Title
Deploying Web-based Visual Exploration Tools on the Grid

Journal
IEEE Computer Graphics and Applications, 23

Authors
Jankun-Kelly, T. J.
Kreylos, Oliver
Shalf, John M.
et al.

Publication Date
2003

Peer reviewed

Deploying Web-based Visual Exploration Tools on the Grid

T.J. Jankun-Kelly* Oliver Kreylos* John Shalf† Kwan-Liu Ma*
Bernd Hamann* Kenneth I. Joy* E. Wes Bethel†

Abstract

We discuss a web-based portal for the exploration, encapsulation, and dissemination of visualization results over the Grid. This portal integrates three components: an interface client for structured visualization exploration, a visualization web application to manage the generation and capture of the visualization results, and a centralized portal application server to access and manage grid resources. Our approach uses standard web technologies to make the system accessible with minimal user setup. We demonstrate the usefulness of the developed system using an example for Adaptive Mesh Refinement (AMR) data visualization.

Keywords: scientific visualization, grid-based computing, world-wide web, visualization interfaces, adaptive mesh refinement

Introduction

The easy access to low-cost, high-performance, network-aware computers has had a great impact on the way scientists conduct their research. Their productivity has improved but they are burdened by the increasing size of data being generated. Visualization is an effective and economical means to explore and communicate with the data and insight obtained in scientific studies. However, due to the size of the generated data, the scientists, their data, and the visualization software are often located on different machines—machines potentially located at geographically distributed locations. Grid-based computing solves some of these problems by managing access and utilization of these different resources. This management, however, is not centralized. Thus, to effectively use grid resources, a central access point is needed. This access point manages the resources, provides a visual means to explore the data, and records these exploration for further investigation and dissemination. This article describes such a system 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. The portal presents a unified interface to the distributed resources via a central application server. The application server communicates with services on the Grid in response to request made by users utilizing the portal. Thus, the portal hides the complexity of grid security and job-launching mechanisms. Scientists utilizing the portal focus on the important task of extracting insight from their data via visualization instead of worrying about the ancillary tasks of data and process management.

Scientists at LBNL and their collaborators require access to the portal world-wide. The computing environments available to these scientists run the gamut of hardware, operating systems, and installed software. To support uniform access to the portal, the portal's interface is entirely web-based. Authenticated users only need a standards-compliant web browser to visually explore their data from anywhere in the world. The impetus for this design decision and its consequences are discussed elsewhere in this article.

A central service the portal provides is a web-based interface for the exploration and encapsulation of visualization data. A structured visualization interface provides an efficient means to interact with scientific data. The encapsulation of the process allows a user to reproduce the visualization results for validation or to extend those results by continuing data exploration. In this article, we discuss the integration of the grid-enabled visualization portal/application server, the visualization web application which performs the visualization session management, and the web-based interface, using an implementation for the visualization of Adaptive Mesh Refinement data—the AMRWebSheet—as an example.

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 in such a way that its complexity and location are abstracted away and hidden from the user. Another aspect of portal technology is that it provides location-independent access to state-information. It does not matter where you are, when you login to the URL of the portal's interface, you get access to the same view of your personalized environment and data (e.g. your email). Yahoo! and HotMail are typical consumer-oriented examples of this capability and are in fact the originators of this new meaning for the term "portal".

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 Grid "virtual organization." Many virtual collaboratory and distributed application developers have turned to grid portals as the primary way of hiding the complexity of distributed applications under a single interface. Consequently, a number of portal development toolkits have emerged including the 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/, see [1]).

Portal interfaces need not be web-based, but web portals have been widely adopted by the Grid community. This is in part due to 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 platform and the comparatively uniform cross-platform programming model it offers for the client UI, it makes an attractive platform for a widely-deployed client interface to Grid services. Furthermore, it requires essentially no custom software installation for the scientists who use the service (a task that many are loath to perform and would otherwise limit acceptance of the tool).

We regard the spreadsheet approach as an excellent match for the interaction modality of the web. Spreadsheets are immediately able to capitalize on the HTML table display paradigm as well as the use of hyperlinks to drill down into information content. Such an interface will be particularly amenable to integration with other ongoing portal development efforts throughout the Grid community.

*Visualization and Graphics Research Group, Center for Image Processing and Integrated Computing, Department of Computer Science, University of California, Davis, CA 95616. E-mail: {kelly,kreylos,ma,hamann,joy}@cs.ucdavis.edu

†Visualization Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720. E-mail: {jshalf,ewbethel}@lbl.gov

Related Web and Grid-based Visualization Work

Web-based control of visualizations using the Grid combines research in two active areas: Web-based visualization systems and distributed visualization in grid-like environments. Though there has been limited interaction between these fields before, most previous research has focused on only one them.

Ang [1] described one of the first web-based visualization systems. In Ang's system, visualization results are displayed 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 [2] generalized Ang's approach to discuss four different compositions of Web-based visualization. In their first scenario, only images are sent to the client with no user interaction with the visualization. Their second scenario allows a user to manipulate the result (for example, interaction with a VRML model), but a user cannot 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. Their last scenario, and the one implemented by Wood and Ang, supports web-based interaction for controlling the parameters and the visualization type performed without requiring a significant client installation—in Wood's case, parameters for IRIS Explorer we sent by the client while VRML (in one of its first web visualization usages) was sent back. 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 scenario: only images are ever sent to the client, but client interaction with the web page causes updates to the visualization process.

Lefer [3] and Bajaj et al. [4] have 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 (LAN). Another interesting property of Lefer's approach is that interaction with the visualization system is done entirely through HTML-based forms—no external applet is needed. We use an all-HTML approach as well, but augment it with enhanced interaction via JavaScript. This approach eliminates the need for HTML forms by allowing the JavaScript events to invoke actions on the visualization server. Bajaj and Cutchin's work [4], unlike the previous approaches mentioned, also coordinates the users, distributed resources, and the utilization of those resources. This coordination is in addition to the coordination of the visualization. Our web interface does not manage the grid resources; this task is handled by the underlying visualization portal as a whole.

Visualization utilizing grid-resources that do not involve web-based interfaces have also been investigated. Three representative examples are presented. GridMapper [5] addresses the problem of determining the performance of grid computations by collating and visualizing this distributed information. This information is dynamically gathered from the sites performing the computation on the Grid. TeraVision [6] allows users to 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 rendering volume renderer (as described elsewhere in this issue). Each of these approaches (except the web-interface for Cactus) require a "thick-client" installation to perform the visualization. In other words, for any particular remote user, their platform and system capabilities must be determined before the appropriate grid visualization client for this system can be installed and launched—if an appropriate version exists. Our approach eliminates the need for a thick-client by using a web-browser—assumed to be installed on the user's system—to perform the visualization.

References

[1] C. S. Ang, D. C. Martin, and M. D. Doyle. Integrated control of distributed volume visualization through the world-wide-web. In R. D. Bergeron and A. E. Kaufman, editors, *Proceedings of the IEEE Conference on Visualization 1994 (Vis'94)*, pages 13–20. IEEE Computer Society Press, Los Alamitos, CA, October 17–21 1994.

[2] J. Wood, K. Brodlie, and H. Wright. Visualization over the world wide web and its application to environmental data. In R. Yagel and G. M. Nielson, editors, *Proceedings of the IEEE Conference on Visualization 1996 (Vis '96)*, pages 81–86, 470. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[3] W. Lefer. A distributed architecture for a web-based visualization service. In D. Bartz, editor, *Proceedings of the Ninth Eurographics Workshop on Visualization in Scientific Computing*. 1998.

[4] C. Bajaj and S. Cutchin. Web based collaborative visualization of distributed and parallel simulation. In J. Ahrens, A. Chalmers, and H.-W. Shen, editors, *Proceedings of the 1999 IEEE Parallel Visualization and Graphics Symposium (PVG'99)*, pages 47–54. IEEE Computer Society Press, Los Alamitos, CA, October 25–26 1999.

[5] W. Allcock, J. Bester, J. Bresnahan, I. Foster, J. Gawor, J. A. Insley, J. M. Link, , and M. E. Papka. GridMapper: A tool for visualizing the behavior of large-scale distributed systems. In *Proceedings of High Performance Distributed Computing*. 2002.

[6] J. Leigh, J. Girado, R. Singh, A. Johnson, K. Park, and T. A. DeFanti. TeraVision: a platform and software independent solution for real time display distribution in advanced collaborative environments. In *Proceedings of the Access Grid Retreat, 2002*. 2002. http://www-fp.mcs.anl.gov/fl/accessgrid/ag-retreat-2002/proceedings/leigh-tera.pdf.

[7] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, H.-C. Hege, A. Merzky, J. Masso, and J. Shalf. Solving Einstein's equations on supercomputers. *IEEE Computer*, 32, 1999.

Portal for Web-based Visualization Exploration and Encapsulation

The system we are 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 which tracks the exploration of visualization results, and the portal application server that manages and coordinates the authentication for and use of grid resources (including the interface, web application, and volume renderer). The application server (the VisPortal) uses established grid technologies to handle user and resource management (such as data transport). Once authenticated, a new visualization exploration session is initialized by the web application; the web application (also called a *servlet*) is a program on the web server that communicates with the

client via HTTP (the Hypertext Transfer Protocol—the protocol for the World-Wide Web). In our case, the servlet maintains the visualization session state. After the visualization session is initialized, the web-based visualization interface is loaded in the client's web browser. As the visualization session progresses, the visualization results and the relationships between those results are stored by the web application for later examination. Finally, when the user is finished visualizing their data, the session is closed. The user can then initialize another session or re-examine previous explorations. We explain the interplay between the interface and the web application next.

Web-based Sheet-like Interface for Visualization

Our web interface implements an entirely web-based version of the visualization exploration sheet-like interface discussed in [2]. The original spreadsheet-like interface (the VisSheet, for short) was designed to assist visualization exploration by providing context for where a user is in their exploration, where they have been, and suggesting where they may go next. The VisSheet handles these tasks by providing a movable, scalable window into the visualization parameter space. By manipulating the visualization parameters, the user changes the position and size of this window. Only two visualization parameters are displayed at a time: one along the rows and another along the columns. For the non-displayed parameters, a set of default values is maintained that may be updated at runtime. Parameter values are rendered as glyphs. Cells—representing a combination of the row, column, and default parameter values—display the visualization results. By changing the default values for non-displayed parameters or which parameters are displayed along the rows or columns, the window can be moved 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 allow a user to easily modify default and displayed parameters via the default parameter bar and drop-down row and column parameter lists (Figure 1). The default bar assists in the identification of 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 these two systems, 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, a platform-independent solution was desired. 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 Macromedia's Flash or Sun's Java could not be used. 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. No permanent state can be stored on the client machine. This limitation is again due to the wide variety of potential platforms for portal users: some may be unable to store such state. By keeping state in a centralized, web-accessible location, visualization sessions can be re-examined by the same user in different locations without loss of information. Due to these constraints, the interface is a web-based and not a single-user, network-unaware Java application like the VisSheet.

There are several consequences resulting from our web interface

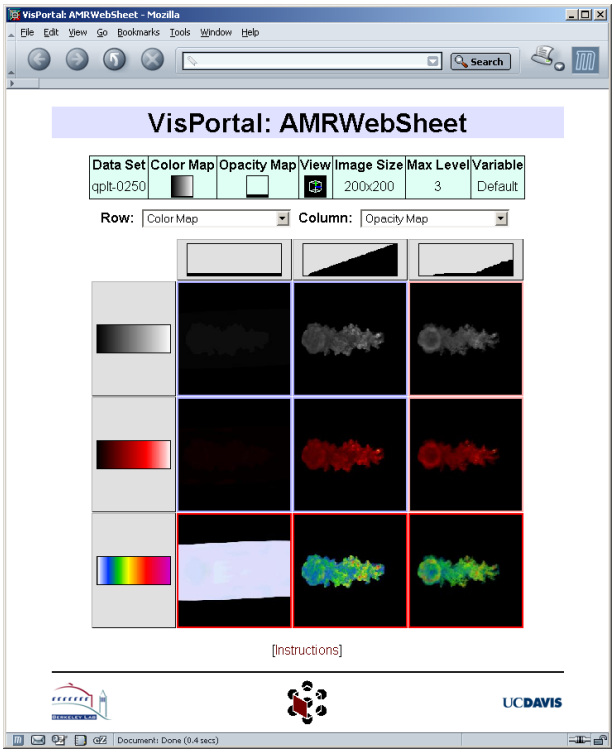

Figure 1: The AMRWebSheet interface, an example of our web interface to grid-based visualizations. The interface consists of three major areas: The default parameter bar that displays and allows the modification of the default parameter values; the displayed row and column parameter drop-down lists;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

design. First, interactivity partially suffers in comparison to the VisSheet. The VisSheet uses the Java Foundation Classes (JFC) for its user interface (UI). The JFC supports a rich set of user interface elements and customizability. In contrast, the interface elements offered by HTML are limited: only checkboxes, radio button, push-buttons, lists, menus, and text fields/areas are supported with little customizability. The use of JavaScript overcomes many of these limitations by allowing different portions of the HTML page to react to mouse events. For example, JavaScript can detect a user dragging the mouse in the view position editor of the AMRWebSheet; the resulting event causes the client to update the corresponding parameter display. Interactivity is still impacted—HTML cannot be used to render glyph icons representing the parameter values in a drop-down list for the default parameter bar, for example (again, JavaScript is used). The complexity of implementing interface interaction is also increased in comparison to the VisSheet implementation—the JFC provides more functionality built-in compared to raw JavaScript. Finally, our web interface cannot query any significant information about a client's machine. Methods exist to extract the client's browser information or to download a single file off of the client machine at a time, but such transfers are not optimized for the large data set sizes common in visualization applications.

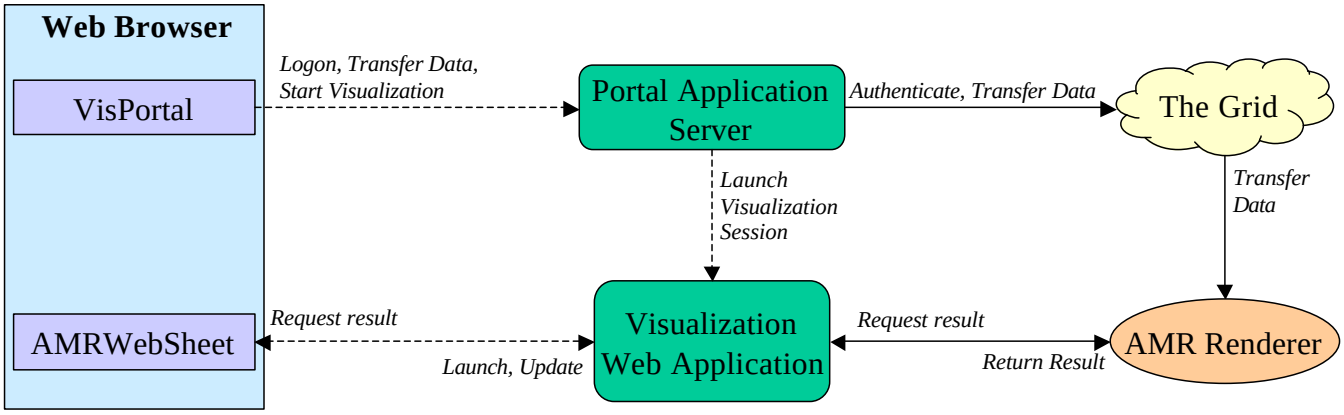

Figure 2: The VisPortal/AMRWebSheet architecture. Elements within the blue box represent web pages that the user interacts with, green boxes represent web servers, and the other nodes represent grid-accessible resources. Dashed lines are HTTP/HTTPS connections for HTML and images (for the client) and connection requests (for the server), and solid lines are 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. Note that the AMR renderer can be located anywhere on the Grid—the VisPortal initializes the connection between the renderer and the visualization web application when a visualization is first requested.

Web-based Encapsulation of Visualizations

The web-based visualization interface structures the visualization exploration process. The visualization web application server captures this process. By capturing the process, we ensure that the visualization results generated, and the relationships between those results, are not lost when the visualization session ends. To record the visualization process, a formal model of the visualization exploration process is used (see the Visualization Exploration Model Sidebar, page 5). As each requested image is rendered, the corresponding visualization session result is stored by the web application server. Thus, at the end of a session, all the rendered images, the parameter value sets (p-sets) used for creating that image, when that 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: the user may start a new visualization session or view previous sessions. When the user chooses to start a new session, another servlet, the UI servlet, is loaded to handle interactions with the visualization UI. 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 UI servlet represents the state of the UI; our web interface web page presents the view of this state.

If a user chooses to examine previous sessions, the session servlet is loaded. Initially, a list of all the previous explorations, sorted by date, is presented to the user. The list supports three actions. A user can re-load a previous session in our web interface by clicking on its corresponding link. New results can be added to this session; when the session terminates, these results will be stored along with the old session information. This capability is crucial to the VisPortal—scientists must be able to distribute their work over time as well as over space.

The second service the session servlet supports is the viewing of 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 for that result 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 are allowed to add or edit annotations of results. These annotations are stored on the web application server for others to access. Scientists can use these annotations to flag certain results as "interesting" to collaborators.

The session servlet also allows a user to view an overview graph of a visualization. While the HTML session document describes the visualization in detail, it is 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 graph is displayed. All the graphs use a new radial focus+context visualization technique. In this technique, the radial distance from the center node to another node represents the distance of that node's result p-sets 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 in order to allow the system to 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 means of understanding what occurred during a visualization session.

Application Domain: Web-based AMR Data Visualization

The web interface and visualization web application we have described can be applied to a variety of scientific visualization problems. Of particular interest to scientists at LBNL is the visualization of Adaptive Mesh Refinement (AMR) data. This section discusses a specific implementation of our grid-enabled, web-based visualization system for the exploration of AMR data—the AMRWebSheet.

Many of the most challenging problems in numerical modeling involve meshes with huge ratios of scale. For instance, when modeling a fuel injection system of an automobile, one must model the fluid dynamics of the 30 μm orifice of the injector as well as the dy-

Visualization Exploration Process Model

The visualization process for both information and scientific visualization is an iterative sequence of user-applied transformations from data to view [1, 2, 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, or *p-sets*, posses 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. In [4], a model of the visualization process based upon a parameter derivation calculus is described. The calculus defines how p-sets—and thus the results rendered from them—are derived from previous p-sets. New p-sets are created by user interaction with the visualization system in one of three ways:

- **Parameter Application.** Parameter values from a p-set are applied to another p-set to generate a new p-set. Example: A new color map replaces an old color map in a previously generated p-set/result in order to render a new result from the new p-set.
- **Parameter Range Sweep.** A single parameter value is interactively manipulated over a range between an initial and final p-set. Example: A range of view positions is generated 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. Example: A new opacity map is created by applying a set union operator to two previously used opacity maps.

In the AMRWebSheet, only two types of parameter derivations are used: parameter application and function parameter generation. When a cell is rendered, the parameters for that cell are collected 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 an operator is applied 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 timestamp 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—e.g. when applying a parameter operator—multiple session results can share the same timestamp. 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 snap-shots are stored for future collaboration. In our system, the complete exploration and derivation information (encapsulated in session results) is stored as an XML document on the portal for later access and re-exploration.

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, T. A. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[3] R. B. Haber and D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. In G. Nielson and B. Shriver, editors, *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990.

[4] T. J. Jankun-Kelly, K.-L. Ma, and M. Gertz. A model for the visualization exploration process. In R. J. Moorhead, M. Gross, and K. I. Joy, editors, *Proceedings of the IEEE Conference on Visualization 2002 (Vis '02)*. IEEE Computer Science Press, Los Alamitos, CA, 2002.

[5] K. Brodlie, S. Lovegrove, and J. Wood. Harnessing the web for scientific visualization. *Computer Graphics*, 34(1), February 2000.

namics of fuel-air mixing in a cylinder chamber that is 10 cm long or more. Cosmologists investigating the structure of the universe require simulations that model the formation of large-scale structures (superclusters) consisting of "clusters of clusters of galaxies." These simulations resolve relevant features down to individual stars. 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 they are needed. For instance, large-scale structures of the superclusters in the cosmology example are relatively compact; it would be a waste to model the 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 are placed. Furthermore, refined regions themselves may 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 of Mike Norman, Greg Bryan, and Tom 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, but they also pose significant challenges for visualization researchers.

AMR data structures do not fit into any of the traditional data structures that are 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. Naïve 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 interfaces at the coarse-fine mesh boundaries can occur and cause "cracks." Finally, direct application of finite-difference techniques to an AMR hierarchy leads to visual artifacts at the coarse-fine boundaries as well as significant data management issues—a typical desktop system cannot process these deep hierarchies inter-

actively. Consequently, there are few visualization algorithms that can be directly applied to hierarchical meshes, and essentially no off-the-shelf commercial software is available for visualizing AMR data. It is critically 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 the remote visualization of the data, we have developed a parallel multiprocessor hardware accelerated volume renderer for AMR data (for more information on the actual AMR volume rendering, see the sidebar on page 7). Since the majority of LBNL visualization users are off-site and have comparatively smaller resources at their disposal, we created a client-server architecture so that the entire system is accessible over the Grid using a traditional client interface. The goal of the AMRWebSheet project is to extend access to this parallel rendering back-end using an entirely web-based Grid portal interface that is suitable for embedding in many emerging web-based co-laboratories.

Architecture

Figure 2 summarizes the VisPortal/AMRWebSheet architecture. The AMRWebSheet interface and web application are implemented in a flexible visualization exploration and encapsulation framework. The framework, implemented in Python (http://www.python.org/), consists of a series of objects that manage visualization sessions and a visualization UI'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 exists to represent the interactions between visualization UIs and the visualization session in a platform independent manner. Other UI and UI toolkit-independent objects representing general visualization exploration spreadsheet views and state also exist within the framework. These later classes are used 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 application uses the framework to implement the AMRWebSheet. A Java applet that combines the properties of both approaches has also been created.

The web application servlets that manage visualization sessions are implemented in Python using the Webware web application environment (http://webware.sourceforge.net/). We use the Apache web server, running under Linux. A group of servlets create, process, and store sessions. When a client connects, a new session—identified by a temporary cookie—is created in addition to servlet-persistent objects. Whenever a user interacts with the generated HTML interface—the AMRWebSheet—HTTP requests are communicated to the interface servlet indicating that the behavior fired. This request in turn modifies the visualization session state. When the client needs to be updated—e.g., after result generation—a server-initiated refresh is performed to display the new information. Finally, when a session terminates or expires due to inactivity, the session results are encoded as an XML document on the web application server for later retrieval as described previously.

The web 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. This communication is enabled by a pure Python implementation of the AMR volume renderer's client-server protocol. When the corresponding result is returned by the volume renderer, a copy is stored by the web application in order to store the visualization session result. This result is then displayed by forcing a refresh on the client's web browser.

Access to the visualization interface is handled by the encompassing VisPortal. The VisPortal provides a single point of access to launch and control all of the components of this distributed tool. The architecture of the VisPortal is based on the Grid Portal Development Kit (GPDK, http://www-itg.lbl.gov/grid/projects/GPDK/) that uses the Java Commodity Grid (CoG) toolkit (http://www.globus.org/cog/java/) in conjunction with an Open Source Java Server Pages application server (Tomcat: 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 in a secure fashion.

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 on the user's behalf using only a single login. From the standpoint of the user, the portal hides a complex application launching mechanism for a multi-component distributed application. In the case of a thick-client application, the portal launches a parallel computing component using the Globus GRAM, brokers a direct socket connection between this computing component and a high-performance back-end data source like a running simulation code. It then launches the thick-client through the web-browser using appropriate MIME-type definitions. The thick-client in turn connects back to the remotely located parallel visualization component, thereby completing the distributed visualization application. This entire elaborate launching procedure is hidden entirely from the user by the portal client interface. The user simply selects remotely located data and presses a button to start the visualization application.

The AMRWebSheet supports an even simpler launching mechanism whereby the back-end is simply started using the GRAM on the resource that contains the data set. 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 presented in the user's web-browser. If the back-end is located on a Silicon Graphics machine, the 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 complexity of grid architecture and distributed applications is hidden from the user by the portal client interface.

The performance of the AMRWebSheet depends on three major factors: the performance of the AMR volume renderer, the performance of the visualization application server, and the performance of the user's web browser. Variance in network traffic between the renderer, application server, and client can also effect performance. Table 1 provides performance measures for the elements under our control: the volume renderer and the application server. In this example, an AMR data set consisting of 501 timesteps and $640 \times 256 \times 256$ cells at the finest level was used. Performance was measured in seconds for three different hierarchy levels (0 being the coarsest and 2 being the finest) using an 800x600 pixel image as our output. Two sets of performance data were collected: 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 pre-loaded data (and pre-generated textures). As the table shows, the rendering time does depend on the maximum level rendered while the application server processing time is nearly constant. This result is to be expected since the application server only processes completed images, which do not depend significantly on the rendered hierarchy level. It is important to note that 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 other new images) are either cached by the web-

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 cannot be rendered directly using graphics hardware's texture mapping capabilities (Figure 1). Instead, a given AMR hierarchy has to be *homogenized*, i.e., it has to be transformed into a set of non-overlapping rectangular grids with acyclic visibility order for any viewing direction. This process generally involves removing parts of lower-resolution grids that are overlayed by higher-resolution grids, and splitting of the resulting non-convex 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 is refined as AMR grids are inserted one at a time, starting with lowest-resolution grids. A 2D AMR hierarchy and its homogenizing k-d tree are shown in Figure 2.

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 α-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 (rendering cost is estimated during k-d tree traversal). The sequences are then assigned to rendering nodes. This step is performed on all nodes in parallel and does not 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. Since the rendering bottleneck is grid patch rendering, and compositing itself is performed in hardware, a simple binary tree compositing strategy is sufficient; it could be replaced with a binary-swap compositing strategy should the need arise—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-only, parallel renderer can also be used on cluster or distributed-memory architectures. At UC Davis, the renderer runs on two Linux clusters (with four and 16 nodes, respectively) using NVidia GeForce3 graphics cards for rendering and 100 BaseT ethernet for inter-node communication. More information is provided in [1].

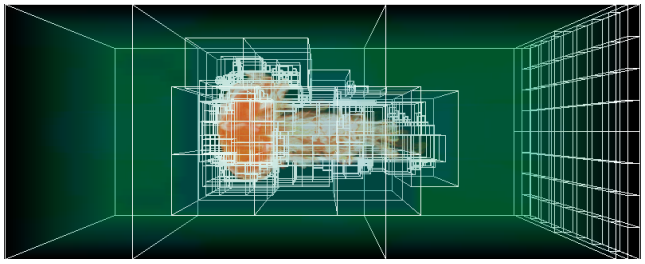

Figure 1: Volume rendering of the argon bubble with superimposed AMR grid hierarchy.

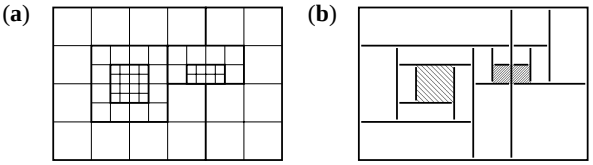

Figure 2: Homogenizing a 2D AMR hierarchy. The hierarchy has a uniform refinement ratio of two. Grid boundaries are denoted by bold lines. All hierarchy levels consist of two grids. Note that finer grids can cross boundaries between coarser grids. (a) Original AMR hierarchy with overlapping grids. (b) Homogenized hierarchy with non-overlapping rectangular grid patches.

References

[1] O. Kreylos, G. H. Weber, E. W. Bethel, J. Shalf, B. Hamann, and K. I. Joy. Remote interactive direct volume rendering of AMR data. Technical Report LBNL-49954, Lawrence Berkeley National Laboratory, 2002.

browser (which does not 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 present a typical scenario. In our example, 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, the scientist 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. Since Alice's virtual organization allows her to access the AMR renderer via the

No. of	Initial Data Load		Pre-Loaded Data	
Levels	Renderer	App. Server	Renderer	App. Server
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

Table 1: Performance measurements the AMR volume renderer and the AMRWebSheet application server versus the maximum level of the hierarchy rendered. For all examples, the argon bubble data set discussed in the usage scenario was used to generate 800x600 pixel images. The renderer used an SGI Onyx3400 while the application server used an 1.8 GHz Intel Pentium 4. The first columns lists times needed to render initial image (including data load); the second column lists the time needed for the application server to process the image from the first column; the third column lists time needed to render an image after the data has been loaded; and the fourth column lists the time needed for the application server to process the image from the third column. Note that application server processing only occurs once (when the result is first rendered). All measurements are in seconds.

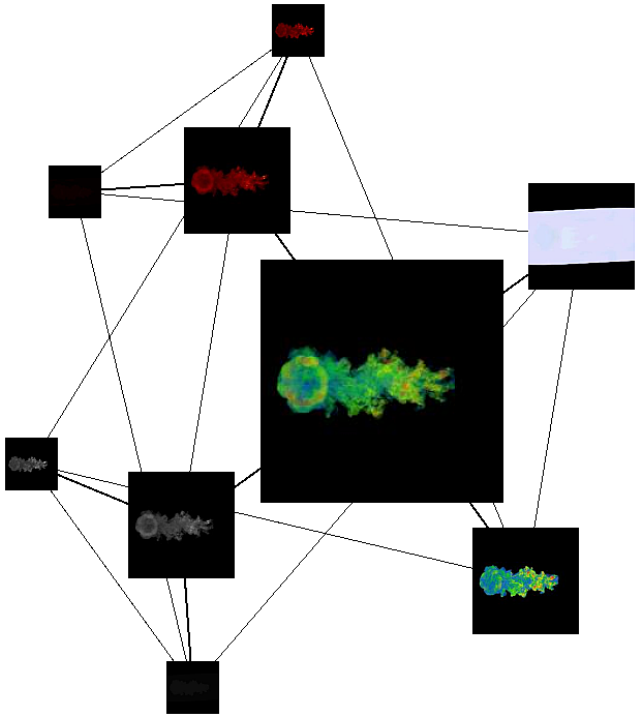

Figure 3: Parameter different session graph for the session in Figure 1. Edges indicate that only one parameter value differs between the two resulting images. Session graphs provide an overview of different information about the visualization session.

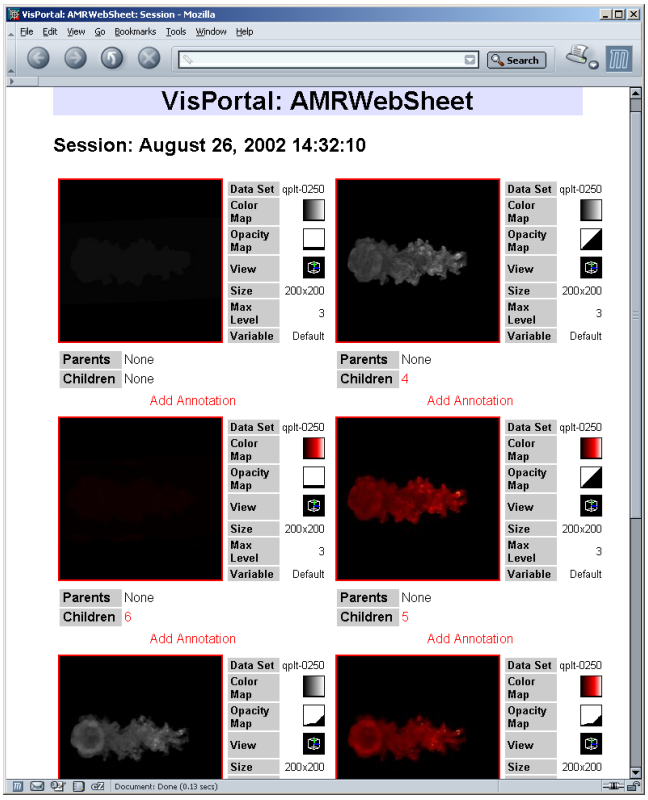

Figure 4: HTML session page for the session in Figure 1. The page provides a summary of the visualization session and supports the annotation of results.

Grid, this transfer is authenticated. The complexities of the transfer (such as using GridFTP) are hidden from Alice; as such, they can be changed to improve efficiency without affecting users. Alice then requests a new visualization session from the portal. Again, Alice's credentials are verified, this time confirming that she can access the visualization service; all of 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 desires to start a new visualization or view/expand an older session. In this scenario, she starts a new visualization session. After specifying an initial data set, the AMRWebSheet page is loaded in Alice's browser, a few results already generated from the default parameter values the AMRWebSheet uses. She then explores the data via the web-page interface until she is satisfied with the results. At this point, Alice terminates the visualization session and exits the portal. When Alice exits, the visualization session is automatically recorded by the system.

At some later date, a colleague of Alice 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 requests to view a previous visualization session instead of starting 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 allows these scientists to focus on using their data, not managing it.

Conclusions and Future Work

Our VisPortal project provides centralized access to grid-enabled resources world-wide. We have described an entirely web-based UI for exploring data created using the portal. The interface is coupled with a web application for recording and managing visualization results. Combined, these tools provide a platform for universally accessible visual exploration of scientific data over the Grid.

Scientists benefit in several ways by using the portal and the AMRWebSheet. The portal and the AMRWebSheet are based on standard HTML; scientists using web browsers from any location can generate and explore their data without requiring significant client installation. The visualization environment structures the visual investigation of the data, preventing costly re-exploration. Remote collaborators can access these explorations on the portal to validate their colleagues' results. Since entire visualization sessions are captured, previous sessions are a launching point for further data exploration. The AMRWebSheet is a Grid application that makes visualization easy to access and utilize by scientists.

The work described here represents one aspect of the VisPortal project. Three areas are under active development: the underlying portal application server, the visual exploration tools available, and the management of visualization sessions. For the application server, current work focuses on improving its low-level implementation and the connection between GPDK and the various CoGs. This work includes adding support for a Python CoG for easier in-

tegration of the visualization exploration and encapsulation framework with the Grid. Finally, integration of a database management system (DBMS) with the application server is underway. Once complete, authentication, session management, and resource allocation will utilize the DBMS to record portal-wide usage behavior.

The AMRWebSheet is only one of several visualization UIs 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. Alternate web-based VisSheet implementations are also being investigated. For example, a VisSheet-like interface for visualizations using the Visualization Toolkit (VTK) would vastly increase the potential number of visualization applications used by scientists interacting with the portal. Additionally, we are interested in utilizing 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. Finally, the interface could be adapted to support computational steering of grid-based simulations. A form of this is already possible—one could visualize results as they are generated in one web browser window and modify simulation parameters in another window. A stronger coupling is possible. For example, the simulation parameters could be incorporated into the visualization transform the interface displays. Then, changes to these parameters would not only update the visualization but could be also communicated to the simulation.

The web application server currently encapsulates the visualization session from the AMRWebSheet. Though previous sessions are stored, more information stored within these sessions can be exploited. For example, when the same result is rendered in two different visualization sessions, this result and its corresponding p-set are stored multiple times on the server. By integrating the session information management with the planned application server DBMS, this redundant storage is eliminated. In addition, storing visualization session information in a DBMS allows the session to be used by different portal applications. Potentially, this session information, combined with other portal usage information stored by the DBMS, can be analyzed and visualized by the portal designers to better understand how scientists are utilizing the system. This understanding can 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 U.S. Department of Energy under contract DE-AC03-76SF00098. The argon bubble data set is courtesy of the Center for Computational Sciences and Engineering, Lawrence Berkeley National Laboratory. We would like to thank the members of the UC 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, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, and G. von Laszewsi. The astrophysics simulation collaboratory: A science portal enabling community software development. *Journal of Cluster Computing*, 5(3), 2002.

[2] T. J. Jankun-Kelly and K.-L. Ma. Visualization exploration and encapsulation via a spreadsheet-like interface. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):275–287, 2001.

[3] T. Abel, G. L. Bryan, and M. L. Norman. The Formation of the First Star in the Universe. *Science*, 295:93–98, January 2002.

[4] M. L. Norman, J. Shalf, S. Levy, and G. Daues. Diving Deep: Data Management and Visualization Strategies for Adaptive Mesh Refinement Simulations. *Computing in Science and Engineering*, 1(4):22–32, 1999.