CEV: Collaborative Environment for Visualization Using Java RMI

RAJEEV R. RAJE, MICHAEL BOYLES, SHIAOFEN FANG

Department of Computer and Information Science, Indiana University Purdue University Indianapolis

Indianapolis, IN 46202-5132, USA

SUMMARY

Collaborative visualization is an active area of research in computer science. Many different techniques and implementations have been proposed. However, most of today's systems require many high-end graphics workstations. By implementing a new technique that separates the 2D and 3D components of the visualization process, we have developed a system called CEV that operates using only one high-end workstation along with many inexpensive machines that need only the capability of running a web browser. Using Java RMI, CEV provides a heterogeneous environment for many users that is easy-to-use. This paper describes the operation, methodology, and implementation of CEV.

1 Introduction

Scientific visualization is the process of using computer graphics to view or explore 3D objects, natural phenomenon, or complex data [1]. It has proven to be very useful to many areas of science and engineering. Collaborative visualization enhances the traditional visualization by bringing together many experts so that each can contribute towards the common goal of the understanding of the object, phenomenon, or data under investigation.

There are many applications in science, technology and engineering which can benefit from collaborative visualization. Medical consulting is one such application. Similarly, geographically scattered engineers may be able to collaborate, design, analyze, and evaluate a prototype through a common engineering data visualization system.

1.1 Characteristics of collaborative visualization software

Due to the nature of the collaborative visualization process, we feel that any system developed for it should contain the following features:

- A support for multiple users and heterogeneous machines that may or may not be geographically distributed.
- A concurrency control mechanism so that all users can express their views clearly and in a fair manner.
- An user-friendly interface, as the participants of a collaboration session are generally domain experts, with lesser computer experience.
- An authentication process to avoid malicious accesses to important data.
- An ability to store/retrieve the results of collaborative sessions along with transcripts.

There are many research efforts to develop collaborative visualization systems. For the sake of braviety, here, we will mention only a few prominent ones.

As cited in [2], MMConf [3], Rapport [4], and CECED [5] allow for the content-independent sharing of drawing and viewing surfaces. Shastra [2] included the ability to perform content-dependent sharing. Shastra applications are "collaboration aware" and support synchronous multiuser manipulations. CSpray uses a "spray painting metaphor for an intuitive user-interface" [6]. The CSpray system decreases network traffic through incremental updates and handles shared resources through privately owned spray cans. CSpray is intended for a small group of collaborators. Wood et al. [7] describe an architecture that can offer a collaborative extension to a dataflow visualization system. Northeast Parallel Architectures Center (NPAC) has developed TANGOsim [8]. TANGOsim is a collaborative environment providing message filters and event driven simulators that was written in Java to provide architecture independence. In TANGOsim, the clients are implemented as applets which are launched from ordinary web browsers.

1.2 CEV: A new approach to collaborative visualization

Most of these existing systems require each participant to have a high-end graphics workstation. Also, because they transmit graphics primitives (mostly surface-based) between machines and force the local client to perform all rendering, these systems do not consider large data visualization applications (particularly volume-based). For an application which generates a typical volume data set of size $256 \times 256 \times 256$, it is not practical to transmit the entire data set over the network and render it on a low-end workstation in a reasonable amount of time.

Here we propose an innovative approach for collaborative visualization which forces a clear separation of 2D and 3D components, thereby allowing the participants to use inexpensive workstations. Based upon this approach, we have developed a prototype called CEV (Collaborative Environment for Visualization). In CEV, each user (or client) connects to a single high-end machine (or server). The server handles all computations necessary to generate the 2D images which each participant will eventually view. Once a new image has been generated, it is sent over the Internet and displayed inside an ordinary web browser. Below, we will discuss how CEV supports the necessary features mentioned in the subsection 1.1.

- *Support multiple users/machines that are geographically dispersed*: In order to provide an environment that allows heterogeneous machines to communicate, CEV is implemented in Java. Communication between users is handled using Remote Method Invocation (RMI). Due to the architecture neutrality, Java and RMI handle geographically distributed heterogeneous machines and provide a transparent view to the participants. For more information on either, the reader is referred to [9].
- *Users must have the ability to communicate concurrently and in a fair manner*: CEV uses the concept of maintaining both a *local* (or private) view and a *global* (or public) view of the object under consideration. At any time, a user can adjust his/her local view and a token passing technique allows a user to perform a global update when he/she has the token. This approach is described in more detail in section 2.1.
- *The interface must be easy-to-use*: CEV is implemented as an Java applet which runs directly through each user's favorite web browser. Because of the widespread use of the WWW, this provides a very intuitive user-interface.
- *In order to protect important data, the system should have an authentication process*: Currently, CEV is implemented using a simplistic authentication technique. The server executes on the desired machine under a very specific name. If a user wants to participate in a the session, he/she must specify this name exactly in a URL format. In the near future, we will enhance the mechanism by adding a password validation scheme.
- *Support a database that stores/retrieves the history of the session*: At present, we store the results (in the form of images) as flat files. These files are stored for only a limited time. We plan to replace this by the use of a back-end database.

2 Operation and Methodology

The motivation leading to the development of CEV is driven by a desire to provide an interactive collaborative tool for visualizing large data sets (or volumes) to scientists who wish to use inexpensive workstations. To accomplish this goal, we must minimize the requirement of special graphics hardware and parallel machines. Our approach is based upon several client machines connected to a single server. The common server is required to be a high-end graphics workstation while the client machines need only be capable of executing a web browser and the associated Java environment.

2.1 Operation

The participants (possibly geographically distributed) use their personal computers and any standard browser to get involved in the collaboration. The interface for CEV (as seen from the browser) is shown in Figure 1. Each researcher has a *local* (or private) view, a *global* (or public) view, and a *list of the current participants'* session name. The session name, specified by a parameter within the .html file used to load the applet, represents each individual participant.

While using CEV, a researcher has several possibilities for interaction. At any time, the user may use the left mouse button to adjust his/her local view of the given scene by clicking, holding, and dragging the mouse within the local window. A click with the right mouse button will reset the viewer to its default position. These moves do not affect other participants and allows the user to investigate the scene without collaboration, i.e. in a stand-alone mode.

The collaboration is achieved through the global view. A token passing approach using the principles of *round-robin* scheduling is used to give each participant a chance to broadcast his/her local view. An interval, specified as

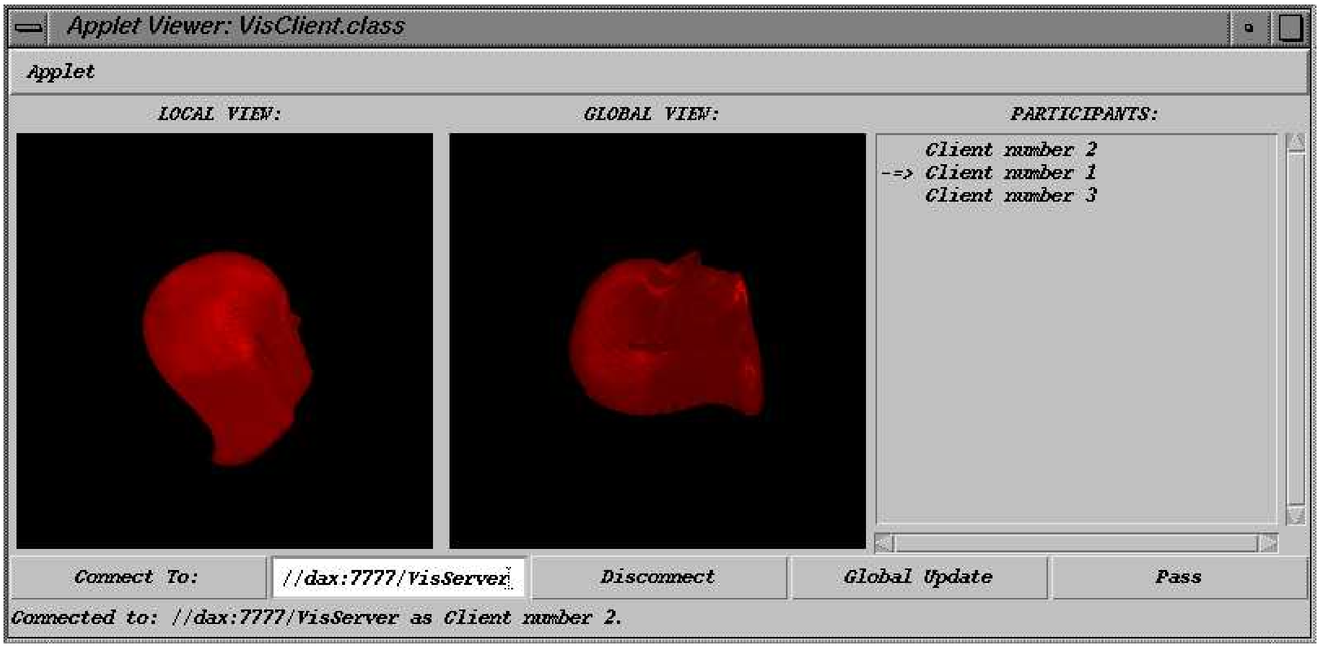

Figure 1: CEV User Interface

a parameter during server initialization, is used to indicate the maximum amount of time a participant gets to keep the token. If this interval expires, the token is passed to the next participant. The client holding the token may use the *global update* button to broadcast to all other members of the current session his/her local view. The result of this broadcast will be the updating of all participant's global views to that of the token holder's local view. The round-robin queuing approach gives each user the same amount of time to update the others' view. The *pass* button allows a participant who does not wish to broadcast the chance to skip his/her turn, resulting in a more effective collaboration.

2.2 Methodology

To eliminate the need of a powerful machine for each client, the server performs all graphical computations. If a local-view update is requested, the client transmits only its current viewer position and angles, and possibly rendering and operation parameters such as color or transparency. The server will then calculate the new image and store it on its disk. The client machine receives the URL of the image calculated. It is the responsibility of the browser to download and update the image within the applet.

If a participant requests a global update, his/her machine only needs to send the URL of his/her current local view. Because this image has already been computed and stored on the disk of the central server, all other clients need only to fetch a single two-dimensional image. The communication required for such an update is common when browsing the web and hence acceptable to most people. This approach has the advantage of transmitting a minimal amount of data over the network and allowing for very efficient global updates.

By performing all graphical computations and image generation on the server, the participants can use significantly cheaper machines. The familiar browser interface provides an easy interaction and gives each participant the feeling of having their image computed locally, but doesn't require the memory typically needed for large data set rendering. The next section provides details of the various Java classes and specifics required for implementation. Section 4 discusses performance issues related this model of collaboration.

3 System Implementation

As previously stated, CEV is implemented primarily in Java because of its ability to easily handle heterogeneous machines. RMI is used to handle network communication between the clients and the server. Because of the

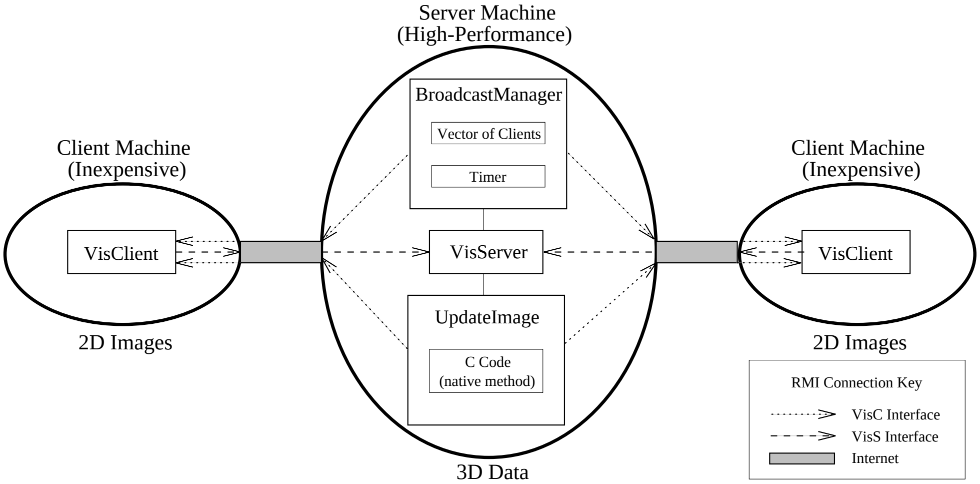

Figure 2: CEV System Implementation

computational intensity of image generation and the existence of libraries written in the C programming language, the server uses native methods to perform all graphical computations. In addition, the server is multi-threaded, as will be described shortly.

Figure 2 provides a diagram of the system. The four classes in Figure 2, `VisClient`, `VisServer`, `BroadcastManager`, and `UpdateImage`, compromise the core of the system. Other classes (not described in detail) include the `Timer` class used by `BroadcastManager` which serves as an alarm for token passing and the `PicBox` class used by `VisClient` to handle the images within the applet. The interfaces, `VisS` and `VisC`, define methods which are invoked either on the client machine or the server. They are depicted in the figure to indicate the direction of the communication.

• `VisClient`: `VisClient` is the applet code for all client machines. It is responsible for all user interactions and managing of the display. As indicated in Figure 2. `VisClient` invokes remote methods on `VisServer` while also receiving invocations from `BroadcastManager` and `UpdateImage`. `VisClient` invokes `VisServer` when its user requests a view adjustment. Once the new image has been computed, the applet is notified by `UpdateImage`. At that time, the browser downloads the new image. The communication from `BroadcastManager` to `VisClient` is used to update all public information such as a new participant entering the session, an existing participant exiting, an exchange of token, or the updating of the global view.

• `VisServer`: The `VisServer` object runs as an application and is only invoked by a `VisClient` object. The role of `VisServer` is to direct the request to the corresponding handler, `BroadcastManager` or `UpdateImage`.

• `BroadcastManager`: `BroadcastManager` resides within `VisServer` and is responsible for sending public session changes to all connected clients. To effectively handle many clients, `BroadcastManager` is multi-threaded. When a new participant connects to `VisServer`, `BroadcastManager` creates and maintains a separate thread which is responsible for actually sending updates dictated by `BroadcastManager`. `BroadcastManager` also has a separate thread, `Timer`, which notifies it of token interval time-outs. The `Timed` interface implemented by `BroadcastManager` is used by `Timer` as a *callback function* and is not related to RMI in any way.

• `UpdateImage`: Similar to `Timer`, `UpdateImage` also extends the `Thread` class. `UpdateImage` objects are created from `VisServer` and are only responsible for calling the native method, which will perform the necessary image generation. Its implementation as a thread is important for concurrent execution. The server machine should be as powerful as possible and ideally a parallel machine. Each thread could (theoretically) be scheduled on an independent processor.

The native method is written in C which does limit portability; however, we are assuming that the server would only be running on a high-end UNIX workstation. As Java technology becomes further developed, we would like to incorporate Java graphics APIs.

4 Experiences, Future Work & Conclusion

Developing CEV was fairly straight forward. The final prototype resulted in about 1100 lines of Java code, 600 lines of C code, and about 7 classes. Because of the inability of browsers such as Netscape to support RMI programs, CEV has only been tested using Appletviewer. The environment has been tested with 1, 2, 4, 6, and 8 participants using Sun Ultra and Sparc machines that were connected via Ethernet to a Sun Ultra 1, acting as the central server. As an initial prototype, we chose to use a homogeneous environment, but certainly plan to use PC's in the near future. Global image updating and user interaction have been very efficient. Because of the multi-threading capabilities of the server, these operations are handled concurrent. While global images are updated quickly, local image requests from multiple clients tend to cause a bottleneck on the server.

Preliminary timing experiments performed on the system with a varying number of clients indicated an increase in local image updating time of about 30 seconds for every 2 clients added to the session. Because of the application, volume visualization, a typical image may take as long as 10 seconds to compute (without collaboration); therefore, collaboration does not add a lot of overhead. The performance can be improved by either limiting the number of clients or purchasing a parallel workstation for the server.

There are many possibilities for enhancements. Currently, CEV lacks a few factors which are available with other collaborative systems. These include an audio/video conferencing system, a session manager that records the session and stores it into a database, and a rigid authentication process. The current control mechanism for collaboration is similar to the round-robin algorithm. The next version will involve the addition of many more features. The various control mechanisms will be selectable by users via the standard interface. In addition to the collaborative extensions, we would like to use other application areas within visualization and incorporate better rendering algorithms which will shorten the time necessary to compute an image.

This paper described both the methodology and implementation of a new collaborative system for visualization called CEV. CEV uses Java and RMI to support a heterogeneous environment through a standard web browser. The interface consists of two views, *local* and *global*, of the current scene and a list of the session participants. The local view is used to investigate privately while the global view serves as a means to collaborate visual ideas. In order to eliminate the requirement of high-end graphics workstations for each individual user, CEV uses an innovative technique that performs all graphics computations on a single server and transmits only the final image to be displayed thus achieving a clean separation between the 2D and 3D components. This technique and its implementation proved to be very effective for a small number of participants.

References

[1] Arie Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1991.

[2] Vinod Anupam and Chandrajit Bajaj. Shastra: Multimedia Collaborative Design Environment. *IEEE Multimedia*, 39-49, 1994.

[3] T. Crowley et al. MMConf: An Infrastructure for Building Shared Multimedia Applications. *Proceedings ACM CSCW 90*, ACM Press, New York, 1990, pp.21-37.

[4] S. Ahuja, J. Ensor, and D. Horn. The Rapport Multimedia Conferencing System. *Proceedings ACM SIGOIS*, ACM Press, New York, 1988, pp.1-8.

[5] E. Craighill et al. Environments to Enable Informal Collaborative Design Processes. *Proceedings First Workshop on Enabling Technologies for Concurrent Engineering*, Morgantown, West Virginia, 1992, pp. 47-51.

[6] Alex Pang and Craig M. Wittenbrink. Collaborative Visualization with CSpray. *http://www.cse.ucsc.edu/research/slvg/cspray.html*, 1997.

[7] Jason Wood, Helen Wright, and Ken Brodlie. Collaborative Visualization. *Proceedings of the IEEE Visualization 97*, Octobeer 19-24, Phoenix, Arizona, 1997.

[8] Lukasz Beca, Gang Cheng, Geoffrey Fox, Tomasz Jurga, Konrad Olszewski, Marek Podgorny, Piotr Sokolowski, and Krzysztof Walczak. Web Technologies for Collaborative Visualization and Simulation. *NPAC Technical Report SCCS-786*, January, 1997.

[9] Gary Cornell and Cay S. Horstmann. *Core JAVA Second Edition*. Sunsoft Press, 1997.