

Deploying visualization applications as remote services

M. Riding¹, J.D. Wood² and M.J. Turner¹

¹ University of Manchester, ² University of Leeds

Abstract

In situations where the size of a remotely stored dataset hinders its transmission over public networks, it becomes feasible to visualize using hardware resources local to the data rather than the user. A difficulty with this approach is that the visualization user often has no control over the software resources available on remote visualization machines. This paper describes a turnkey visualization system that provides a common user interface to a range of visualization applications running as services on remote hardware resources. It describes the motivations for designing such a system, the challenges faced in its construction, and the mechanism by which third party developers can integrate their own visualization applications.

1. Introduction

Continuing advances in the development of computational and graphical hardware have greatly increased the visualization capability of modern machines, from desktop PCs to specialist workstations and dedicated graphics clusters. This has been accompanied by ongoing software development effort to both develop new visualization techniques and algorithms, and to optimise existing applications in order to harness the power offered by the latest hardware. Together these efforts have enabled the visualization of ever larger datasets and now, with high performance computing machines rapidly approaching the petascale level and simulation scales increasing in size accordingly, such datasets are abundant.

One such example is provided by the TeraShake software from the South California Earthquake Center [ODM*06] which is capable of generating terabytes of data. A time dependent dataset from this simulation, downsized by a factor of 64, was used as the basis of the 2006 IEEE Visualization Design Contest. The winning entry visualized the resulting 75Gb dataset in real time on a single machine; a dual core 2.2GHz Athlon XP processor, with 2Gb of RAM, a GeForce 7900GTX graphic card with 512Mb of RAM and a single IDE hard disk. Such machines are, or soon will be, commonplace throughout the visualization community, both as individual nodes in graphics clusters and as desktop workstations.

Network speeds however, have not been increasing at the same rate. Though most users will have access to hardware with the necessary processing power to visualize large datasets, not all will have sufficient network bandwidth to

quickly access the data if it is stored remotely. Such a situation might lead to the adoption of an 'owner computes' approach to visualization, where large datasets are visualized on hardware as close as possible to their storage location in order to minimise the transfer of data over public networks. In this case, the output from the visualization pipeline (a stream of rendered images) is transferred to the user's desktop machine, and changes to pipeline parameters are communicated back. This approach is feasible in situations where the network bandwidth is sufficient for the interactive transfer of rendered images, but insufficient to quickly transfer the dataset to the client prior to visualization.

Technologies such as VNC (Virtual Network Computing), and commercial offerings such as SGI's VizServer and HP's Remote Graphics Software (RGS) support this method of operation, but require that the end user be familiar with both the server side operating system, and the particular visualization software installation. The emergence of the Grid computing paradigm provides an alternative solution, in which a single desktop application can seamlessly access and interact with remote visualization hardware and software.

In this paper we describe the issues involved in the creation of such a system, in which individual visualization applications are encapsulated as services that can be selected and remotely instantiated from within a thin-client. We begin with a brief overview of related work, before introducing in Section 3 the concept of a turnkey visualization system for the Grid, and describing the challenges involved in its creation in Sections 4 to 6. We then provide an example of such a system in operation in Section 7, where we consider the visualization of large volume datasets.

2. Related Work

The Grid Visualization System (GVS) component of the National Research Grid Initiative (NAREGI) project [KNT*04] provides an API that can be used to encapsulate visualization applications into remote services. Details are not provided of how the services can then be combined to form a fully functioning visualization application.

The e-Demand project [CHM03] [CHM04] describes an architecture for Grid visualization where an individual visualization operation is considered to be a service, rather than a whole application. In this system, pipeline components can be distributed at runtime across different machines. A similar approach is taken by the developers of the NoCoV (Notification-service-based Collaborative Visualization) [WBHW06] system, which extends the concept of a visualization service to include WS-Notification.

The Resource Aware Visualization Environment (RAVE) project [GAPW05] is a Java application providing a remote visualization service that can employ either server side rendering, client side rendering, or a combination of the two.

In our earlier paper [RWB*05] we discussed the concept of an abstract framework for Grid based visualizations, highlighted the benefits to end-users and described a prototype implementation. We now build on this work, and introduce techniques to bridge the gap between a demonstrator application and an extensible system for visualization on the Grid.

3. A Meta Visualization System

Modular visualization environments (MVEs) allow the construction of complex visualization pipelines from a library of reusable components. They are generally simple enough to use that new users from outside the domain of scientific visualization can construct powerful applications after only a small amount of training. The creation of efficient pipelines, however, remains a complex task, requiring in depth knowledge of the techniques involved, and of the internal architecture of the chosen implementation software and hardware resources. The distribution of pipelines over Grid resources introduces further complications, not only in terms of bottlenecks to system performance, but also in configuration issues relating to system security policies such as firewalls, for example.

In circumstances such as these, there is an advantage in creating pre-configured and optimised visualization pipelines for end users, either by saving configurations of MVEs, or by constructing turnkey visualization applications. This makes a distinction between the roles of visualization developer and visualization user, and is the approach we have chosen to pursue in our work.

Turnkey visualization applications such as MicroAVS and ParaView [LHA01] [CGM*06] simplify the process of creating visualizations by presenting users with a selection of preconfigured pipelines appropriate for use with the chosen input data. The user does not need to know how to construct a pipeline from a collection of smaller modules, but

can quickly experiment with the effects of each technique when used to visualize his or her data. As shown in Figure 1, a visualization session begins with the user selecting an input dataset (1). The system then identifies visualization techniques suitable for use with the data (2). The user then selects a particular technique that they would like to use with their data (3), and the system constructs a visualization pipeline to implement that technique (4). The user can then begin interacting with their data, modifying pipeline parameters as desired (5). After trying a particular technique, users might select a new pipeline from the initial offering (6), or load a new dataset to start a new visualization session (7).

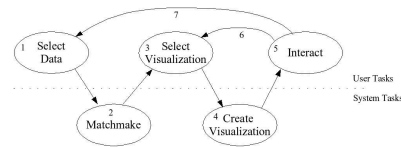


Figure 1: Tasks in the creation of a visualization session with a turnkey visualization application

A key difference between turnkey and standard visualization applications is that the matchmaking and pipeline construction steps are performed by the visualization application itself (based on information provided by the visualization developer), rather than the user. In addition to the standard visualization components of a graphical user interface and an underlying visualization pipeline, a turnkey visualization system therefore includes a matchmaking component.

It is typically the case that the user interface to the visualization application is closely coupled to the underlying visualization system. MicroAVS for instance, relies on the capabilities of AVS/Express, and ParaView similarly utilises the Visualization Toolkit (VTK). There is currently no mechanism to allow MicroAVS to use VTK to provide its visualization, or for ParaView to use AVS/Express. Furthermore turnkey visualization systems may run in a non-distributed manner, with the display machine also used to execute the pipeline. ParaView is a notable exception to this situation, and has been designed from the ground up to support remote data processing and rendering.

In an 'owner computes' situation, where we aim to visualize remote data using remote hardware, we would like to resolve these difficulties in order to create a turnkey visualization system that provides a common user interface to a number of different remote visualization services. There are three main challenges involved: matchmaking - to identify the visualization techniques suitable for use with a particular dataset on a particular machine; abstraction - the creation of a common user interface to each remote visualization application; and extensibility - to create a mechanism by which third party developers can add their own visualizations into the system. We now assess each of these challenges in turn.

4. Matchmaking

A turnkey visualization system will have a number of pre-built pipelines ready to be used with certain types of input data. The process of matchmaking determines which pipeline can be used with which type of input data. Ordinarily a simple mapping based on the data storage format and number of dependent and independent variables would be sufficient to represent this relationship. In our distributed system, there is an extra complication since in addition to mapping between the input data and the visualization pipeline, we must also map between visualization pipelines and implementation software instances, and again onto hardware resources. This is further constrained by the access rights of individual users to particular machines and software applications, as well as the limitations of spare capacity on the target machine.

This information can be represented as a directed acyclic graph, as shown in Figure 2. Data input type instances form the top level of the graph, and are then related to visualization techniques. Each technique is related to those visualization application instances that provide an implementation pipeline. Note that a particular visualization application may be capable of implementing more than one visualization technique. Applications are then related to the hardware instances on which they can run. Again, each machine may be able to run more than one type of visualization application. Individual users may only have license rights to certain software instances and accounts on particular machines, as indicated by the inverted colours in the figure. Similarly, machines may not have any spare capacity for a new user process. The process of matchmaking then involves identifying the techniques suitable for use with a particular dataset, but with additional checks to ensure that hardware and software resources exist to provide an implementation, that the particular user has access to those machines, and that there is sufficient spare capacity to support the visualization job.

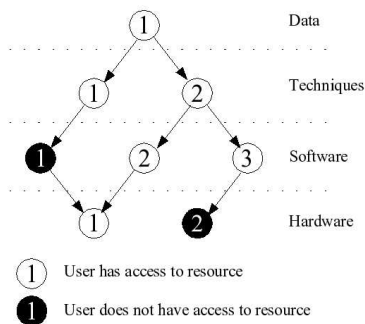


Figure 2: Levels of abstraction in matchmaking

We can imagine a simple example of this matchmaking strategy by considering the nodes in the graph to represent the following:

- Data Input: a raw binary volume

- Techniques: isosurfacing (1) and direct volume rendering (2)
- Software: a VTK isosurfacing application (1), a software ray casting volume renderer (2), a GPU volume rendering application (3)
- Hardware: an SGI Prism (1), a GPU equipped cluster (2)

In this instance, the user only has access to software instances 2 and 3, and to hardware instances 1 and 2. However, the GPU equipped cluster has no spare resource, and so is not eligible to run new jobs. Thus the only candidate technique to be returned by the matchmaker is technique 2 - direct volume rendering - which can only be implemented using the software ray casting volume renderer, running on the SGI Prism.

We have used a relational database, implemented using PostgreSQL to implement this system. Tables hold information on machines, applications, pipelines, data types and users, as well as the relationships between them. Matchmaking queries are then provided to identify candidate visualization pipelines for particular datasets for particular users. No attempt is made to rate pipelines for suitability, only the filtering of implementable pipelines from non-implementable pipelines. Due to time constraints, the system currently makes no effort to determine spare capacity on remote resources, and adopts an optimistic approach, assuming the machines to always have sufficient capacity to launch a job. The authors acknowledge this limitation, and look to efforts in the scheduling and resource brokering communities to provide a standardised resolution.

Since we are describing a distributed system with multiple users at multiple sites, the matchmaking service must be centralised. This enables any updates to the database to be immediately available to all system users without the need for a software update. This is achieved by embedding the database queries into a web service. The system front-end then interrogates the database through the web service interface and displays the results to user. We have implemented the web service in WSRF::Lite [BMPZ05], a Perl implementation of the WSRF standard.

5. Abstraction

A graphical user interface to a visualization pipeline performs two tasks: modification of pipeline parameters, and the display of the pipeline output. When visualizing remotely the situation is no different. We provide this functionality through two separate components; an abstract user interface that adapts to expose the parameters of the underlying visualization application, and an interactive viewer component that displays the images output from the pipeline.

5.1. Abstract User Interface

The abstract user interface is discussed in detail in a separate paper (submitted to the 2007 UK e-Science All Hands Meeting), so only a functional overview here is provided here. In brief, it provides a set of widgets to control pipeline parameters, selecting which to display based on a pipeline definition

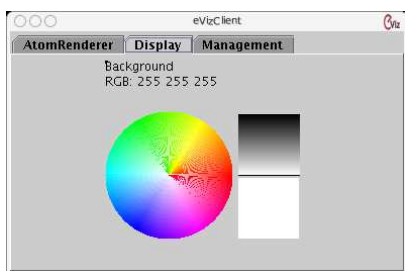


Figure 3: Adaptive user interface, showing a colour wheel used to set the background colour for a molecular visualization

stored in an XML configuration file. The configuration file adheres to the skML language developed as part of the gViz project [DS05]. Each module in the pipeline is represented as a tab in the GUI, with a widget being assigned to represent each parameter of that module. Default widgets exist for basic types (text boxes for strings, sliders for bounded integers and floats, drop down lists for enumerations etc.). More complex widgets exist for standard visualization interfaces such as colour and transparency transfer functions and colour wheels, as shown in Figure 3. Additionally, users can provide their own widget plugins to override system defaults at runtime. The abstract user interface is implemented in Java.

5.2. Interactive Viewer

The interactive viewer displays remotely rendered images and transforms user interactions into remote camera parameter events. Multiple visualizations are supported through the use of tabs, as shown in Figure 5. User input can be sent to the visualization shown in the active tab, or to all tabs at once, allowing different visualization techniques to be used to explore a dataset simultaneously. Alternatively, comparisons can be made of different implementations of the same visualization technique. Visualization outputs in different tabs can also be combined, to support for example, stereo visualizations with each eye being rendered on a separate machine.

In situations where the same dataset is being visualized using different techniques implemented by different applications, care must be taken to maintain consistency of viewpoint. It is necessary to ensure that a rotation of ninety degrees along the data's y axis, for example, is correctly implemented by each application. To resolve this, an abstract camera model is employed based on the axis aligned bounding box of the original dataset. In this model, we define a left handed coordinate system based on the bounding box, with the origin in the centre of the box. We then define a distance unit to be the length of the longest box half height. The initial camera viewpoint coordinate is then set at (0,0,-5), looking along the z-axis (0,0,1), with an up vector of (0,1,0). Absolute camera positions are then calculated and transmitted from the viewer at each user generated event. It is then

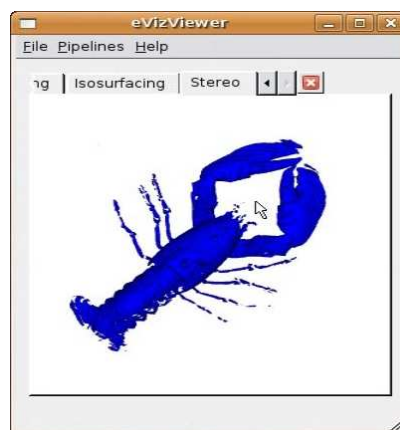


Figure 4: Interactive viewer, showing multiple tabbed visualizations combined to form a stereo pair (limitations in the screen capture technology cause only the image for one eye to be shown in print)

the job of the server application to convert back from this coordinate system. The interactive viewer application is implemented using QT and OpenGL.

5.3. Coordination

User interface applications are supported through a centralised web service representing the state of the visualization session. Each session can incorporate multiple visualization pipelines, and each pipeline can optionally be implemented by multiple redundant servers for reasons of fault tolerance. This information, together with the pipeline skML is stored in the session web service. Hardware resources register themselves with the web service when they initialise, providing a resource discovery mechanism for the client applications.

6. Extensibility

It is important to have a mechanism by which new visualization pipelines can be added to the system. The main obstacle to be overcome is in defining an abstract interface to visualization applications. This is achieved through the provision of a middleware library encapsulating the functionality of remote parameter modification (computational steering) and the transmission of pipeline output images (remote rendering). The high level design aims of the library are for a visualization to support the following features:

- Multiple users. A visualization should be able to be viewed and controlled by multiple users across multiple sites
- Redundant servers. A visualization should be capable of being run simultaneously on redundant servers for reasons of fault tolerance
- Non-blocking operation. The library should not interfere with the threading or event models employed in the visualization application

- Server push operation. A visualization should be able to respond to an externally generated event, such as the arrival of new data from a simulation, and to automatically push updated images out to all connected clients. Clients should also be able to request updates from the server, but in contrast to a full client-pull implementation, the communication channels remain open during the course of the session

A client API is provided to interface with the graphical user interface components described in Section 5. A server API interfaces with the visualization applications themselves. A brief overview of the functionality of each follows:

6.1. Server API

The server side API provides functions to accomplish the following:

- create and initialise a data structure to hold the library state
- register parameters and visual outputs
- begin a visualization session
- get parameter values from the client
- transmit an image to all connected clients
- process a request for an update (a callback function)
- finalise and destroy the data structure holding the library state

6.2. Client API

The client side API provides functions to accomplish the following (really a complement of the server side functions):

- create and initialise a data structure to hold the library state
- begin a visualization session
- get current parameter values from the server
- set new parameters values
- send a request for a new frame
- receive a new frame (a callback function)
- finalise and destroy the data structure holding the library state

6.3. Library implementation

The library is implemented in C and so can be used directly with both C and C++ applications. Bindings to other languages are not provided, since the majority of visualization applications are written in C or C++, (although commodity tools exist that would allow the creation of wrappers for languages such as Java, Python or Perl). Both the client and server libraries create their own control threads in order to meet the requirement for non-blocking operation previously identified.

We used the gViz computational steering library [BDG*04] to control the parameters of the visualization pipeline. Within our implementation, the interface to this library is abstract enough that alternative implementations, such as that provided by the RealityGrid project [PHPP04], could be used instead.

We have implemented our own library to handle the server side compression, transmission and client side decompression of image data. This library provides a number of different image compression codecs, including colour cell compression (CCC) [CDF*86], JPEG, PNG and runtime length encoding of difference images, and attempts to choose at runtime the most suitable codec for minimising transmission times. Each technique offers different compression ratios and processing and transmission times depending on the image size, image complexity, network bandwidth and client and server CPU loading.

In order to avoid problems with client side firewalls, the library opens sockets on server machines only. Since this approach may still be hampered by server side firewalls, the choice of port for individual sockets can be chosen at runtime through the use of environment variables. Other than this, the library hides the details of networking and data transmission code from the developer.

Three sockets are opened by the library. One for the gViz computational steering library, one for the transmission of rendered images, and a third used to transfer a regular pulse from server to client. This allows the client to quickly become aware of a server side software failure or loss of network connectivity, and is useful for providing runtime fault tolerance through redundant servers.

The process of integrating a new application with our system involves three tasks: instrumentation of the application with the server API described above, documentation of the pipeline functionality to enable both remote steering and automatic GUI construction, and registration with the match-making system.

6.4. Instrumentation

The first step to be undertaken when instrumenting a new application is to define a render callback function and register it with the library through the server API. This provides a mechanism for the library to request new frames from the server whenever necessary. Since the function callback is executed from a thread within the library, care must be taken to prevent simultaneous access to the visualization pipeline. The function must perform a render operation and then provide the library with a pointer to the image in memory so that it can be compressed and transmitted to clients.

The next step is to modify the event loop of the visualization application so that it checks the status of any steered parameters registered with the library. If any updates have occurred, they must be fed into the visualization pipeline, a render performed, and the image then passed to the library as with the render callback.

6.5. Documentation

A visualization pipeline must be accompanied by a skML document describing the modules, parameters and hardware resources involved in its implementation. This information is used by the adaptive user interface in order to construct a

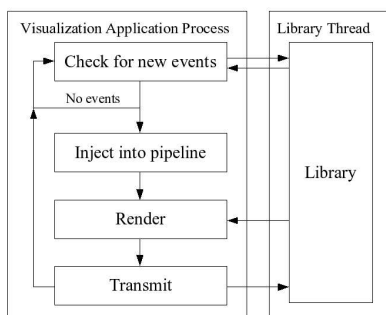


Figure 5: Threading model, depicting changes that must be made the visualization application event loop

GUI with widgets representing each of the steered parameters.

Each parameter is described by a unique name, an indication of the data type and of the read/write permissions (whether we can modify a parameter, or just view it), and optionally, minimum and maximum values. Supported data types are scalar and vector instances of long integers, real floating point values and strings;

Parameters can be grouped together to form modules, which are also assigned a unique name. Ideally, the parameter naming scheme would be based on an ontology of visualization terms. This would allow different developers to independently create an identical skML description of the same visualization technique implemented with different applications. Unfortunately no such ontology currently exists, and so the potential remains for functionality identical visualizations to be represented by different skML files, and therefore different user interfaces. This lack of consistency may be confusing for users.

6.6. Registration

Once a new visualization application has been instrumented and documented, it must be registered with the matchmaking system so that it can be recommended to users as a candidate pipeline. Registration involves detailing the acceptable input data format, the visualization technique implemented, the software itself, as well as the machines on which it is installed, and finally user access rights. A client application is provided to allow this information to be entered through a set of 'wizard' style input dialogues 6.

The final registration task is to integrate the visualization application with the system launch framework. Jobs are launched via a Java CoG kit, which executes a wrapper script on the server resource. The same wrapper script is used for each target visualization application, and performs the tasks of setting up the execution environment, and launching the job with the correct command line arguments. This allows machine specific environment details (such as library paths) to be configured separately for each target machine, greatly

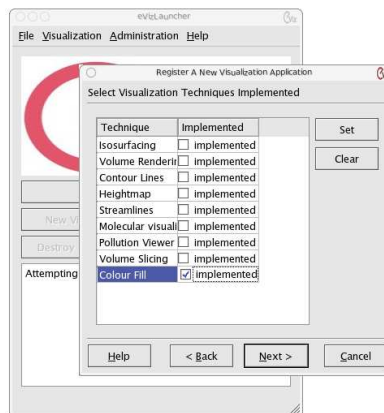


Figure 6: Wizard style application used to update the system database.

simplifying the number of arguments that must be passed via RSL.

7. Usage

We have used the server component of the API described in Section 6 to instrument a number of open source visualization applications and toolkits, including VTK, VMD [HDS96], the Real Time Ray Tracer (RTRT) [Par02], and ParaView. Supported visualization techniques include volume rendering (ray casting), isosurfacing (marching cubes), molecular visualization (numerous techniques) and cut plane interrogation of volumes. Due to time constraints, not all of the functionality of each application has been exposed through the API.

It was found that the main obstacle to the instrumentation of new applications was the difficulty in developing a clear understanding of the programming model of the visualization application in question, especially in larger applications such as ParaView. The threading model of the target visualization systems is of particular importance, especially since our implementation relies on callback functions executed from within a separate thread. The threading requirements of the X11 library must also be adhered to on Unix like systems, (access to X11 system functions must be serialised). Another concern was the time taken to expose a complete set of visualization parameters.

We now provide details of the integration and subsequent use of a ParaView visualization pipeline with our system.

7.1. Volume Visualization on a Cluster with ParaView

To illustrate a potential use of our system, we consider the difficulties faced in attempting to visualize a large volume dataset. We base this scenario on experiences with material scientists wishing to visualize the output of tomographic scanners; datasets 10s of gigabytes in size. We created a synthetic volume dataset by upscaling the visible human female

CT scan by a factor of 8. This yields a volume of dimensions $1024 \times 1024 \times 3468$, and a total size of 7.3Gb. The data was stored on the storage network forming part of the North West Grid (NW-GRID) at the University of Manchester. The transfer of such a volume of data over a public network will be a timely operation, and so at the very least the data read component of a visualization pipeline should be executed on a machine local to the data.

We begin by considering the manual process of creating such a visualization. Our aim is to visualize the data with cut planes and isosurfaces.

ParaView was chosen as the visualization software resource, since it is specifically designed to work in parallel and so will make good use of a hardware cluster. ParaView can be configured to run in a number of different modes each with a different degree of distribution. The simplest mode of operation involves running the entire application on the same machine, which performs the tasks of data processing, rendering and display. This requires that the user have physical access to the display of the machine in question. An alternative mode of operation is to allow the data processing and rendering tasks to be performed on a remote machine, with the output displayed on the user's own desktop. A final mode extends this model further to allow separate remote machines to be used for both data processing and rendering. Each approach allows the use of parallel processing through MPI. In our case, we do not have physical access to the compute resource, discounting the integrated mode of operation. Experimentation proved the fully distributed mode to be inappropriate also, since individual nodes in the computer cluster do not have direct network to the external networks. This means all network traffic must be routed through the head node, introducing a significant bottleneck. Our only feasible option is to use the cluster to perform both data processing and rendering. This is complicated further by the fact that the cluster has no graphics hardware, and so we have to resort to software rendering.

We chose to use an NW-GRID cluster machine, 'man2', which offers 48 cores for parallel jobs, each with 2Gb of memory.

Having now identified our hardware and software resources, we still need to configure ParaView. As already stated, our NW-GRID cluster machine is only accessible through the head node, yet our ParaView job runs exclusively on back-end nodes without network access. Since there is no port forwarding software installed on the cluster, we then need to tunnel network connections from back end nodes through the head nodes to the external network. Because we have no control over which particular back end nodes our job runs on, we must create our network tunnels dynamically.

Only at this point are we able to start using the ParaView software to visualize the input data. The configuration process is involved, requiring knowledge of the architecture of both the hardware and software resources, coupled with the skills necessary to circumvent firewall restrictions. By taking the step of integrating ParaView with our system, we can

enable end users to visualize their (large) data, but without them having to learn the configuration step, or have a visualization developer on hand to do it for them.

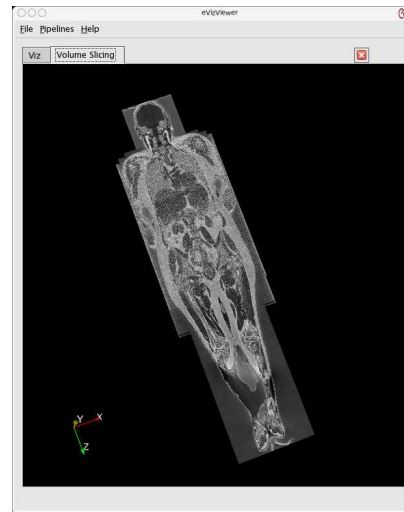


Figure 7: *e-Viz integrated with ParaView showing a cut plane through the visible human female dataset.*

Figure 7 shows our system in use with ParaView as the remote visualization service provider. A cut plane through the 7.3Gb dataset is depicted. Slice extraction was found to take round 10 seconds when using 48 processors. A framerate of approximately 1.5 fps was achieved when rendering a single slice through the coronal plane on 48 nodes. This slightly disappointing result is caused by ParaView rendering each pixel as a separate quadrilateral, yielding nearly 8 million triangles. An additional pre-rendering process to convert to a small number of textured triangles would undoubtedly improve performance.

8. Limitations

The most significant limitation of the current system is the lack of an underlying distributed file system. When visualizing remote data, a mechanism must be provided in order to discover and reference the input files and datasets. This could either be through the use of a Grid file system, an SRB, or by interrogating datasets that expose a machine readable interface.

A secondary limitation is the lack of an underlying ontology of visualization terms. As discussed earlier, this would provide a pipeline parameter naming strategy, which would ensure that functionality identical pipelines are represented by identical interfaces, regardless of the implementation software. Without an ontology it is impossible to guarantee that users will see a consistent graphical user interface to remote applications. This limitation is unlikely to be resolved without the creation and visualization community wide adoption of an ontology of visualization terms, though there is research in this direction [SAR06].

Further limitations exist in the brokering aspect of the matchmaking service. There is currently no provision in our system for determining the spare capacity of target hardware resources, though this is a problem addressed by other work in the community. Similarly, there is no mechanism for determining the degree of parallelism required to achieve interactive frame rates for a particular dataset and visualization technique. This is a research topic within our project, and will be addressed in a forthcoming publication.

9. Conclusions

We have introduced a system for the deployment of visualization applications as remote services within a turnkey application. End users are assisted in the process of creating visualizations running on Grid resources by matchmaker and job staging processes. Abstraction is provided through the use of an adaptive user interface. By integrating their software with our API, developers of visualization applications can benefit from a framework for deploying applications onto Grid resources, support for multiple users and an automatically generated user interface running on multiple platforms.

We recognise limitations in the lack of an underlying distributed file system and visualization ontology, as well as the need for a more sophisticated brokering strategy. It is hoped that future work will address these issues.

10. Acknowledgements

Financial support for this work was provided by the Engineering and Physical Sciences Research Council through grant numbers GR/S46567/01, GR/S46574/01 & GR/S46581/01.

References

- [BDG*04] BRODLIE K., DUCE D., GALLOP J., SAGAR M., J. WALTON, WOOD J.: Visualization in grid computing environments. In *IEEE Visualization* (2004), IEEE Computer Society, pp. 155–162.
- [BMPZ05] BROOKE J., MCKEOWN M., PICKLES S., ZASADA S.: Implementing ws-security in perl. In *Proceedings of the UK e-Science All Hands Conference, Nottingham* (2005).
- [CDF*86] CAMPBELL G., DEFANTI T. A., FREDERIKSEN J., JOYCE S. A., LESKE L. A.: Two bit/pixel full color encoding. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1986), ACM Press, pp. 215–223.
- [CGM*06] CEDILNIK A., GEVECI B., MORELAND K., AHRENS J., FAVRE J.: Remote large data visualization in the paraview framework. In *Proceedings of the Eurographics Parallel Graphics and Visualization* (2006), pp. 162–170.
- [CHM03] CHARTERS S. M., HOLLIMAN N. S., MUNRO M.: Visualisation in e-demand: A grid service architecture for stereoscopic visualisation. In *paper presented at the UK e-Science All Hands Conference, Nottingham* (2003).
- [CHM04] CHARTERS S., HOLLIMAN N., MUNRO M.: Visualization on the grid: A web services approach. In *paper presented at the UK e-Science All Hands Conference, Nottingham* (2004).
- [DS05] DUCE D., SAGAR M.: skml: A markup language for distributed collaborative visualization. In *Proceedings of Theory and Practice of Computer Graphics* (2005), pp. 171–178.
- [GAPW05] GRIMSTEAD I., AVIS N., PHILP R., WALKER D.: Resource-aware visualization using web services. In *Proceedings of the UK e-Science All Hands Conference, Nottingham* (2005).
- [HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics* 14 (1996), 33–38.
- [KNT*04] KLEIJER P., NAKANO E., TAKEI T., TAKAHARA H., YOSHIDA A.: Api for grid based visualization systems. In *Workshop on Grid Application Programming Interfaces in conjunction with GGF12, Brussels, Belgium* (20 Sept. 2004).
- [LHA01] LAW C. C., HENDERSON A., AHRENS J.: An application architecture for large data visualization: a case study. In *PVG '01: Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics* (Piscataway, NJ, USA, 2001), IEEE Press, pp. 125–128.
- [ODM*06] OLSEN K., DAY S., MINSTER J. B., CUI Y., CHOURASIA A., MOORE R., HU Y., ZHU J., MAECHLING P., JORDAN T.: Scec terashake simulations: High resolution simulations of large southern san andreas earthquakes using the teragrid. In *Proceedings of the TeraGrid Conference* (2006).
- [Par02] PARKER S.: Interactive ray tracing on a supercomputer. In *Practical Parallel Rendering* (2002).
- [PHPP04] PICKLES S. M., HAINES R., PINNING R. L., PORTER A. R.: A practical toolkit for computational steering. *Philosophical Transactions of the Royal Society* (2004).
- [RWB*05] RIDING M., WOOD J., BRODLIE K., BROOKE J., CHEN M., CHISNALL D., HUGHES C., JOHN N., JONES M., ROARD N.: e-viz: Towards an integrated framework for high performance visualization. In *Proceedings of the UK e-Science All Hands Conference, Nottingham* (2005).
- [SAR06] SHU G., AVIS N., RANA O.: Investigating visualization ontologies. In *Proceedings of the UK e-Science All Hands Conference, Nottingham* (2006).
- [WBHW06] WANG H., BRODLIE K., HANDLEY J., WOOD J.: Service-oriented approach to collaborative visualization. In *Proceedings of the UK e-Science All Hands Conference, Nottingham* (2006).