Homogenizing a 2D AMR hierarchy. The hierarchy has a uniform refinement ratio of two. Bold lines denote grid boundaries. All hierarchy levels consist of two grids. Finer grids can cross boundaries between coarser grids. (1) Original AMR hierarchy with overlapping grids. (2) Homogenized hierarchy with nonoverlapping rectangular grid patches.

Reference

1. O. Kreylos et al., *Remote Interactive Direct Volume Rendering of AMR Data*, tech. report LBNL49954, Lawrence Berkeley National Laboratory, 2002.

application then displays the result by forcing a refresh on the client's Web browser.

The VisPortal handles access to the visualization interface. It provides a single point of access to launch and control all the components of this distributed tool. We based the VisPortal's architecture on the Grid Portal Development Kit that uses the Globus Project's Java Commodity Grid (CoG) toolkit (http://www.globus.org/cog/java/) in conjunction with Tomcat, an Open Source Java Server Pages application server (http://jakarta.apache.org/tomcat/). Users authenticate to the portal using the MyProxyServer (http://www.ncsa.uiuc.edu/Divisions/ACES/MyProxy/) to supply their X.509-delegated credentials. The Grid Security Infrastructure (GSI) X.509 credentials make it possible for the portal application server to transfer files, launch jobs, and otherwise access any Globus grid services on remote hosts using only a single login.

From the user's standpoint, the portal hides a complex application-launching mechanism for a multicomponent distributed application. For a thick-client application, the portal launches a parallel-computing component and brokers a direct socket connection between this computing component and a high-performance back-end data source. It then launches the thick client through the Web browser using appropriate multipurpose Internet mail extensions-type definitions. The thick client in turn connects back to the remotely located parallel-visualization component, thereby completing the distributed visualization application. The system hides this entire elaborate launching procedure from the user. The user simply selects remotely located data and presses a button to start the visualization application.

The AMRWebSheet supports an even simpler launching mechanism where the back end connects directly to the Python visualization Web application. The server makes requests of the back end and then formats the output images appropriately for the HTML interface. If the back end is located on a Silicon Graphics machine, it can employ hardware-assisted off-screen rendering using Infinite Reality Engine pipes. If the back-end host is a cluster or distributed-memory computing architecture, it can employ the parallel software rendering back end. Again, the portal client interface hides the complexity from the user.

The AMRWebSheet's performance depends on the performance of the

- AMR volume renderer,
- visualization application server, and
- user's Web browser.

Variations in network traffic between the renderer, application server, and client can also affect performance. Table 1 provides performance measures for the elements under our control: the volume renderer and the application server. For this test, we used an AMR data set consisting of 501 time steps and 640 x 256 x 256 cells at the finest level. We measured performance in seconds for three different hierarchy levels (0 being the coarsest and 2 being the finest) using an 800 x 600 pixel image as our output. We collected two sets of performance data: one for the initial loading of the data set (which requires the hardware textures on the SGI Onyx3400 renderer to be initialized) and one for preloaded data (and pregenerated textures).

As the table shows, the rendering time depends on the maximum level rendered, while the application server processing time is nearly constant. We expected this result because the application server only processes completed images, which don't depend significantly on the rendered hierarchy level. Processing on the application server only occurs once when a result is initially rendered. Subsequent requests for the same image (such as when the HTML page is refreshed to add new images) are either cached by the Web browser (which doesn't require retransmission) or cached by the application server (which requires retransmission over the network, but no further rendering).

Usage scenario

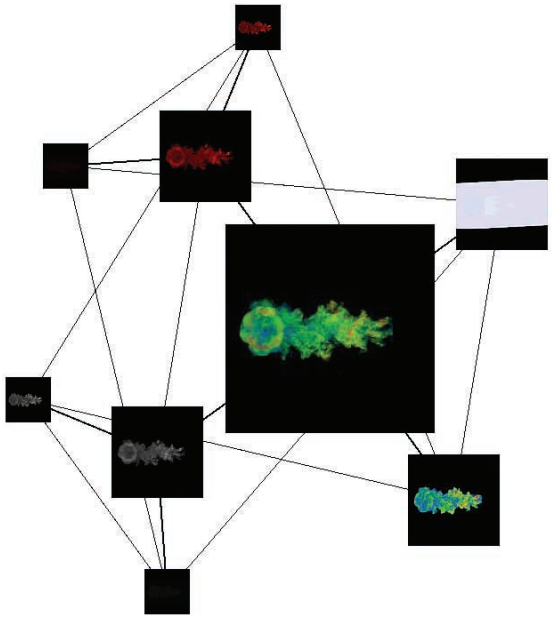
To demonstrate the VisPortal/AMRWebSheet concepts, we'll describe a typical scenario. In this scenario, a scientist at LBNL named Alice decides to visualize results from a shock-refraction and mixing computational fluid dynamics (CFD) simulation. The data set shows the time evolution of an argon bubble after being disturbed by a shock wave. The bubble moves steadily from one side of the volume used for the simulation to the other, while deforming. The user is interested in a particular time step in the later stages of the simulation. The data set is located on the LBNL intranet and is accessible over the grid.

Alice first enters the VisPortal URL into her Web browser. After logging onto the system, she uses the portal's access to the grid to transfer the argon bubble data set from its original location to the AMR volume renderer server. Because Alice's virtual organization lets her access the AMR renderer via the grid, the system authenticates the transfer. The system hides the complexities of the transfer (such as using GridFTP) from Alice. She then requests a new visualization session from the portal. Again, the system verifies Alice's credentials, this time confirming that she can access the visualization service—all the authentication occurs behind the scenes.

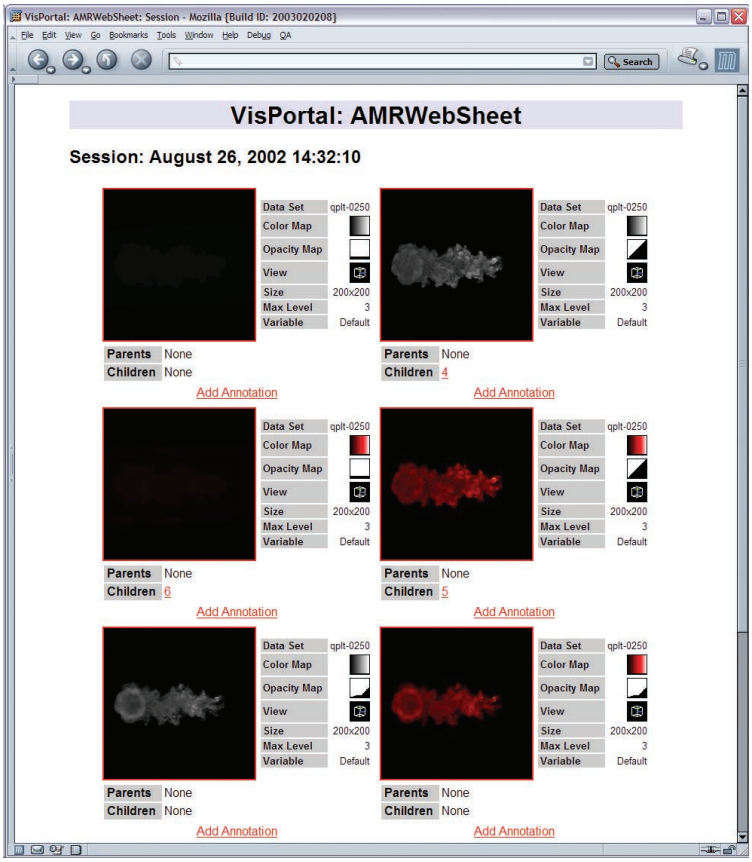
Once the verification is complete, the portal transfers the authentication to the visualization Web application. Upon initialization, the Web application determines whether Alice wants to start a new visualization or view

Table 1. Performance measurements of the AMR volume renderer and the AMRWebSheet application server.

	Initial Data Loaded		Preloaded Data	
Levels	Renderer (seconds)	Application Server (seconds)	Renderer (seconds)	Application Server (seconds)
0	0.29	0.35	0.30	0.31
1	0.32	0.36	0.30	0.32
2	1.4	0.37	0.53	0.30



3 Parameter session graph for the session shown in Figure 1. The edges indicate that only one parameter value differs between the two resulting images. Session graphs provide an overview of different information about the visualization session.



4 HTML session page for the session shown in Figure 1. The page provides a summary of the visualization session and supports the annotation of results.

or expand an older session. In this scenario, she starts a new visualization session. After specifying an initial data set, the AMRWebSheet loads in Alice's browser. She then explores the data via the Web page interface until she is satisfied with the results (Figure 1). At this point, Alice terminates the visualization session and exits the portal. When she exits, the system automatically records the visualization session.

At some later date, Alice's colleague, named Bob, wishes to verify the results generated during the visualization. Bob is also part of Alice's virtual organization. Like Alice, Bob logs on to the VisPortal. Unlike Alice, Bob wants to view a previous visualization session instead of a new one. The visualization Web application presents Bob with a list of sessions from which he can choose. Bob first chooses to examine an overview graph of the visualization session (Figure 3). After familiarizing himself with the visualization results, Bob loads the HTML session document (Figure 4). Bob then annotates a few results of interest and exits the system. As with the original session, the visualization Web application stores Bob's annotations automatically when he exits. Later, Alice can reload the session, view Bob's comments, and perhaps add some comments of her own. The portal lets these scientists focus on using their data rather than managing it.

Future work

The work described here represents one aspect of the VisPortal project. We're actively developing three core areas: the underlying portal application server, the visual-exploration tools, and the session-management capabilities. For the application server, our current work focuses on improving its low-level implementation and the connection between the Grid Portal Development Kit and the various CoGs. This work includes adding support for a Python CoG for easier integration of the visualization exploration and encapsulation framework with the grid. Finally, we're also working on integrating a database management system (DBMS) with the application server. Once complete, authentication, session management, and resource allocation will use the DBMS to record portal-wide usage behavior.

The AMRWebSheet is only one of several visualization interfaces planned for the VisPortal. Visapult, a visualization system that uses both client and server resources to perform interactive visualization (see the article about Cactus and Visapult elsewhere in this issue) has already been integrated with an earlier version of the portal. Researchers are also investigating alternate Web-based VisSheet implementations. For example, a VisSheet-like interface for visualizations using Kitware's Visualization Toolkit would vastly increase the potential number of visualization applications used by scientists interacting with the portal.

Additionally, we're interested in using more grid resources for the visualization. The interface should allow access to visualization resources, numeric and statistical analysis codes, and other related services. The grid would then transparently manage the resource discovery, process allocation, and data transport between these services. The interface could be adapted to support computational steering of grid-based simulations.

You could then visualize results as they're generated in one Web browser window and modify simulation parameters in another window.

The Web application server currently encapsulates the visualization session from the AMRWebSheet. Although the system stores previous sessions, we could exploit more information stored within them. For example, when the system renders the same result in two different visualization sessions, the server stores this result and its corresponding p-set multiple times. By integrating the session information management with the planned application server DBMS, we could eliminate this redundant storage. In addition, storing visualization session information in a DBMS would let different portal applications use the session. Potentially, the portal designers could analyze this session information, combined with other portal usage information stored by the DBMS, to better understand how scientists use the system. This understanding could then lead to future improvements of the portal and its applications for grid-based visual data exploration. ■

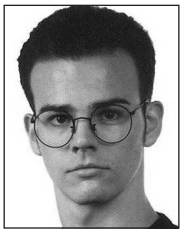
Acknowledgments

This work was supported by the National Science Foundation, the Lawrence Berkeley and Lawrence Livermore National Laboratories, and the Director, Office of Science, of the US Department of Energy under contract DEAC0376SF00098. We used the argon bubble data set courtesy of the Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory. We thank the members of the University of California, Davis, Visualization and Graphics Research Group and the Lawrence Berkeley National Laboratory Visualization Group for their input and assistance. We especially thank Tom Hsu and Praveenkumar Shetty for their work on the portal, Jason Novotny for creating the Grid Portal Development Kit and for considerable technical support, and Xia Liu for her work on the DBMS for distributed applications.

References

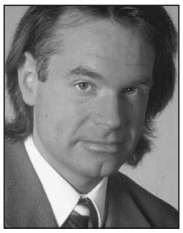
1. M. Russel et al., "The Astrophysics Simulation Collaboratory: A Science Portal Enabling Community Software Development," *J. Cluster Computing*, vol. 5, no. 3, Jan./Mar. 2002, pp. 297-304.
2. T.J. Jankun-Kelly and K-.L. Ma, "Visualization Exploration and Encapsulation via a Spreadsheet-like Interface," *IEEE Tran. Visualization and Computer Graphics*, vol. 7, no. 3, July–Sept. 2001, pp. 275-287.
3. T. Abel, G.L. Bryan, and M. L. Norman, "The Formation of the First Star in the Universe," *Science*, vol. 295, no. 5552, Jan. 2002, pp. 93-98.
4. M.L. Norman et al., "Diving Deep: Data Management and Visualization Strategies for Adaptive Mesh Refinement Simulations," *Computing in Science and Engineering*, vol. 1, no. 4, July/Aug. 1999, pp. 22-32.

T.J. Jankun-Kelly *is a PhD candidate in computer science at the University of California, Davis, where he is a member of the Visualization and Computer Graphics Research Group of the Center for Image Processing and Integrated Computing. His research interests include scientific and information visualization, Web programming, and theory. He received a BS in physics and computer science from Harvey Mudd College and an MS in computer science from the University of California, Davis. He is a member of ACM, ACM/Siggraph, the IEEE, and the IEEE Computer Society.*



Oliver Kreylos *is a PhD candidate in computer science at the University of California, Davis. His research interests include multiresolution methods for scientific visualization and interaction techniques for virtual-reality environments. He received a Diploma in computer science from the Universitat Karlsruhe.*

Kwan-Liu Ma *is an associate professor of computer science at the University of California, Davis. His research interests include improving the overall experience and performance of data visualization through more effective interactive techniques, user interface designs, expressive rendering, and high-performance computing. He received a PhD in computer science from the University of Utah. He is a member of IEEE, the IEEE Computer Society, and ACM.*



Bernd Hamann *is codirector of the Center for Image Processing and Integrated Computing and a professor of computer science at the University of California, Davis. He is an adjunct professor of computer science at Mississippi State University, a faculty computer scientist at the Lawrence Berkeley National Laboratory, and a participating guest researcher at the Lawrence Livermore National Laboratory. His research interests include visualization, geometric modeling and computer-aided geometric design, computer graphics, and virtual reality. He received a PhD in computer science from Arizona State University. He is a member of ACM, IEEE, the Society for Industrial and Applied Mathematics, and the IEEE Technical Committee on Visualization and Graphics.*

Kenneth I. Joy *is a professor of computer science at the University of California, Davis; a faculty researcher in the Center for Image Processing and Integrated Computing; and a faculty member in the Institute for Advanced Scientific Computing Research at Lawrence Livermore National Laboratory. His research interests include visualization, geometric modeling, and computer graphics. He received a PhD in computer science from the University of Colorado, Boulder. He is a member of ACM, the IEEE Computer Society, and the Society for Industrial and Applied Mathematics.*

John Shalf *is a researcher in the Visualization Group at Lawrence Berkeley National Laboratory. His research interests include projects that cover visualization of AMR data, distributed-remote visualization, grid-portal technology, high-performance networking, and computer architecture. He is a member of the Technical Advisory Board for the European Union GridLab project, which seeks to create application-oriented APIs and frameworks for grid computing.*

E. Wes Bethel *is a staff scientist at Lawrence Berkeley National Laboratory, where he is a member of the Scientific Visualization group. His research interests include computer graphics and visualization software architecture, remote and distributed visualization algorithms, latency-tolerant and parallel-graphics techniques. He received an MS in computer science from the University of Tulsa. He is a member of ACM, ACM/Siggraph, and the IEEE.*

Readers may contact T.J. Jankun-Kelly at the Visualization and Graphics Research Group, Center for Image Processing and Integrated Computing, Dept. of Computer Science, Univ. of California, Davis, CA 95616-8562, email tjk@acm.org.

For further information on this or any other computing topic, please visit our Digital Library at http://computer.org/publications/dlib.