

# Resource-Aware Visualization Using Web Services

Ian J. Grimstead, Nick J. Avis and David W. Walker  
Cardiff School of Computer Science, Cardiff University  
Roger N. Philp

Cardiff School of Physics and Astronomy, Cardiff University

**Abstract**—We present a status report on RAVE, our distributed, collaborative grid enabled visualization environment. We briefly review our architecture and related work, examine our collaborative support and relate this to an experiment carried out between SuperComputing 2004 (Pittsburgh, PA, USA) and the Cardiff School of Computer Science. Load distribution in RAVE is described and analysed, using a tiled rendering technique to share rendering workload between machines. Finally, we review various applications that have been extended to use RAVE.

## I. INTRODUCTION

Increases in network speed and connectivity are promoting the use of remote resources via grid computing, based on service-oriented architectures such as the Open Grid Services Architecture (OGSA) [9]. These permit users to remotely access heterogeneous resources without considering their underlying implementations, both simplifying access for users and promoting the sharing of specialised equipment.

With datasets rapidly increasing in size, the visualization of such datasets can quickly overwhelm local computing power. The availability of Grid computing enables users to recruit resources to supply datasets or to assist in their rendering. Grid computing enables co-operation between remote teams at interactive rates and becomes more desirable as network technology improves.

With this in mind, we present the Resource-Aware Visualization Environment (RAVE), using Grid/Web Services to advertise data sources and recruit rendering assistance from remote resources.

## II. PREVIOUS WORK

Current collaborative visualization systems often make assumptions about the available resources; for instance, COVISE [24] (a collaborative modular visualization package) assumes local rendering support, whilst OpenGL VizServer [18] (collaborative sharing of OpenGL programs) assumes the client has modest rendering capability and instead relies

on remote resources. The ARTE environment [14] presents a hybrid approach whereby a full bitmap or geometry may be transmitted, but runs as a single server on a single platform and does not make use of remote resources. gIX [23] is a method for rendering data stored remotely over X11, sending unprocessed primitives—requiring both high network bandwidth and local rendering.

MVEs and Problem Solving Environments (PSEs) are popular tools with visualization, with several projects using this approach. The e-Demand project [5] is implementing a PSE on the Grid, where each module in the environment is represented by an OGSA service, whilst the gViz project [4] is extending IRIS Explorer [22] to be grid-enabled and collaborative (where users can independently control each module of the MVE).

For a fuller review of remote visualization applications refer to [3] and [11].

The RAVE system differs from such systems by making best use of available local or remote resources and reacting to changes in these resources. In addition, RAVE provides a collaborative environment, a data repository and connects to 3rd party data.

## III. RAVE ARCHITECTURE

We propose a novel and unique visualization system that will respond to available heterogeneous 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, and distributing workload amongst resources. The RAVE architecture presented in Figure 1, which we now discuss. An in depth review of the RAVE architecture may be found in our SC2004 paper [11].

### A. Data Service

The data service imports data from either a static file or a live feed from an external program, either

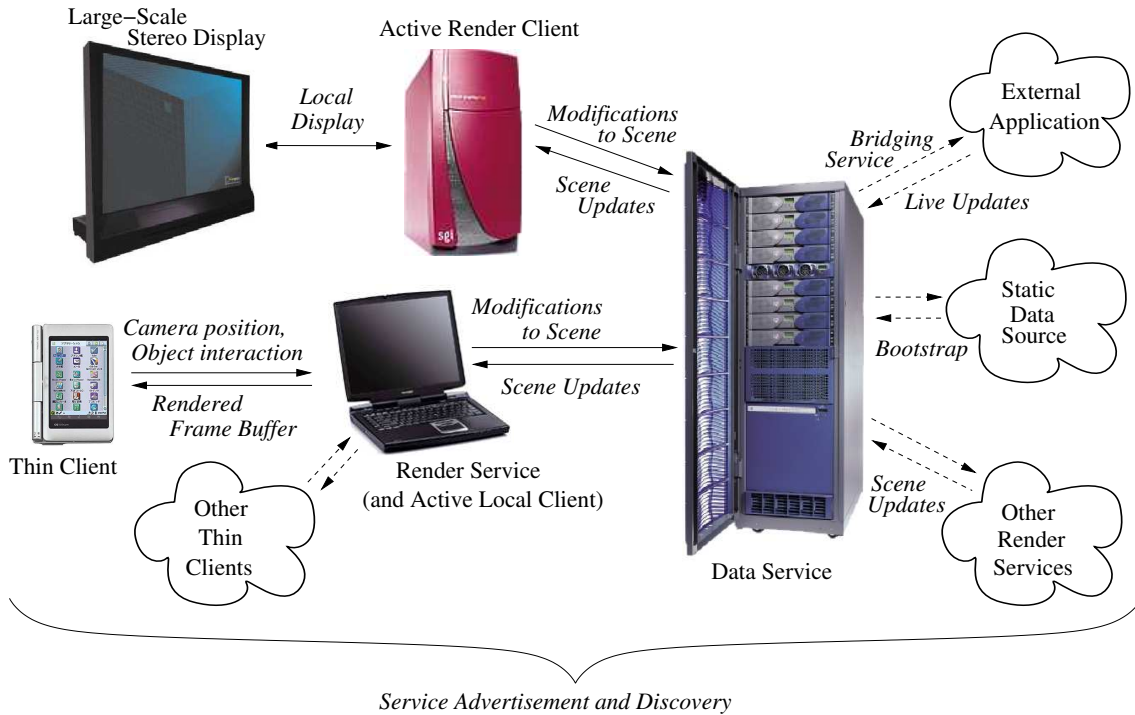


Fig. 1. Diagram of basic RAVE architecture

of which may be local or remotely hosted. 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 that are made 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.

The data are stored in the form of a scene tree; nodes of the tree may contain various types of data, such as voxels, point clouds or polygons. This enables us to support different data formats for visualization.

### B. Active Client

An active client is a machine that has a graphics processor and is capable of rendering the dataset. The active client connects to the data service and requests a copy of (or a subset of) the latest data, bootstrapping the client. The client maintains a connection to the data service, receiving any changes made to the data by remote users whilst sending any local changes made by the local user, who interacts with the locally rendered dataset (Figure 2. Note that only changes are sent—this reduces the bandwidth required, akin to COVISE [24].

The selection of a subset of the data can enable a client with insufficient rendering power to render a lower resolution Level Of Detail (LOD), as used in ARTE [14] and MUVEES [6], or to render just the area of interest.

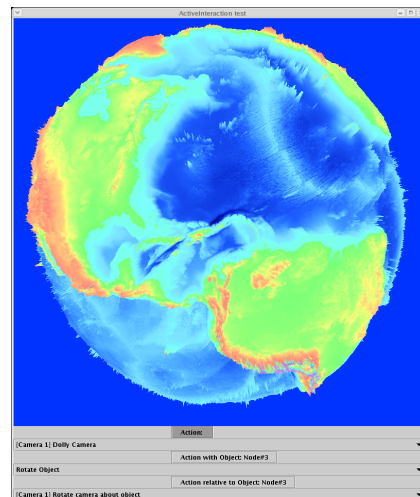


Fig. 2. Screenshot of ETOPO [15] dataset from RAVE (64-bit Linux)

### C. Render Service

Render services operate in a similar manner as an active client, except these do not render to the

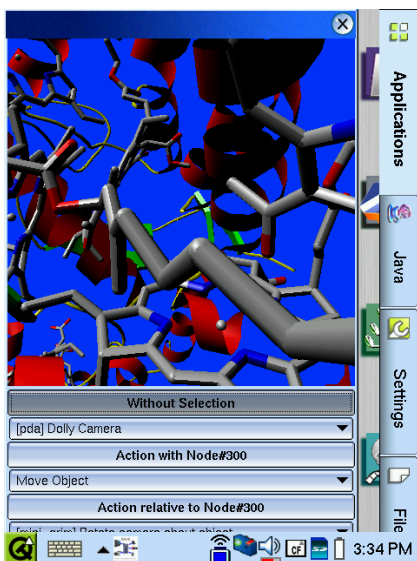


Fig. 3. Screenshot of Molecule 1PRC [8] (VRML format) from RAVE (Sharp Zaurus PDA)

local console, instead operating in the background. If a local client does not have sufficient resources to render the data, a render service can perform the rendering instead and send the rendered frame over the network to the client for display. 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 single copies of the data are stored in the render service to save resources.

The host console is not visually affected when running a render service (the only side effect being a slow-down in on-screen rendering as the GPU is being shared). A specialised graphics card can then be used simultaneously by multiple users (local and remote), promoting the sharing of graphical resources. This is in contrast to other systems, such as VizServer [18], which may take over the host console to perform rendering.

A render service may be requested to render a subset of the scene tree or frame buffer; this enables several render services to render a dataset that would otherwise overwhelm an individual service. The subset is rendered locally and the resulting framebuffer is then transmitted to another render service or active client, where the results are composited. Compositing is currently restricted to opaque solids, as this does not require any specific ordering of frame buffers at composition.

#### D. Thin Client

Whereas an active client is a render-capable client for the data service, a thin client (such as a PDA, as the screenshot presented in Figure 3)

represents a client that has no or very modest local rendering resources. The thin client must make use of remote rendering, so it connects to the render service and requests rendered copies of the data. The local user can still manipulate the camera view point and the underlying data, but the actual processing and rendering transformations are carried out remotely whilst the local client only deals with information presentation. OpenGL VizServer [18] works in a similar manner, only our implementation is totally heterogeneous.

## IV. COLLABORATION

RAVE is designed as a collaborative environment; we review its facilities and discuss a collaborative experiment carried out at SC2004.

### A. Supported Collaboration

RAVE implements an underlying shared scene-graph, with the data service receiving updates from active clients and render servers, reflecting such updates to all connected parties. This enables multiple users to view the same dataset collaboratively, as if one users modifies the data, the modification will be reflected to all connected users, regardless of platform or client.

Users are represented in the dataset by Avatars, and are able navigate around the dataset and alter it at will. The avatar is tied to the user's camera position, so when the user changes their view, their related avatar is also updated (and reflected to all other parties via the data service).

We are current assuming trust between users, although we aim to implement a more secure approach later in the project. This means that any user can modify any part of the data, without restriction. This includes the ability to modify another party's camera; this could be a problem with untrusted users, but enables one client to control another. An example is using a PDA to steer a large-scale display (such as a CAVE or presentation projector), the PDA acting as a wireless remote-control of a RAVE client.

An example of a successful collaboration, including remote steering of a large-scale display, is presented in the next section.

### B. Trans-Atlantic Collaboration

The Resource-Aware Visualization Environment enabled a PDA to visualize large, shared datasets (in the range of 0.5-4.5 million polygons) stored in Cardiff, interacting with a laptop (rendering the data locally). This was reported in depth at the Web3D 2005 conference [10], so we present a brief overview here.

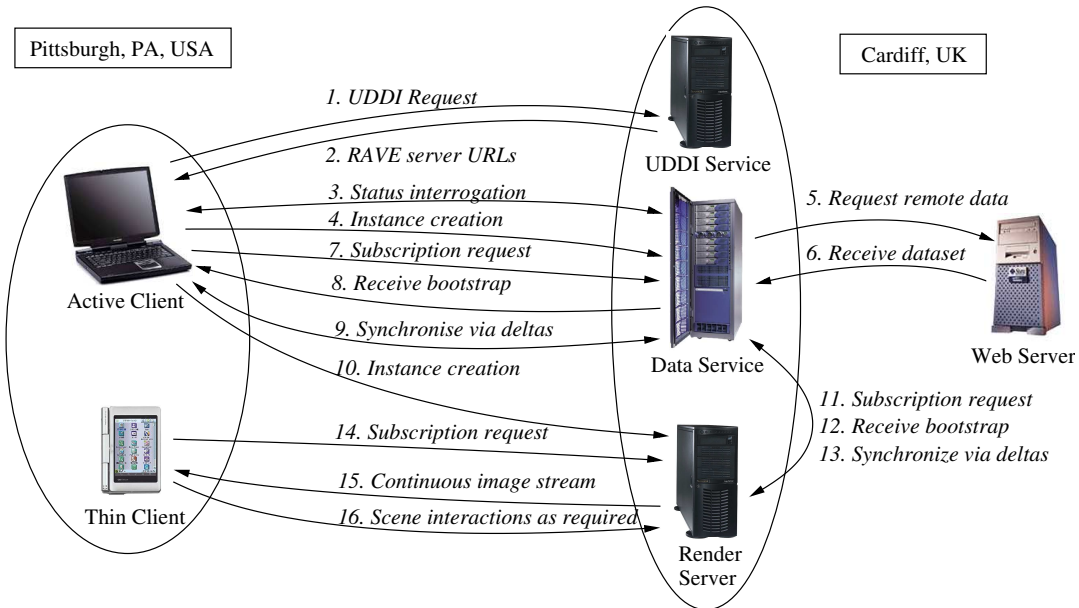


Fig. 4. Test configuration as used at SC2004

A Toshiba Satellite Pro M10 laptop was used as an active client for local rendering, whilst a Sharp Zaurus C760 PDA was used as a thin client. Both these devices were located in Pittsburgh at SC2004, and utilised resources in WeSC and the Cardiff School of Computer Science. An SGI Onyx 300 at WeSC was running a data service and an AMD-based Fedora 2 Linux server at the Cardiff School of Computer Science was running a render service. The configuration is presented in Figure 4 with a numbered sequence of events.

When the laptop and PDA were both displaying the DTI dataset, an avatar was visible for each client (the laptop can see the PDA's avatar, and vice-versa). This enabled the laptop to follow the PDA around the dataset, and share the same view. At this early stage of the project, we have only implemented minimal collaboration support, so we have yet to introduce gesticulation or direct communication (textual, speech or other), so the experience was limited. However, the thin and active clients were designed to continuously send messages when navigating the scene, at a frequency independent of the local rendering speed. Hence if the local machine slowed down, the actual movement of the avatar in space was still continuous. This was observed by the frame rate of the PDA being around 1-2 frames per second (fps) whilst the PDA's avatar was seen moving smoothly on the laptop.

The limitation of the collaboration (aside from an incomplete feature set) was mainly the speed of update on the PDA, which was hampered by the wireless connection. With exclusive use of an

802.11b wireless connection, the PDA can achieve around 4fps with a  $400 \times 400$  run-length encoded image, so we are limited by the network bandwidth. At SC2004, we were obtaining around 1-2fps due to the wireless bandwidth being shared with other show attendees. As the results are from observed rather than measured delays, they have a margin of error, but the observed latency between interacting with the PDA's GUI and a result arriving on the laptop was around 0.5s, with a total delay of around 0.7s before seeing the rendered result on the laptop.

## V. RESULTS

We now discuss the latest results from the RAVE project; namely heterogeneous tiled rendering.

### A. Heterogeneous Tiled Rendering

A RAVE Active Client can automatically utilise additional rendering resources; when the local host is overwhelmed by the dataset, it can offload portions of the rendering to remote render services to improve interactivity. In our tests, an active client was monitored for interactivity whilst rendering the entire frame buffer, and an alternative approach where one half of the frame buffer was rendered locally, the remaining half rendered off-screen on a remote render service.

Several datasets were tested, as shown in Figure 5, with results presented in Table I. The active client used was a Toshiba Satellite Pro M10 (1.6GHz Centrino, 512Mb RAM, NVidia GeForce Go! 420 GPU) running Linux, whilst the render server was a 1.2GHz Athlon (512Mb RAM, NVidia

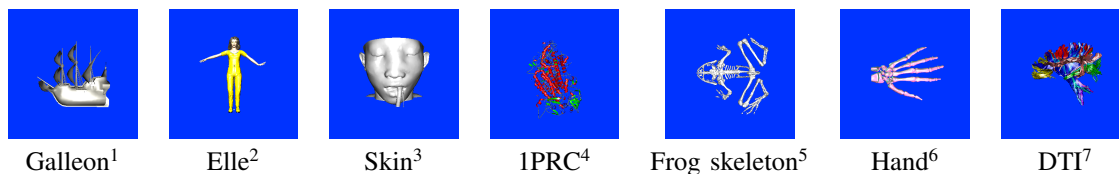


Fig. 5. Tiled rendering test models

TABLE I  
TILED RENDERING RESULTS

Dataset	KPoly	KVert	Vert/Poly Ratio	# Nodes	Active client FPS:		Render service FPS:			
					100%	50%	50%	Mb/sec	100%	50% on
Galleon	5.5	6.4	1.16	21	330.0	50.0	33.0	7.5	55.6	386.0
Elle	17.5	52.5	1.33	301	87.0	15.0	31.0	7.1	42.0	96.2
Skin	71.7	107.6	1.50	5	28.0	6.0	24.0	5.5	33.4	57.3
IPRC	429.3	939.6	2.19	29011	5.5	6.6	4.6	1.0	1.9	1.8
Frog skeleton	663.0	901.0	1.36	5	3.5	3.6	7.2	1.6	6.8	6.9
Hand	834.0	855.0	1.03	8	8.7	16.2	21.0	4.8	8.7	9.0
DTI	935.2	1,246.0	1.33	2167	7.1	7.6	10.2	2.3	9.7	10.4

GeForce FX 5200 GPU), also running Linux. Note that the tiling will work between any combination of machines; these particular machines were selected for convenience and availability during testing.

The datasets were rendered at  $400 \times 400$  resolution with 24 bits per pixel; the images were transmitted uncompressed to require minimal processing by the CPU. We used a private 100MBit ethernet network for testing, using a peak of 7.5Mb/sec (maximum theoretical bandwidth is 10Mb/sec).

Each row of the table represents a different dataset, with a note of its number of polygons (1000's), vertices (1000's), ratio of vertices to polygons (to give an indication of triangle strips, for render efficiency) and the number of nodes in the scenegraph (more nodes imply a higher maintenance overhead during rendering). The first result column for the active client is the frames per second (FPS) rendered when 100% of the dataset was rendered on-screen, followed by 50% rendered locally, the remainder of the image rendered by the render service, received by the active client and blitted into the presentation window. The render service result columns represent the render time taken for 50% of the image to be rendered off-screen (as transmitted to the active client). 100% off-screen is of the entire image rendered for comparison, whilst 50% on-screen is actually the same

view as 100% off-screen (i.e. the entire dataset), but rendered at half-width as a test of fill rate (same number of polygons drawn for 100% off and 50% on with the render service results).

### B. Discussion and Analysis

The first point to note is that the active client is actually slowed down if tiling is enabled with small datasets (e.g.  $< 100$  kpolygons). This is because the local GPU is running at an interactive rate, so any additional work (namely blitting an image to the GUI) will interrupt the accelerated rendering. The local rendering is running in different threads to the remote tile, and they do not (need to) synchronise. The remote rendering is received into a buffer, converted to a 2D image and the Java GUI is informed that the image needs to be refreshed. The local rendering is continuous and maintained by the Java3D system threads, independently of RAVE.

Note that the render service is slower at producing 50% of the image than 100%; this is because 100% image was produced for local off-screen testing with the render service, whereas the 50% image was transmitted over the network. Once the tests use a low amount of network bandwidth ( $< 5$ Mb/sec), the network ceases to be a bottleneck. Note that image compression would alleviate this, at the expense of CPU load on the host machine, but this would in turn reduce the local FPS. As our peak uncompressed network transmission is 7.5Mb/sec (33 FPS), this is perfectly adequate for an interactive session and hence does not bias this particular test with an unrealistic framerate.

Rendering 100% off-screen on the render service machine is slower than rendering 50% on-screen; this is caused by the Java3D model for off-screen

<sup>1</sup>Wavefront OBJ file provided with Java3D

<sup>2</sup>VRML file from Blaxxun benchmarks [2]

<sup>3</sup>Sample ASCII file provided with VTK [17]

<sup>4</sup>VRML file produced via MolScript [13] of IPRC [8] molecule from the Protein Data Bank [12]

<sup>5</sup>Sample binary file provided with VTK

<sup>6</sup>PLY file from Georgia Tech large model archive [20]

<sup>7</sup>Diffusion Tensor Imaging data file [7]

rendering, where the user most poll to determine if the current frame has completed rendering. On-screen rendering runs continuously without user maintenance, hence running faster when the GPU is not overloaded. Once the GPU is overloaded (for instance, producing  $<10$  FPS), the additional overhead of polling has little effect on the final off-screen framerate.

With complex images, when the local GPU is starting to struggle (indicated by the frame rate falling below 10 FPS), the tiling achieves slight to moderate improvement on render times. The best result is obtained with the hand model; this is probably explained as the geometry is efficiently tri-stripped (as show by the ratio of vertices to polygons), so is processed rapidly by the GPU. The remote render service's GPU is slightly faster than the local GPU, and when the image is split in half, the visible vertices are roughly distributed 50-50 between client and service (i.e. the camera location and model orientation produce an even load balance of vertices).

However, it appears that in general distributing the rendering across several machines only produces a significant speedup when the bottleneck is the fill rate of the GPU, not the transform and lighting (as this remains constant). This can be seen with the high-polygon IPRC and DTI models which are less efficiently stored than the hand (as shown by the vertex to polygon ratio), where the local client only achieves a slight speedup. With the skeletal hand, the distribution of the triangle fill boosts the local render speed with the remote render speed significantly faster than the local GPU.

RAVE would require a heuristic that can determine if the local GPU is bottlenecking on fill rate or transform and lighting; such a heuristic could then determine whether it would be worthwhile to engage in tiled rendering mode or simply revert to thin client mode. Thin client mode would produce a speedup if the remote render service had a faster GPU than the active client, and sufficient network bandwidth was available.

## VI. APPLICATIONS

### A. GECEM

RAVE is currently being integrated into the GECEM project [21] to enable collaborative, remote visualization of electromagnetic datasets. This requires support for a variety of systems, ranging from CAVEs to PDAs. Project partners are based in Wales and Singapore, hence RAVE must react to available network bandwidth and latency. The RAVE portlet currently in development enables RAVE to be inserted into any JSR168 compliant

portal, such as GridSphere [16] (as used in the GECEM project). This will enable RAVE to be used transparently by the project partners, with the underlying rendering mechanism hidden from users.

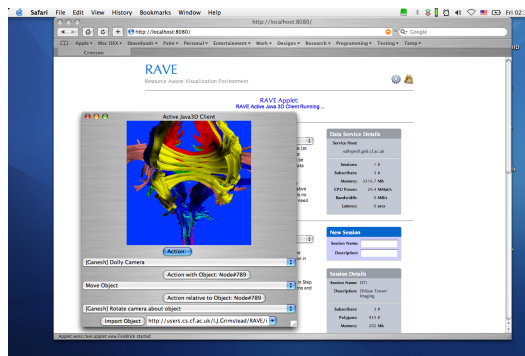


Fig. 6. Screenshots of RAVE launched from a Web Browser, showing the DTI dataset [7] (MacOS X)

Initial results of RAVE running as an applet are presented in Figure 6. The web page is supported by a Java Bean, which interrogates the UDDI server for available services. The Java Bean also checks if the local machine supports Java3D, and if so an Active Client is made available to the user. Otherwise, the user must use a Thin Client via a Render Service. A separate window is then spawned to display the visualization.

### B. Molecular Dynamics Simulation

RAVE can also maintain a live link with a 3rd party application, an example of which is a molecular dynamics simulation, which has been

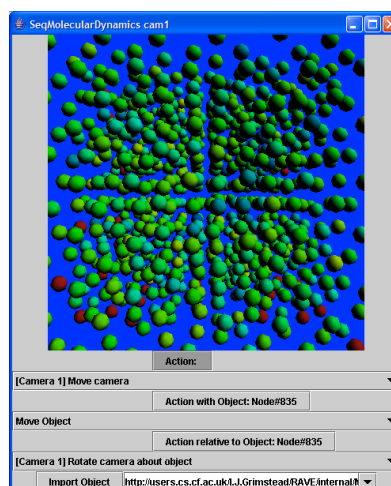


Fig. 7. Screenshot of Molecular Dynamics Simulation of a Liquid from RAVE (MS Windows XP)

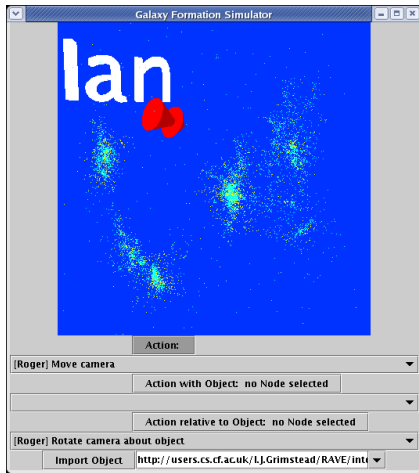


Fig. 8. Screenshot of Galaxy Formation Simulator from RAVE (32-bit Linux)

instrumented to communicate with RAVE; this enables multiple users to collaboratively view the simulation as it progresses. A sample screenshot is presented in Figure 7, where false colour has been applied to molecules to indicate their current velocity.

### C. Galaxy Formation Simulation

Point cloud rendering has been implemented to support the visualization of galaxy formation, as presented in Figure 8. The datasets are output from the simulator at linearly spaced internal time epochs. Each time frame of the dataset consists of an entry for each sample point in the simulation, including position, velocities, properties etc. For even a modestly sized number of initial sample points (e.g. one million), the high level of information required about each sample point leads to dataset frames being tens of Mbs, and hence the overall size of the dataset can be of the order of several Gb. Post processing and analysis of this data takes several forms but is primarily render based: a morphological study of the galaxy structure for instance (see Figure 8); or a line-of sight mass density calculations.

Serial and distributed analysis tools such as Anim and Column<sup>1</sup> have been developed but given their limited infrastructure, processing and data access ability these packages can consume considerable resources producing slow renders.

<sup>1</sup>Anim and Column were first generated as single platform tools in the 1980s by members of the Dept of Physics and Astronomy, Cardiff University. A distributed animation and analysis tool 'Animator' has been developed by R Philp since 2002, based on JAVA-RMI technology combining the functionality of Column and Anim. Its components also form units for the Triana project [19]

However in the context of the autonomous nature of RAVE and its infrastructure, the data is stored at each client, with the current time step broadcast to keep clients and services loosely synchronised with the playback. This has been used to interactively visualize the simulation output in stereo, using the Portico Workwall at WeSC supported by an SGI Onyx 300.

### D. General Applications

Datasets visualized under RAVE are in the range from a few thousand polygons up to the 4.5 million polygon ETOPO dataset (a surface relief of the world). Formats supported include VRML, VTK, Ensign Gold, ETOPO, J3D stream, Wavefront OBJ and specialised importers (such as for Diffusion Tensor Imaging). The data is imported into the Data Service at session creation, and may be from a local file or a remote site (<http> or <ftp>).

## VII. FUTURE WORK

At present, we are only using polygonal and point datasets. We will extend our support and rendering services to include voxel based methods; these will distribute across multiple render services. Subset blocks of the volume can be blended, even though they contain transparency, by considering their relative distance from the view in the order of blending (such as Visapult [1]).

We will investigate automatic selection of level-of-detail, to attempt to maintain a given frame rate and to enable a dataset to be rendered on a machine with otherwise insufficient resources. This will extend our work for variable frame buffer compression to vary the render time in addition to varying compression and transmission time.

All image and data compression methods must be switchable, in case a user requires perfect detail and is prepared for slow frame rates (an example would be a medical practitioner examining data—they do not want important image detail obscured by a compression artifact).

We will implement a “Visualization Wizard GUI, where the user initially enters a dataset they wish to view. If this is already loaded into a data service, then the user will be given the option to join the existing service or create a new session. Similarly, the user can opt to reuse an existing render service or create a new session as required. This wizard can then be extended to support user preferences, used to determine which service will provide the most appropriate level of service for the user.

## VIII. CONCLUSIONS

We have presented a novel and highly capable visualization framework that supports heterogeneous resources, in terms of system architecture and system resources.

The use of web services has enabled us to connect clients and servers without any underlying knowledge of the source/target hardware, operating system or implementation language. This enables RAVE to be supported on a wide variety of platforms, from a C++ based PDA client (which has minimal local resources), through desktop machines running Mac OS, Windows XP or Linux, up to large-scale computers such as an SGI Onyx 300.

Our framework supports collaboration, enabling a hand-held device to interact with a user in an large-scale environment (such as a Portico WorkWall or a projected display). This also enables the PDA to be used as a private display by the presenter, and then to remote-control the publicly visible display when required (all from the PDA).

The flexible nature of RAVE has supported the integration into other systems (such as GECM and various simulators), and the use of Java3D has simplified the task of importing of external data formats.

Java3D has enabled us to access GPU resources without interrupting the local user, to the extent of real-time cross-platform load distribution through tiled rendering. Tiled rendering returned the best results when a large scene was presented that bottlenecked on fill rate in the GPU, rather than vertex transformation and lighting. Distribution of subsets of the scene graph (whilst rendering a full image) would be worthy of investigation with such scenes, to distribute the load of transformation and lighting.

## REFERENCES

- [1] E.Wes Bethel and John Shalf. Grid-distributed visualizations using connectionless protocols. *IEEE Computer Graphics & Applications*, pages 51–59, March/April 2003.
- [2] blaxxun. blaxxun—Contact 3D, VRML Benchmark. <http://developer.blaxxun.com/developer/contact/3d/bench/elle.htm>, 2004.
- [3] 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. In *STAR—State of the Art Report, Eurographics*, 2004.
- [4] Ken Brodli, Jason Wood, David Duce, and Musbah Sagar. gViz: Visualization and Computational Steering on the Grid. In *Proceedings of UK e-Science All Hands Meeting 2004*, September 2004.
- [5] Stuart M. Charters, Nicolas S. Holliman, and Malcolm Munro. Visualisation on the Grid: A Web Service Approach. In *Proceedings of UK e-Science All Hands Meeting 2004*, September 2004.
- [6] Jim X. Chen, Yonggao Yang, and Bowen Loftin. MUVEES: a PC-based Multi-User Virtual Environment for Learning. In *Proceedings of the IEEE International Symposium on Virtual Reality 2003 (VR '03)*, pages 163–170. IEEE Computer Society, March 2003.
- [7] Martin Connell and Mark Bastin. Diffusion Tensor Imaging (DTI) dataset. Personal communication, SHEFC Brain Imaging Research Centre for Scotland, 2004.
- [8] J. Deisenhofer, O. Epp, I. Sinning, and H. Michel. Crystallographic refinement at 2.3 Å resolution and refined model of the photosynthetic reaction centre from *Rhodospseudomonas viridis*. *Journal of Molecular Biology*, 246:429, 1995. PDB ID: 1PRC.
- [9] 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.
- [10] Ian J. Grimstead, Nick J. Avis, , and David W. Walker. Visualization across the pond: How a wireless pda can collaborate with million-polygon datasets via 9,000km of cable. In *Proceedings of the 10th International Conference on 3D Web Technology (Web3D 2005)*, March 2005.
- [11] 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 SC2004: SuperComputing 2004*, November 2004.
- [12] H.M.Berman, J.Westbrook, Z.Feng, G.Gilliland, T.N.Bhat, H.Weissig, I.N.Shindyalov, and P.E.Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000.
- [13] Per Kraulis. MolScript. <http://www.avatar.se/molscript/>, 1999.
- [14] Ioana M. Martin. Hybrid Transcoding for Adaptive Transmission of 3D Content. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)*, August 2002.
- [15] National Geophysical Data Center. GEODAS Grid Translator—Design-a-Grid. [http://www.ngdc.noaa.gov/mgg/gdas/gd\\_designagrid.html](http://www.ngdc.noaa.gov/mgg/gdas/gd_designagrid.html), 2004.
- [16] Jason Novotny, Michael Russell, and Oliver Wehrens. GridSphere: An Advanced Portal Framework. In *Proceedings of the 30th EUROMICRO Conference*. IEEE Computer Society, September 2004.
- [17] William J. Schroder, Kenneth M. Martin, and Lisa S. Avila. *VTK User's Guide - VTK File Formats*, chapter 14. Kitware Inc., 2000. Section on VTK File Formats published independently.
- [18] SGI. SGI OpenGL VizServer 3.1. Data sheet, SGI, March 2003.
- [19] Ian Taylor, Matthew Shields, Ian Wang, and Roger Philp. Distributed p2p computing within triana: A galaxy visualization test case. In *International Parallel and Distributed Processing Symposium (IPDPS'03)*, page 16a, April 2003.
- [20] Greg Turk and Brendan Mullins. The Large Geometric Models Archive. [http://www.cc.gatech.edu/projects/large\\_models/](http://www.cc.gatech.edu/projects/large_models/), Georgia Institute of Technology, 2003.
- [21] David W. Walker, Jonathan P. Giddy, Nigel P. Weatherill, Jason W. Jones, Alan Gould, David Rowse, and Michael Turner. GECM: Grid-Enabled Computational Electromagnetics. In *Proceedings of the UK e-Science All Hands Meeting 2003*, pages 436–443, September 2003.
- [22] Jeremy Walton. *Visualization Handbook*, chapter NAG's IRIS Explorer. Academic Press, 2003.
- [23] Paula Womack and Jon Leech. OpenGL Graphics with the X Window System. <http://www.opengl.org/documentation/specs/glx/glx1.3.pdf>, October 1998.
- [24] U. Wössner, J. Schulze-Döbold, S.P. Walz, and U. Lang. Evaluation of a Collaborative Volume Rendering Application in a Distributed Virtual Environment. In *Proceedings of the 8th Eurographics Workshop on Virtual Environments (EGVE)*, pages 113–122, 2002.