

Grid Enabled Remote Visualization of Medical Datasets

Nick J. AVIS, Ian J. GRIMSTEAD and David W. WALKER

*Cardiff School of Computer Science, Cardiff University, Queen's Buildings,
5 The Parade, Roath, Cardiff CF24 3AA, United Kingdom*

We present an architecture for remote visualization of datasets over the Grid. This permits an implementation-agnostic approach, where different systems can be discovered, reserved and orchestrated without being concerned about specific hardware configurations. We illustrate the utility of our approach to deliver high-quality interactive visualizations of medical datasets (circa 1 million triangles) to physically remote users, whose local physical resources would be otherwise overwhelmed. Our architecture extends to a full collaborative, resource-aware environment, whilst our presentation details our first proof-of-concept implementation.

1. Introduction

3D medical datasets (point based, polygonal or volumetric) and their rendering and visualization are important elements of a growing number of medical diagnosis, treatment planning, training and education activities. Traditionally to process these large datasets would require direct access to specialist graphics supercomputers, often tightly coupled to local display systems (ranging from the standard desktop to large scale immersive facilities such as CAVEs [1]). Dataset size continues to grow faster than Moore's Law [2], so whilst local processing capacities are increasing, such datasets can quickly overwhelm standard local computational infrastructures especially when users demand interactive data visualization and navigation.

Increases in network speed and connectivity are simultaneously allowing more co-operation between geographically remote teams and resources. Such developments break some of previous dependencies on the availability of local resources and afford new ways of working. Grid computing, based on service-oriented architectures such as OGSA [3], permits users to remotely access heterogeneous resources without considering their underlying implementations. This simplifies access for users and promotes the sharing of specialised equipment (such as rendering hardware). The motivation for this work arises from the desire to both investigate how interactive services can be supported within a Grid infrastructure and to remove the constraint on physical co-location of the end user with the high capability visualization engines.

We briefly review current Grid-based visualization systems and introduce our Resource-Aware Visualization Environment (RAVE). We present an overview of its architecture, and show initial test results of remote visualization using a PDA to observe remotely-rendered complex polygonal objects. We conclude this paper with a discussion of our findings and future work.

2. Previous Work

Remote access to rendering hardware enables expensive, specialist hardware to be used without having to move from the user's current place of work to a specific site. This is especially important when the user cannot move to the machine (for instance, a surgeon performing an operation) - the machine must come to the user.

OpenGL VizServer 3.1 [4] enables X11 and OpenGL applications to be shared remotely, with machines that do not have specialist high-performance graphics hardware. VizServer renders the OpenGL remotely, and transmits the rendered frame buffer to the collaborator. This has been used in a collaboration between the Manchester Royal Infirmary and the Manchester Visualization Centre [5,6], where MRI data is processed remotely by VizServer. The three-dimensional output is projected in the operating theatre, for the surgeon to refer to during the operation. This is in contrast to the two-dimensional image slice films that are usually viewed on a standard lightbox.

COVISE [7] is a modular visualization package, where one user is the master with control over the entire environment, whilst other users are slaves that can only observe. COVISE takes advantage of local processing on the slaves by only transmitting changes in data. For instance, the master sends changes in viewpoint, and the slaves then render the new view using a local copy of the data. COVISE also supports running slaves without a GUI -- these machines become remote compute servers, enabling the Master to distribute processes over multiple machines.

For a fuller review of remote visualization applications refer to [8] and [9].

3. RAVE Architecture

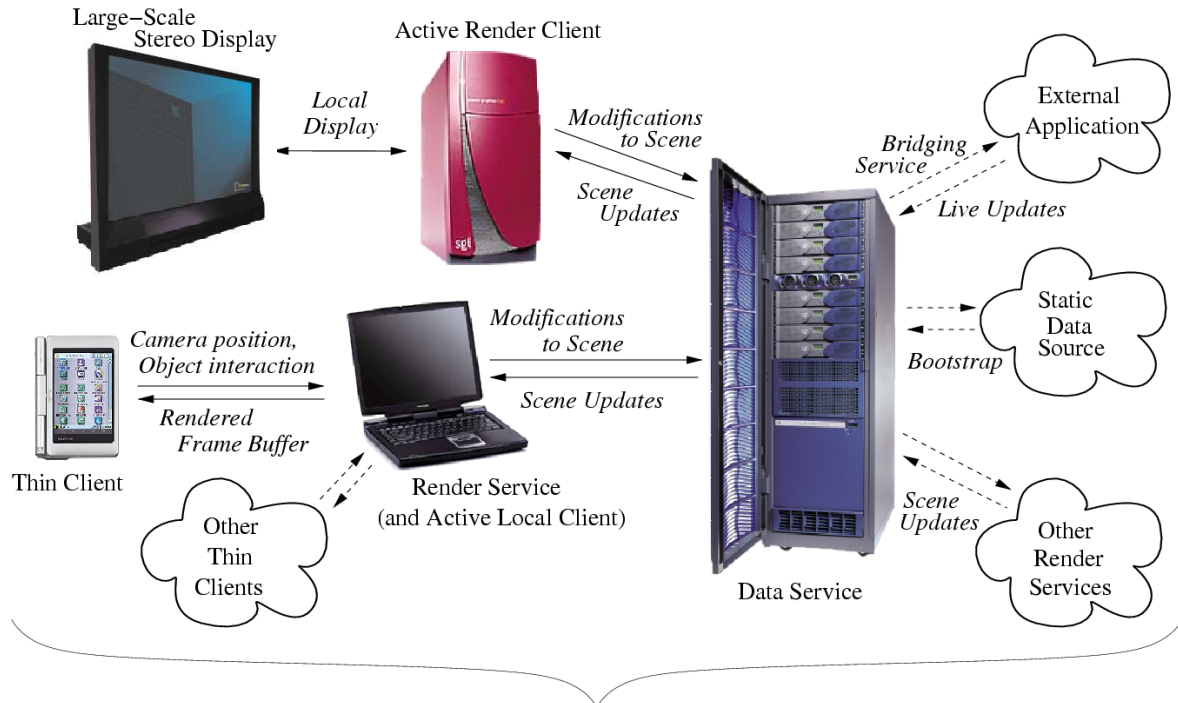
We propose a visualization system that will respond to available resources, provide a collaborative environment, and be persistent (enabling users to collaborate asynchronously with previously recorded sessions). The system must not only react to changes in resources, but also make best use of them by sharing resources between users. To implement our visualization system, we have three components: a data service, a render service and a thin client. The architecture is presented in Figure 1.

The data service imports data from either a static file/web resource or a live feed from an external program. The data service forms a persistent, central distribution point for the data to be visualized. Multiple sessions may be managed by the same data service, sharing resources between users. The data are intermittently streamed to disk, recording any changes in the form of an audit trail. A recorded session may be played back at a later date; this enables users to append to a recorded session, collaborating asynchronously with previous users who may then later continue to work with the amended session.

A client with local rendering capability (referred to as an "active client") may connect to the data service and request a copy of the data. A user can interact with the shared data through the client by modifying their viewpoint or the actual scene itself. This permits large immersive displays such as an Immersadesk R2 to be used, along with commodity hardware. The client informs the data service of any changes, which are then reflected to all other clients, to ensure all clients have an up-to-date copy of the dataset. Multiple clients simultaneously visualize the same data, forming a collaborative environment where each client is represented in the dataset by a simple avatar.

Whereas the active client is a render-capable client for the data service, a thin client (such as a PDA) represents a client that has no local rendering resource. The thin client must make use of remote rendering, so it connects to a render service and requests rendered copies of the data. Render services connect to the data service, and request a copy of the latest data in an identical manner to an active client, except that the render service can render off-screen for remote users, utilizing available rendering resources. Multiple render sessions are supported by each render service, so multiple users may share available

rendering resources. If multiple users view the same session, then a single copy of the data is stored in the render service to save resources. The thin client can still manipulate the camera and the underlying data, but the actual data processing is carried out remotely whilst the thin client only deals with information presentation (such as displaying the remotely rendered dataset).



Service Advertisement and Discovery

Figure 1: Diagram of RAVE architecture

Once Grid/Web resources (with a known API) are exposed as services, they can be inter-connected without knowledge of the contents of the modules. To open our system to any resource, we chose to implement Grid/Web services and advertise our resources through Web Service Definition Language (WSDL) documents. WSDL can be registered with a UDDI server [10], enabling remote users to find our publicly available resources and connect automatically (no configuration is required by the client, although resources may need to have access permissions modified to permit new users).

4. Initial Test Results

We used three machines in this test; a laptop running Linux as the render service, an SGI Onyx as the data service, and a Sharp Zaurus PDA as the thin client. Files are supplied remotely to the data service via HTTP or FTP connections. The use of Grid middleware permits these resources to be substituted transparently, with the user being isolated from these implementation details.

The actual demands on the PDA are for a network connection, enough memory to import and process a frame buffer (120kB for a 200x200 pixel image with 24 bits per pixel, 480kB for a 400x400 image) and a simple GUI. We are platform and operating system agnostic, so any PDA could be selected or other devices running Microsoft Windows or Sun Solaris.

4.1 Test Models

Two test models (a skeletal hand and a skull) were obtained from the Large Geometric Models Archive at Georgia Institute of Technology [11]. The models were in PLY format, converted to Wavefront OBJ and then imported into our data service. The skeletal hand is from the Stereolithography Archive at Clemson University, with a view presented from the PDA in Figure 2. The skull is a section of the Visible Man project (from the National Library of Medicine), where the skeleton was processed by marching cubes and a polygon decimation algorithm. Two sample datafiles provided with VTK [12] were also imported; the frog skeleton and the scan of a human head (both isosurfaces). The VTK files are presented in Figure 2, as displayed on the active RAVE client (running on a laptop).

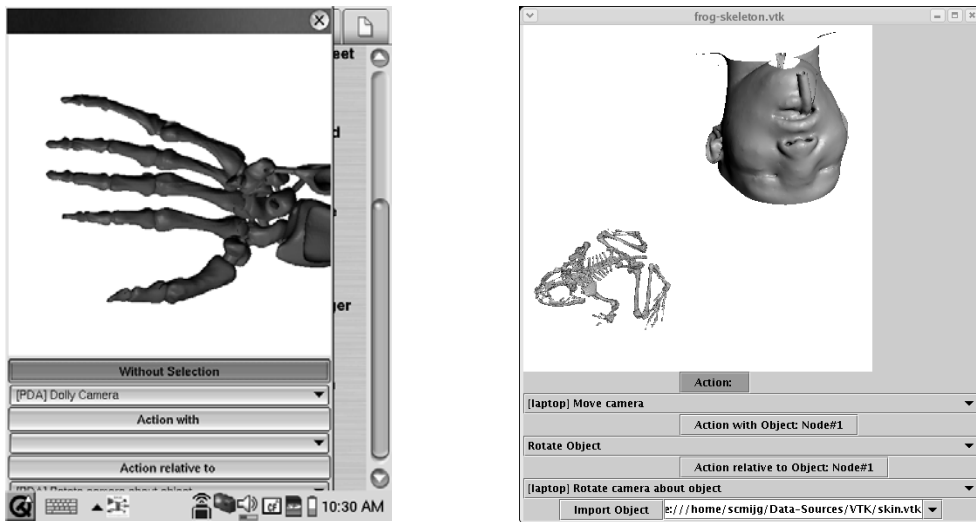


Figure 2: Skeletal hand displayed on PDA (C++), and VTK files on active client (Java)

4.2 Web Services, UDDI and RAVE

The GUIs presented in Figure 2 shows an identical menu structure, but the PDA client operates through Web Services, whilst the active client can modify local geometry directly. With the PDA, the render service is interrogated for available interactions, whilst the active client can directly contact the scene graph. The interactions are then used to populate the menu, and are activated either via a standard Web Service call or a standard Java interface. All interactions take the form of a mouse/pen drag, with an optional previously selected object. This enables us to use a standard API common to all interfaces (such as a pen based thin client PDA or mouse-based active client).

To discover what RAVE sessions are in progress, and what resources are available, we publish our services to a UDDI server, which we can then interrogate via our RAVE manager GUI, as presented in Figure 3. The services can be sorted by resource: available memory (as in Figure 3), number of session instances, million matrices processed per second, network bandwidth or latency. Only the data service section is shown here; the render service section is similar, except with an additional field, “polygons per second” for rendering performance. The user can now create a new session (providing a URL to a data source if required, otherwise an empty session can be started, ready for clients to remotely insert datasets into the session). Active or thin client sessions may also be launched from this GUI.

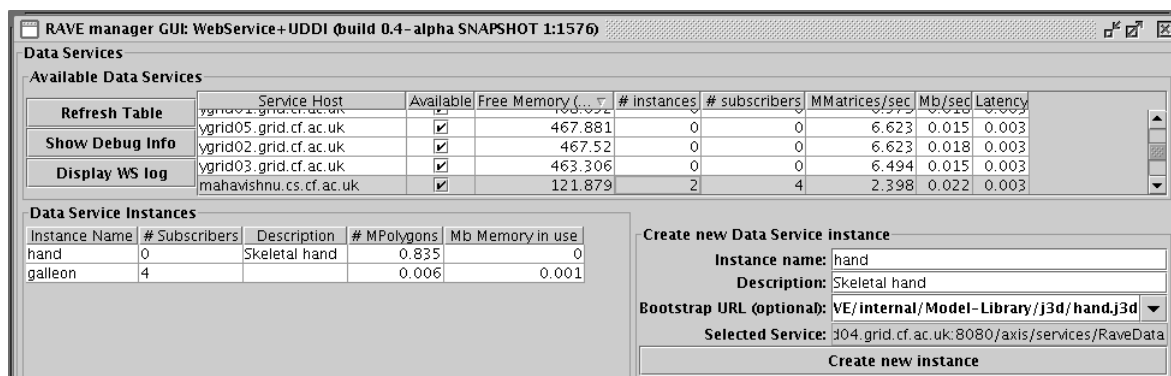


Figure 3: RAVE UDDI manager

4.3 Timings

An 802.11b wireless card was used on the PDA, which contacted the render service on the laptop via a wireless access point. This implementation of the RAVE architecture uses a simple run-length encoding scheme for images, so we are limited by network bandwidth. A 400x400 true colour (24 bits per pixel) image was transmitted, with timings presented in Table 1. The on-screen render speed is shown for an active client as a contrast against that for the thin client. The time taken to render the image at the render server is shown as “Render Time”, and the time to transmit/receive the image as “Image Transmission”. Note that images are continuously streamed to the PDA, forming an interactive environment, rather than a batch-processing model.

It can be seen that the render time is dominated by the image transmission (between 60% and 95%). The peak laptop rendering throughput was around 4.5 million triangles/sec, but the laptop was running at around 10% load. Further investigation showed that Java3D off-screen rendering was significantly slower than that of on-screen rendering (varying between 10 to 30%, depending on the number of triangles in the model being rendered). However, other clients can use the unused resources; the render server is then limited to available network bandwidth to support multiple clients.

Table 1: Benchmarking timings for 400x400 pixel image

Model Name	Number Triangles	Active Client FPS	Thin Client FPS	Image Transmission	Render Time
Skeletal Hand	834,884	5.4	3.2	0.172s	0.134s
Frog Skeleton	663,926	6.3	3.6	0.111s	0.162s
Visible Man Skull	473,281	10.1	1.3	0.625s	0.108s
Human Head	72,701	45.1	2.7	0.342s	0.025s

The image receipt times indicate a logical network bandwidth of between 2.6Mb/s and 0.6Mb/s. A direct socket test between the Zaurus and the main network returns an available bandwidth of 0.6Mb/s, so although our image compression is working, our system is still limited by the available network bandwidth. We need to investigate more efficient encoding schemes for image streaming, as run-length encoding produces little benefit with highly varying images (such as the visible man skull dataset).

Various issues were encountered when using the mobile edition of Java (J2ME, available on the Zaurus), so we were forced to use C++ to implement the PDA client. For further detail refer to [8].

5. Discussion

The current OGSA model [3] of Grid services is changing rapidly, converging with the WS-Resource Framework [13]. This means that service implementations must not become tied to any particular infrastructure, as this is subject to change. We wrap our serving technology inside OGSA, Web Services or a test environment, so our service "engine" is unaffected by changes in infrastructure, only the wrapper needs to be modified.

Our initial framework is a proof of concept; we now have to react to available resources (e.g. network bandwidth, overloaded machines). This includes alternative transmission schemes to cope with lower bandwidth, and automatically switching from an overloaded or faulty service (providing a degree of fail safe operation). We will investigate volume rendering and "bridging" libraries, where we can plug existing applications into RAVE, effectively adapting a single-user application into a collaborative, remote environment.

Many medical imaging applications combine the requirements for high capability computing and data storage with the need for human-in-the-loop supervision. The value of the latest generation of tomographic scanners and digital imaging systems cannot be realised unless appropriate processing and dissemination of the images take place. As such there is a need to move many of the existing batch-oriented services of present Grids to interactive or responsive service provision.

We maintain that the above prototype system represents present state-of-the-art in terms of Grid-enabled remote visualization and provides a platform for further development and refinement for applications such as intraoperative imaging.

References

1. C. Cruz-Neira, D.J.Sandin and T.A. DeFanta. Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *ACM Computer Graphics*, Vol. 27, Number 2, pages 135-142, July 1993.
2. Gordon E. Moore. Cramming more components onto integrated circuits. In *Electronics*, Volume 38, Number 8, April 1965.
3. Ian Foster, Carl Kesselman, Keffrey M. Nick, and Steven Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Globus, February 2002.
4. SGI. SGI OpenGL VizServer 3.1. Data sheet, SGI, March 2003.
5. N. W. John. High Performance Visualization in a Hospital Operating Theatre. In *Proceedings of the Theory and Practice of Computer Graphics (TPCG03)*, pages 170-175. IEEE Computer Society, June 2003.
6. R. F. McCloy and N. W. John. *Computer Assisted Radiology and Surgery (CARS)*, chapter Remote Visualization of Patient Data in the Operating Theatre During Helpato-Pancreatic Surgery, pages 53-58. Elsevier, 2003.
7. High Performance Computing Centre Stuttgart. COVISE Features. <http://www.hlrs.de/organization/vis/covise/features/>, HLRS, September 2000.
8. Ian J. Grimstead, Nick J. Avis and David W. Walker. Automatic Distribution of Rendering Workloads in a Grid Enabled Collaborative Visualization Environment. In *Proceedings of SuperComputing 2004 (SC2004)*, November 2004.
9. K. Brodli, J. Brooke, M.Chen, D. Chisnall, A. Fewings, C. Hughes, N. W. John, M. W. Jones, M. Riding and N. Roard. *Visual Supercomputing – Technologies, Applications and Challenges*. STAR – State of the Art Report, Eurographics, 2004.
10. Organization for the Advancement of Structured Information Standards (OASIS), Universal Description, Discovery and Integration (UDDI) – Version 2 Specification, <http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>, July 2002.
11. The Large Geometric Models Archive. http://www.cc.gatech.edu/projects/large_models/, Georgia Institute of Technology.
12. Visualization ToolKit (VTK), downloadable from <http://www.vtk.org/>
13. Grid and Web Services Standards to Converge. Press release at http://www.marketwire.com/mw/release_html_b1?release_id=61977, GLOBUSWORLD.