Available online at www.sciencedirect.com

SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 896–900

FGCS FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

Highly interactive distributed visualization

M. Scarpa[a], R.G. Belleman[a,*], P.M.A. Sloot[a], C.T.A.M. de Laat[b]

[a] *Section Computational Science, Scientific Visualization and Virtual Reality Group, Informatics Institute, Faculty of Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, Netherlands*

[b] *Advanced Internet Research Group, Informatics Institute, Faculty of Science, Universiteit van Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, Netherlands*

Available online 11 May 2006

Abstract

We report on our iGrid2005 demonstration, called the "Dead Cat Demo"; an example of a highly interactive augmented reality application consisting of software services distributed over a wide-area, high-speed network. We describe our design decisions, analyse the implications of the design on application performance and show performance measurements.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Highly interactive visualization; Distributed visualization; Augmented Reality

1. Introduction

While the Grid paradigm has proven useful for high-performance computing tasks in many occasions, using this paradigm for visualization purposes remains a challenge, especially when considering highly interactive visualization as found in e.g. Virtual Reality environments. The reason for this lies in the demands that this class of visualization applications in general impose: graphical representations must be of high quality, response to user interaction must be immediate, the different components are often diverse and sometimes many. Different demands may often seem irreconcilable as design decisions on one invariably influence another. Addressing these problems is a non-trivial task, especially considering the distributed nature of a Grid. In this work we investigate some of these issues through the design and implementation of a highly interactive visualization application that consists of distributed components. We use this application as a test case to investigate the implications of a service oriented design (as proposed by e.g. WSRF[1]) on distributed, highly interactive visualization applications.

* Corresponding author. Tel.: +31 20 525 7510.
E-mail addresses: mscarpa@science.uva.nl (M. Scarpa), robbel@science.uva.nl (R.G. Belleman), sloot@science.uva.nl (P.M.A. Sloot), delaat@science.uva.nl (C.T.A.M. de Laat).

[1] http://globus.org/wsrf/.

2. Related work

Several publications report on Grid-based scientific visualization: Brodlie et al., analyze and revisit the data flow concept used in most visualization systems in the light of Grid computing developments [1]. Similarly, Charters et al. present an approach based on Web Services to provide desktop based visualization [2]. Their approach deploys the visualization on a Grid by encapsulating every single element of the visualization pipeline in a separate Web Service. In the CrossGrid project, a framework is presented that deploys a flow-visualization service on a Grid [3,4]. Stanton et al. discuss the integration of scientific visualization facilities in a component based Grid middleware [5]. While the authors discuss the implications of Grid and Web Service technology in scientific visualization, they do not address the implications of their designs in highly interactive visualization applications.

Singh et al. tackle the issues raised by remote visualization in their "Tera-Vision" setup based on dedicated hardware used for capturing, compressing, transferring and decompressing image data in the form of video signals [6]. Also, Luke and Hansen propose a framework for remote visualization that implements various scenarios for distributing data computation, visualization and rendering in a client–server architecture [7].

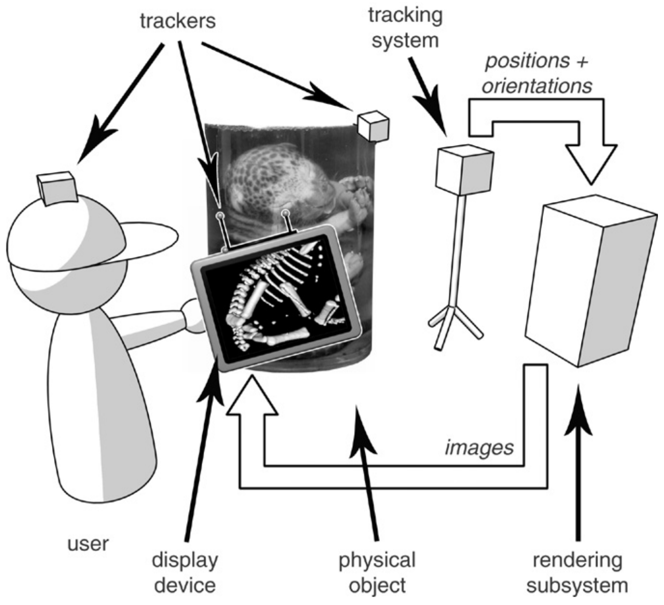


Fig. 1. Design of our demonstration.

3. Design of the demonstration application

For our test case, we designed a highly interactive visualization application that can easily be broken down into different software components, each providing a unique functionality. We refer to these components from here on as *services*.[2] These services are loosely coupled over a generic network connection and can thus be distributed over a Grid-like environment.

The application provides intuitive access to visualizations of data that is associated with physical objects. This data is visualized “collocated” with the real object (superimposed on top of it), allowing the user to inspect both the real object as well as the computer generated images at the same time. For this purpose, the user holds a hand-held display device in front of the real object. The system renders images on the display such that the user is given the impression of looking through the device onto the real object, with the major difference, though, that the user sees the visualized data in place of the real object (see Fig. 1). When the user moves the display, the image must change immediately to give the impression of looking through the device. This method of highly interactive visualization falls under the category “Augmented Reality” and has been described in various works preceding ours, e.g. the work of Tsang et al. [8], although we are not aware of any such work that uses a comparably distributed design as we used here.

To achieve the described functionality, we identify the need for three main services, as shown in Fig. 2. In order to display the images correctly onto the display device, the exact position and orientation of the user’s point of view, the display device itself, as well as the inspected object are required. These are

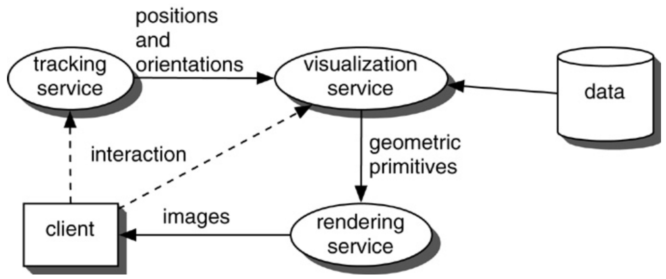


Fig. 2. A schematic representation of the system design.

obtained using a *tracking service*, which for obvious reasons must be physically present at the user’s location. The positions and orientations reported by the tracking service are provided to a *visualization service*. This service translates the data to be visualized according to the given positions and orientations into geometric primitives (triangles, textured planes, etc.). The geometric primitives are then handed over to a *rendering service*. This service generates an image from the geometric primitives and then transfers this image to a client for display.

3.1. Performance model

Clearly, to achieve the described impression, response of the application must be close to immediate. To model the performance characteristics of our design we analyze the latencies introduced by each service. We consider the data flow from the moment the user interacts with the system, moving any of the tracked objects, to the moment the correct image is displayed (see Fig. 3). The functional requirements of our test case application require that these latencies are as small as possible.

The first group of delays is specific to the tracking service and depends on the technology used to track objects. The next delay is specific to the connection between the tracking service and the visualization service. In our distributed design, this is often a network connection. The following delays are specific to the visualization service and include also delays related to the chosen visualization method and its resource requirements. To increase performance, this service could be executed on a dedicated visualization platform. The next delay is again related to the connection between the services. The delays within the rendering service are specific to the used hardware. For performance reasons, the images are often compressed by the service. The delay between the rendering service and the display device depends on the connection method used. Finally, the last group of delays is related to the display device, specifically the reception, decoding and displaying of the image served by the rendering service.

3.2. Demonstration setup

For SuperComputing (SC2004) in Pittsburgh and the iGrid 2005 workshop in San Diego, a demonstration application of the described architecture was developed. All the services are hand-configured and managed manually in this early prototype.

[2] Please note that our application was not implemented using grid or web services.

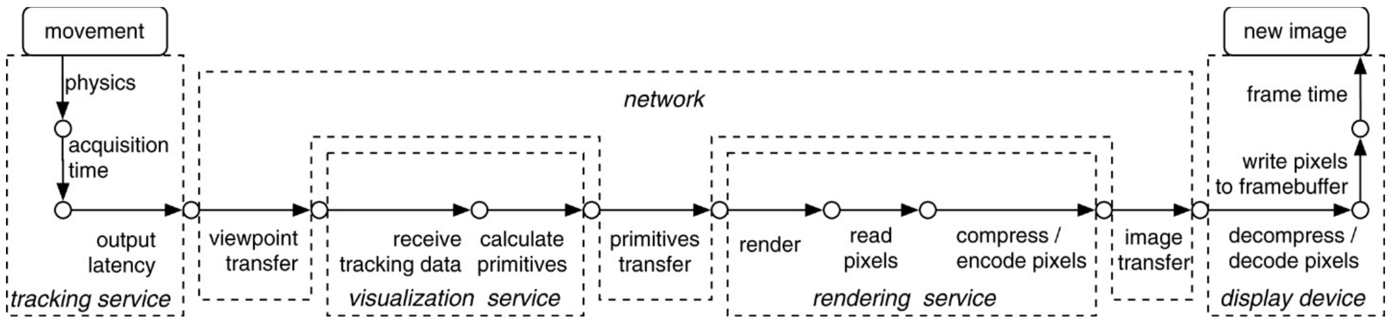


Fig. 3. A diagram representing the various latencies that affect the responsiveness and interactivity of the system.

The tracking service uses an optical tracking system provided by ART GmbH.[3] This system allows tracking position and orientation of multiple objects at an average update rate of 60 Hz.

The visualization service uses SGI OpenGL Volumizer[4] and Kitware's Visualization Toolkit (VTK) [9]. For the rendering service, we use SGI OpenGL Vizserver.[5] A Vizserver *client* application is executed on the display device: an HP 1100 Tablet PC running Windows XP Tablet edition is used as a display device. The images are displayed full-screen at a resolution of 1024×768 pixels and 32 bit color depth. Vizserver provides image compression algorithms on the server and decompression on the client.

For visualization and rendering we use an SGI Onyx4 with 24 MIPS R16000 processors running at 600 MHz, 20 GB of memory and 10 FireGL X2 graphics cards, hosted at SARA Computing and Networking Services in The Netherlands.[6] The Onyx4 uses a VANier board to do pixel read-out in hardware. This provides a considerable speedup compared to performing the same operation in software. By running both services on the Onyx4, we minimize the communication overhead between the visualization and rendering services.

A 100 year old preserved panther cub was used as the subject of interest. The panther cub is conserved in a glass jar and was scanned with a CT scanner at the Academic Medical Center (AMC), Amsterdam. The data set consists of 172 by 246 by 167 voxels, each represented by two bytes of data, resulting in a raw data size of about 13.5 MB.

In our setup we track a cap worn by the user (to retrieve the position of the user's head), the display device, as well as a small rectangular frame used for user interaction. The glass jar with the panther cub is placed into a fixed position and is not moved during demonstrations.

4. Results and discussion

The demonstrations at SuperComputing and iGrid gave us two opportunities to test the performance on long distance network connections. In both cases, the hardware for the tracking service and the display device were physically present at the demonstration site, while the Onyx4 running the visualization and rendering services was located in Amsterdam and accessed remotely.

For the demonstration during iGrid we used a routed path from San Diego over Chicago/StarLight, over Abilene to New York and from there over the transatlantic SURFnet link directly to Amsterdam. The main challenges were the high round-trip-time (160 ms) and the slowest link in the connection: the 100 Mb ethernet interface of the Tablet PC. The bandwidth delay product for this setup results in approximately 2 MB [10]. For the demo at iGrid the window sizes were set to 16 MB on each side, which was more than sufficient. Since this network connection is used to transfer only the image data produced by the rendering service, the amount of transferred data depends only on the resolution of the images and the compression achieved by the rendering service; it is independent of the size of the data set being visualized. Additionally, TCP Selective Acknowledgements were used on the Onyx4 in Amsterdam, due to the use of large TCP window sizes.

At both Supercomputing and iGrid, we achieved frame rates of approximately 4 frames per second (fps). The latency of our setup was noticeably high, as can be expected from our performance model. To further investigate the performance of the different components in our setup, we conducted more detailed experiments in Amsterdam. To recreate the high latencies experienced at the demonstration locations, traffic between the Onyx4 and a client was routed over a loop to New York and back. The average latency on this connection was approximately 176 ms. For this loop, the TCP window size of the Onyx4 and the client was set to the bandwidth delay product: 2.1 MB. This setup was compared to a local area network connection between the Onyx4 and the client.

The tracking service uses UDP to transmit packets to the rendering system. In our setup three objects are tracked and we have measured that on average, one packet is 502 bytes long (including header). Measurements indicate that packet loss is negligible, and considering the small amount of data, payload transfer time can also be neglected. As such, the only latencies introduced by the tracking service are the data acquisition and the output time. On average, data acquisition takes 16.7 ms. The output time is only limited by the network latency.

Since both the visualization and the rendering service are deployed on the same system, there is no transfer of primitives over the network as depicted in Fig. 3. Our measurements

[3] http://www.ar-tracking.de/.
[4] http://www.sgi.com/products/software/volumizer/.
[5] http://www.sgi.com/products/software/vizserver/.
[6] http://www.sara.nl/.

Table 1
Comparison between volume rendering and polygon rendering applications

	Volume rendering			Polygon rendering		
	Min	Avg	Max	Min	Avg	Max
Render time (ms)	156.59	222.9	365.78	3.82	4.64	22.58
Read back[a] (ms)	17	21.4	30	0	1.98	17
Compress/encode[a] (ms)	12	14.01	22	18	25.53	50
Server frame rate[a] (fps)	2	4.28	7.5	41.9	53.45	60.7
Discarded[a] (fps)	0	0.001	2	32.7	43.26	51
Client frame rate[a] (fps)	2	4.28	7.5	6.2	10.21	14.7

[a] Values measured using the utility "vsmonitor", part of Vizserver.

(see Table 1) show that the calculation of rendering primitives and the rendering thereof introduce a considerable latency: on average, it takes 222.9 ms to render one frame.

The Vizserver server reads back images as fast as possible. If the display device running the Vizserver client is too slow to show the same frame rate as the server provides, the latter discards older frames, only transmitting the latest frame. Because of the high latency in image generation, the Vizserver server only reads images at 4.28 fps. Because this rate is so low, barely any frames are discarded and the same frame rate is achieved on the client.

Running the same application over a local area network connection, we found comparable values for all the latencies except the network latency. This results in similar values for the achieved frame rate, although the responsiveness is obviously higher. Measurements of the bandwidth usage for the image transfer show that the communication between the Onyx4 and the client reaches less than 1 MB/s.

For comparison, we performed the same measurements over the high latency loop with another visualization (see Table 1). In this case we used a very simple scene which on average takes 4.64 ms to render. The rendering service reads back images at an average of 53.47 fps, but discards them at a rate of 43.26 fps, resulting in a frame rate of 10.03 fps on the client. Repeating the same measurements over the local network connection yields an average frame rate on the client of 16.85 fps (between 12.1 and 21.3). The bandwidth usage of the communication between the Onyx4 and the client in both cases increases only marginally to about 1.2 MB/s.

Our measurements show that the main bottleneck of our demonstration lies in the image generation, which causes the low frame rate. The measurements with the simpler visualization show that also the high network latency reduces frame rate. It must be noted at this point that the technology used to transfer images over the network adds a proprietary acknowledgment scheme on top of the native scheme of TCP. Our measurements give us reason to believe that this causes sub-optimal bandwidth use.

The overall measured latency (neglecting payload transfer time) of our test setup in Amsterdam results in 465.02 ms. Even disregarding the high latency of the image generation, the latency still results in 242.12 ms, which clearly needs to be improved for highly interactive applications.

5. Conclusions and future work

This demonstration shows that a highly interactive application as described can be implemented on a Grid environment using a service-based architecture. One of the main challenges in this design is posed by the requirements to the network infrastructure.

We would like to point out that there is currently no control over the network resources for the deployment of such highly interactive applications. We believe that it would be extremely beneficial for the realization of architectures as we envision in this paper, to have access to services that could allocate the required network resources in a similar way as computing and storage resources can be allocated in a Grid environment.

Another aspect that should be addressed is the networking protocol used for image transfer from the rendering service to the client. The technology used in our prototype employs TCP, which suffers some limitations that make it less suitable for long distance high performance data transfer. Also, the communication scheme implemented by this technology additionally decreased performance. We therefore believe that it would be beneficial to replace the current implementation of the image transmission using alternative technologies, e.g. using revised versions of TCP [11] or UDP-based protocols [12,13].

Considering the various limitations in the currently chosen platform for the visualization and rendering services, we are also looking into alternatives to speed these up. The modularity of our design proves of advantage in this case, since it allows us to easily implement these services on other resources without major changes in the overall architecture, nor in the other services.

If issues related to image generation and network protocols are neglected, the network imposes the highest latency on our system design. To achieve a lower latency – which is an unavoidable necessity for highly interactive visualizations – this latency needs to be reduced further, although both technical as well as physical limitations set boundaries to the possible improvements.

Acknowledgements

This work was carried out in the context of the Virtual Laboratory for e-Science (VL-e) project.[7] This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT Innovation program of the Ministry of Economic Affairs (EZ). We also acknowledge the help and support of the following organizations: SARA Computing and Network Services, Academic Medical Center (AMC) Amsterdam, SGI Netherlands, GigaPort and the Zoölogisch Museum Amsterdam.

More details on this work can be found on http://www.science.uva.nl/research/scs/projects/visualization/.

[7] http://www.vl-e.nl/.

References

[1] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, J. Wood, Visualization in grid computing environments, in: VIS '04: Proceedings of the Conference on Visualization'04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 155–162.

[2] S. Charters, N. Holliman, M. Munro, Visualization on the grid: A Web Service approach, in: Proceedings UK eScience third All-Hands Meeting, Nottingham Conference Center, August 31st–September 3rd, 2004.

[3] P. Sloot, G. van Albada, E. Zudilova, P. Heinzlreiter, D. Kranzlmüller, H. Rosmanith, J. Volkert, Grid-based interactive visualisation of medical images, in: S. Nørager (Ed.), Proceedings of the First European HealthGrid Conference, 2003, pp. 57–66.

[4] A. Tirado-Ramos, H. Ragas, D. Shamonin, H. Rosmanith, D. Kranzmueller, Integration of blood flow visualization on the grid: the FlowFish/GVK approach, in: Proceedings of the 2nd European AcrossGrids Conference, Nicosia, Cyprus, in: Lecture Notes in Computer Science, vol. 3165, 2004, pp. 77–79.

[5] J. Stanton, S. Newhouse, J. Darlington, Implementing a scientific visualisation capability within a grid enabled component framework, in: 8th International Euro-Par Conference, Paderborn, Germany, in: Lecture Notes in Computer Science, vol. 2400, 2002.

[6] R. Singh, J. Leigh, T.A. DeFanti, F. Karayannis, TeraVision: a high resolution graphics streaming device for amplified collaboration environments, Future Generation Computer Systems 19 (6) (2003) 957–972.

[7] E. Luke, C. Hansen, Semotus visum: A flexible remote visualization framework, vis, in: 13th IEEE Visualization 2002 (VIS'02), 2002.

[8] M. Tsang, G.W. Fitzmaurice, G. Kurtenbach, A. Khan, B. Buxton, Boom chameleon: simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display, ACM Transactions on Graphics 22 (3) (2003) 698.

[9] B.L.W. Schroeder, K. Martin, The Visualization Toolkit. An Object-Oriented Approach to 3D Graphics, 3rd edition, Prentice Hall, 2003. http://www.vtk.org/.

[10] W.R. Stevens, Unix Networking Programming, in: Networking APIs: Sockets and XTI, vol. 1, 2nd edition, Addison-Wesley, Reading, MA, 1998, p. 357.

[11] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, Stream Control Transmission Protocol (RFC 2960). http://www.ietf.org/rfc/rfc2960.txt.

[12] E. He, J. Leigh, O. Yu, T. DeFanti, Reliable blast UDP: Predictable high performance bulk data transfer, in: Proc. IEEE Cluster Computing 2002, Chicago, Illinois, September 2002.

[13] E. He, J. Alimohideen, J. Eliason, N.K. Krishnaprasad, J. Leigh, O. Yu, T.A. DeFanti, QUANTA: A toolkit for high performance data delivery over photonic networks, in: IGRID 2002, Future Generation Computer Systems 19 (6) (2003) 919–933 (special issue).

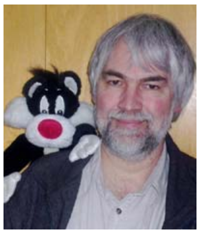
Michael Scarpa received his master in computer science at the Johannes Kepler University in Linz, Austria, and is currently a Ph.D. student at the University of Amsterdam, The Netherlands. His research interests include Scientific Visualization, Virtual Reality and interaction in Virtual Environments. More information at: http://www.science.uva.nl/~mscarpa/

Robert G. Belleman is the head of the Scientific Visualization and Virtual Reality group of the Section Computational Science at the University of Amsterdam. He obtained his Ph.D. in computer science from the Universiteit van Amsterdam for his work on Interactive Distributed Exploration Environments. He currently participates in several joined projects where he works on Grid-based problem solving environments and interactive scientific visualization environments with a focus on medical applications. More information at http://www.science.uva.nl/~robbel/

Prof. Dr. Peter M.A. Sloot, studied physics and chemistry, did his Ph.D. work at the Dutch Cancer institute (NKI) and is currently a full professor in Computational Sciences at the Informatics Institute of the Faculty of Science of the Universiteit van Amsterdam, the Netherlands.

In his research, he focuses on theory, methods and tools for distributed mesoscopic simulation. He is General Chair of the ICCS series of conferences on Computational Sciences and Editor in Chief of Elsevier's science journal: FGCS: International Journal on Grid Computing.

Professor Sloot received in 1996 an NNV distinguished professorship in computational physics and has published over 120 journal papers, 100 proceeding papers and 7 chapters in books. He supervised 18 Ph.D. theses. His current interest is in Grid Computing and distributed simulation.



Cees de Laat is associate professor in the Informatics Institute at the University of Amsterdam. Current research includes optical/switched networking to optimize Internet transport of massive amounts of data for the Grid, distributed cross organization Authorization architectures and grid Workflow systems. With SURFnet he implements projects in the GigaPort Research on Networks area. He serves as Grid Forum Steering Group (GFSG) Infrastructure Area Director and IETF Liaison. He is co-founder and organizer of several of the past meetings of the Global Lambda Integrated Facility (GLIF). For more info: http://www.science.uva.nl/~delaat