1993

Distributed and Collaborative Volume Visualization

Vinod Anupam

Chandrajit Bajaj

Daniel Schikore

Matthew Schikore

Report Number:

93-050

DISTRIBUTED AND COLLABORATIVE VOLUME VISUALIZATION

Vinod Anupam, Chandrajit Bajaj,
Daniel Schikore and Matthew Schikore

CSD-TR-93-050
August 1993

Distributed and Collaborative Volume Visualization *

Vinod Anupam Chandrajit Bajaj Daniel Schikore Matthew Schikore†

Department of Computer Sciences

Purdue University

West Lafayette, IN 47907-1398

{anupam,drs,bajaj,mcs}@cs.purdue.edu

Tel: 317-494-6531, FAX: 317-494-0739

Abstract

We describe the design and implementation of a distributed volume rendering algorithm in our distributed and collaborative software environment, called SHASTRA. The algorithm uses the computational power of multiple networked workstations to speedily produce translucent shaded images of extremely large volume data sets. The color graphics in SHASTRA are built on top of XS, a machine independent 3D-graphics and windows library, that runs on multiple platforms in a heterogeneous environment. We also describe a synchronously conferenced environment that we have built in SHASTRA to support collaborative visualization, allowing multiple users to share and interact over a volume data set while viewing multiple renderings with independent viewing directions, cutaways, shading parameters, etc.

Keywords

Distributed Volume Rendering, Ray Casting, Collaborative Visualization, Teleoperation.

*Supported in part by NSF grants CCR 92-22467, DMS 91-01424, AFOSR grant F49620-93-10138, NASA grant NAG-1-1473 and a gift from AT&T

†While at Purdue for the summer of 1993 as an NSF-REU student. Current address is Department of Computer Science, University of Iowa, Iowa City, IA 52242

1 Introduction

Interactive volume rendering is compute and memory intensive [5, 8, 3]. Distributed systems, riding on current advances in computer processor and memory technology, and high speed networking, provide a mechanism for effectively harnessing the total computational power of multiple workstations available on a network. We have adopted a hybrid strategy to benefit from distributed systems. Distributing the output of a high computation task emphasizes sharing of resources among applications. In addition, partitioning a high computation task, and distributing it, accords us the benefit of parallelism of distribution. The distributed system, thus, serves as a high-performance multi-user virtual machine for large volume rendering computations and collaborative visualization.

Volume visualization is a very intuitive method for interpretation of volumetric data [8]. Measurement-based volumetric data sets arise from sampling e.g. medical imaging (Computed Tomography – CT, Magnetic Resonance Imaging – MRI, Laser Surface Imaging – LSI), geological and geophysical measurements, 3D scanning, etc [4]. Synthetic volume data sets are generated by computer based simulation and modelling – meteorological and thermodynamic simulations, finite element stress analyses, computational fluid dynamics, molecular modelling etc. Volume visualization provides mechanisms to express information contained in these, typically huge, data sets via images – the challenge, of course, lies in making these images easy to understand. The volume visualization system we describe provides several ways of viewing volumetric data – cross sectional viewing, isosurface reconstruction, and direct volume rendering using ray casting. It provides facilities for interactive control and specification of the visualization process.

Our goal is to depart from traditional single user systems and build computer-enhanced multi-user collaborative scientific visualization and analysis environments. These CSCW environments provide support for collaboration in the problem-solving phase, as well as in the review phase. A synchronously conferenced collaborative volume visualization environment lets multiple users on a network share a volume data set, simultaneously view shaded volume renderings of the data, and interact with multiple views, cutaways and iso-surfaces. In this paper we describe a distributed volume rendering algorithm and the facilities for collaborative visualization, all implemented on top of the distribution and collaboration mechanisms of SHASTRA [1]. We also present specific implementation details together with timing tradeoffs based on data set sizes, number of workstations and network bandwith.

2 Distributed Volume Rendering

We present our approach to the volume rendering problem on a network of workstations and describe an algorithm for volume rendering by distributed ray-casting on a rectilinear volume mesh, using the distribution facilities of SHASTRA.

2.1 Ray Casting

Ray casting is a direct volume rendering algorithm in which sight rays are cast from the viewing plane through the volume, accumulating the effects of sampled data encountered along their paths, see for e.g. [11, 9, 7, 10, 12]. The 'tracing' stops based on some termination criteria e.g. when opaque or visible voxels are encountered, or when some accumulation threshold is crossed. The opacity accumulation is based on a classification of the material which gives rise to the data values, see for e.g., [11, 12].

The basic process is as follows:

For each pixel p in the final image we compute the pixel color in the following manner. There exists a prespecified opacity value α with each voxel.

1. Compute the 3D world space line (ray) which maps to this pixel.
2. Intersect the world line with the volume to be rendered (typically a pyramid frustum, if there are no cutaways) to compute the intersections of the line passing through the volume.
3. Perform a 3D-Bresenham (front (closest) point to back point) to determine the various voxel intersections with the line. The data and opacity values of these voxels are used in the subsequent color and opacity accumulation process.
4. Accumulation of pixel color is most natural back to front. For each voxel intersecting the line, accumulate newrgb $= \alpha * rgb + (1-\alpha) * oldrgb$ giving a result of $\alpha_1 * rgb1 + (1-\alpha_1) * (\alpha_2 * rgb2 + (1-\alpha_2) * (...))$. However this leads to unnecessary work because a nearly opaque voxel near the front cancels out all that is behind it. Instead, accumulation is performed front to back, with the use of a temporary variable *temp* to hold the current value of $(1-\alpha_2) * (1-\alpha_2) * (1-\alpha_2)...$
5. If *temp* reaches very close to 0.0 before casting the entire line, exit and store the value and location of the last point. This is for shading calculations, because we have apparently hit a predominant surface in the volume data (for e.g., when opacity is set high for bone densities).
6. If the surface value stored for pixel p is not zero, and the above color accumulation steps of its adjacent pixels are done, then compute an approximate gradient at that point based on the adjacent surface values stored, and shade the pixel based on this gradient.

2.2 Overview of the Distributed Algorithm

Rendering of volume data by ray casting is very amenable to parallelization because different rays cast into the volume are independent of each other. Distributed parallelism is achieved by partitioning the final image space, and using separate processes to render different parts, by providing them with the appropriate input volume data and visualization control parameters. We first divide the image space into cells in order to distribute the task of rendering. We then compute the portion of the volume data which needs to be distributed in order to render each cell in the image space. Servers are then assigned cells of the image space to be rendered until the entire image is complete. We apply gradient shading image cells using the z-buffer as an approximation to the gradient[6]. As adjacent cells become available, gradient shading calculations for the common edges of cells can be completed. Since we partition based

Figure 1: Distributed Rendering of a 512 by 512 by 113 volume of a skull

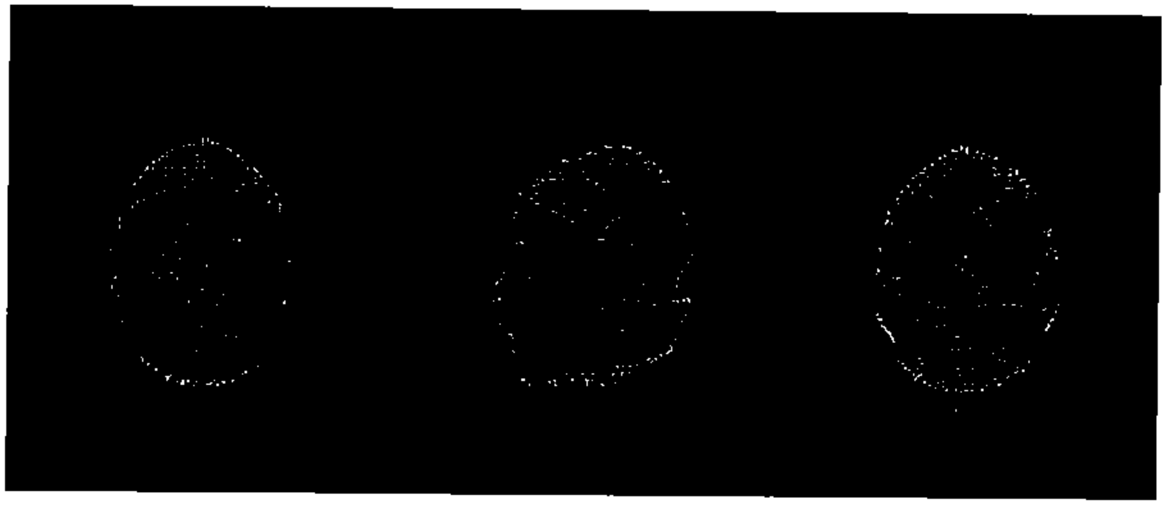

Figure 2: Distributed Rendering of a 512 by 512 by 109 volume of a Human Head Cutaway

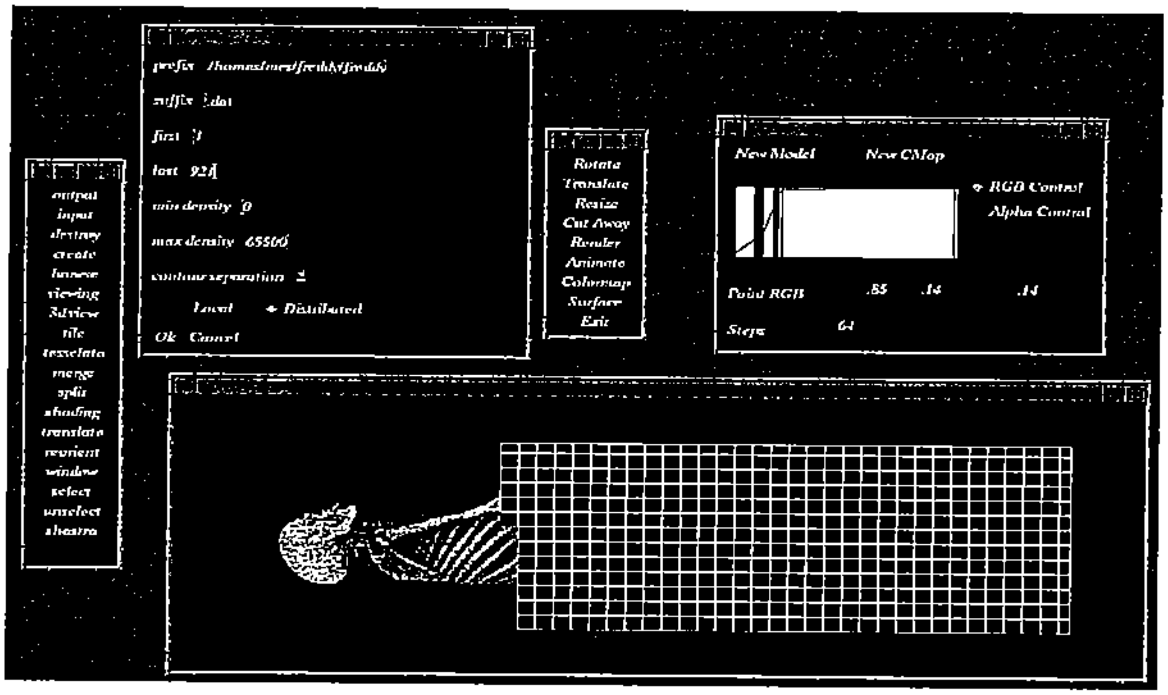
prefix /home/mer/fredh/f/fred/l/
suffix .dat
first 0
last 921
min density 0
max density 65500
contour separation 2
Local [✓ Distributed]
Point RGB .85 .14 .14
Steps 64
[✓ RGB Control] Alpha Control

Figure 3: Image Space Partitioning for Distributed Rendering of a 512 by 512 by 920 Human Cadaver

on the final image space, reassembly of computed information from the servers is trivial, since the pieces of the final image returned by the servers are the needed results. Example renderings of a 512 by 512 by 113 volume of a skull and a 512 by 512 by 109 volume of a cutaway head, produced by the distributed algorithm are shown in Figures 1,2.

The main steps of the distributed algorithm are as follows:

1. Subdivide image space into rectilinear (rectangular and orthogonal to the X and Y axes) cells to be rendered by a server
    - Divide into equal area cells.
    - Adaptively divide image space based on the 'weight' of a cell.
2. Determine the portion of the volume which lies within a cell.
3. If there is an available server, send it the necessary data to render the cell.
4. When results are received from a server, store the colors in a pixmap and the depth values (to be used for shading) in an array.
5. When results of adjacent cells are received, shade the image cell based on the depth values stored.
6. Repeat steps 2 to 5 till all image cells are processed.

2.3 Image Space Partitioning

The first step of the distributed rendering is to partition the image space into regions to be distributed to the various servers. We consider the "weight" of a cell as the amount of volumetric data contained

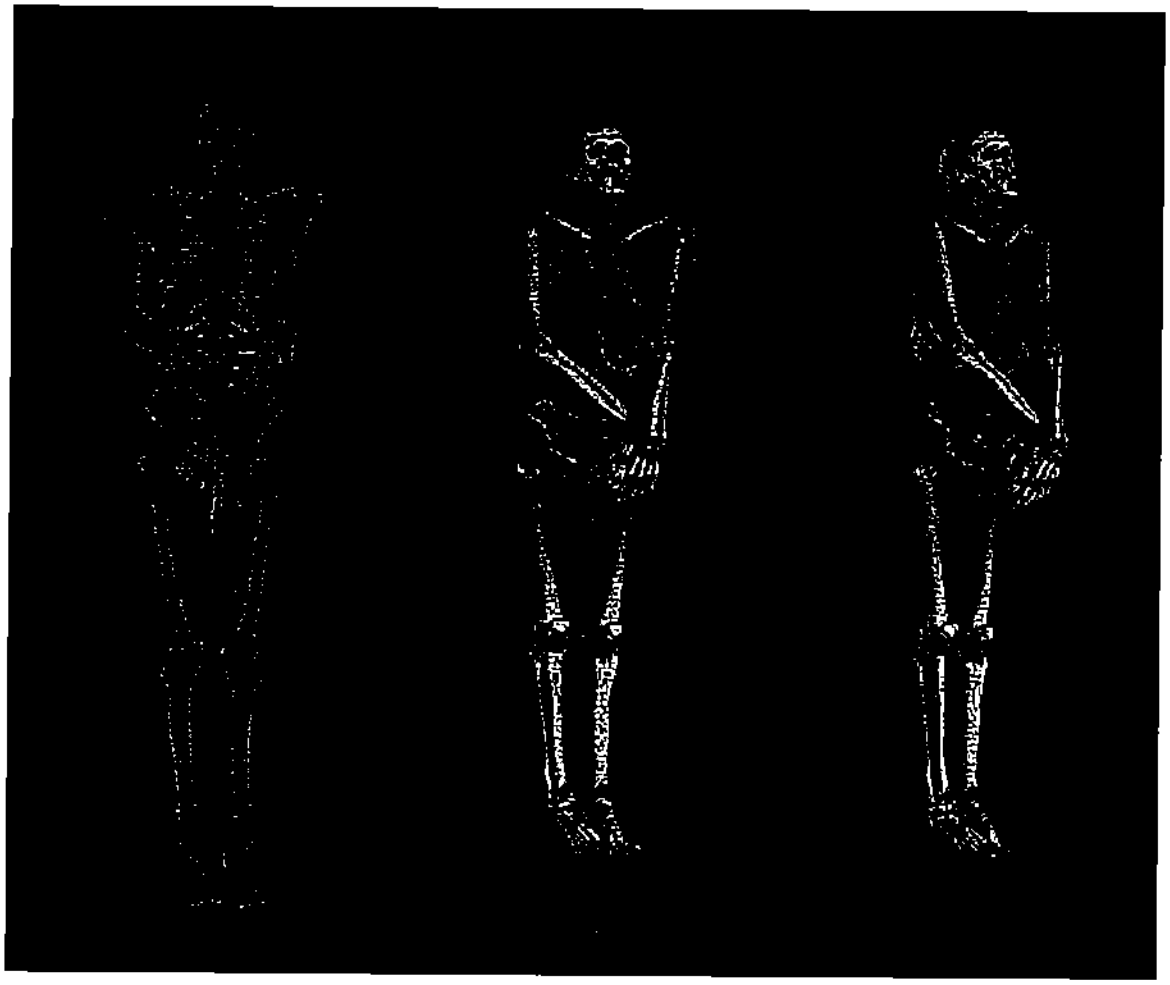

Figure 4: Distributed Rendering of a 512 by 512 by 920 Human cadaver with Different Levels of Skin Transparency

within the volume corresponding to the cell. Volume rendering requires evaluation at every point in the data set, and computation time is thus proportional to this weight. The goals in dividing the image space are to separate the image space into regions of roughly equal weight, and to balance the amount of time needed to transmit data and render image cells such that both the servers and the client remain as busy as possible. Several methods, both adaptive and non-adaptive, have been explored.

In each case, we divide the image space into rectangles which are orthogonal to the screen coordinate system. See Figure 3 which shows the graphical user interface. The easiest static method of defining the cells is to divide the space into squares of equal size. This method is fast but has the disadvantage of not being sensitive to the image size or the amount of data within the volume corresponding to the cell. An adaptive method which takes both of these factors into consideration is the method of repeated subdivision. In this method, we begin with a region the size of the image and subdivide in a quadtree manner based on certain criteria. Using this method, we can guarantee a upper and lower bounds on the amount of data to be sent to a server, and can balance the network transmission time with the server rendering time.

Distributed volume rendered images of a 512 by 512 by 920 MRI data set of a human cadaver, Freddy, are shown in Figure 4. The upper image shows a frontal view with bone viewed as opaque, and other tissues visualized as being transparent, revealing the entire skeletal structure. The lower image shows a dorsal view, with tissues assigned translucency values in relation to their density.

2.4 Input Volume Partitioning

To determine the portion of the data volume which is contained within an image cell, consider the world space within a cell to be the intersection of four half spaces which are defined by four planes bounding the volume from top, bottom, left, and right. We map the corners of the cell to 8 points in world space which define these planes. Each slice in the volume is a polygon in a constant Z plane. We clip this polygon by each of the planes bounding the cell to determine the portion of each slice contained in the cell. Since we send only rectangular portions of slices, we take the bounding box of the clipped polygon as the required data to be distributed.

2.5 The Distribution Infrastructure

All systems designed to run in the SHASTRA environment have certain features which make them amenable to inter-operation. A typical system has an application specific core - the Application Engine - which implements all the functionality offered by the system as a tool or service. On top of the Engine is an Interface Mapper which actually calls upon functionality embedded in the engine in response to requests from the interfaces – ASCII, GUI and Network Interfaces. Inter-system communication occurs via their Network Interfaces. See Figure 5 which shows this architecture in a block diagram. The SHASTRA layer, comprised of the network session and data communication substrates is the ether that joins the network interfaces of various toolkits. This layer provides connection setup and multiple-connection management facilities in the distributed environment. It also implements the SHASTRA data communication protocol for peer to peer communication. The set of connected application-object interfaces of SHASTRA toolkits comprise a distributed virtual machine over which distributed parallel algorithms can be implemented, and synchronous conferences can be conducted,

2.6 Data Communication Optimizations

The data which is sent from client to server consists of rectangular portions of input data slices which contain the region of the slice needed to render one of the partitions of the image space. We take advantage of the fact that the data being transmitted between workstations contains large regions of zero value by run-length encoding only the zeroes in the stream of input data values sent to the server processes. In the worst case, there may be numerous isolated zeroes which would be doubled by the RLE encoding, but in practice with this method we have realized compression ratios of 4:1 to 10:1.

An important factor to take into consideration is the classic compute-communicate tradeoff. Since a large amount of data needs to be moved from the client to multiple servers, the local area network

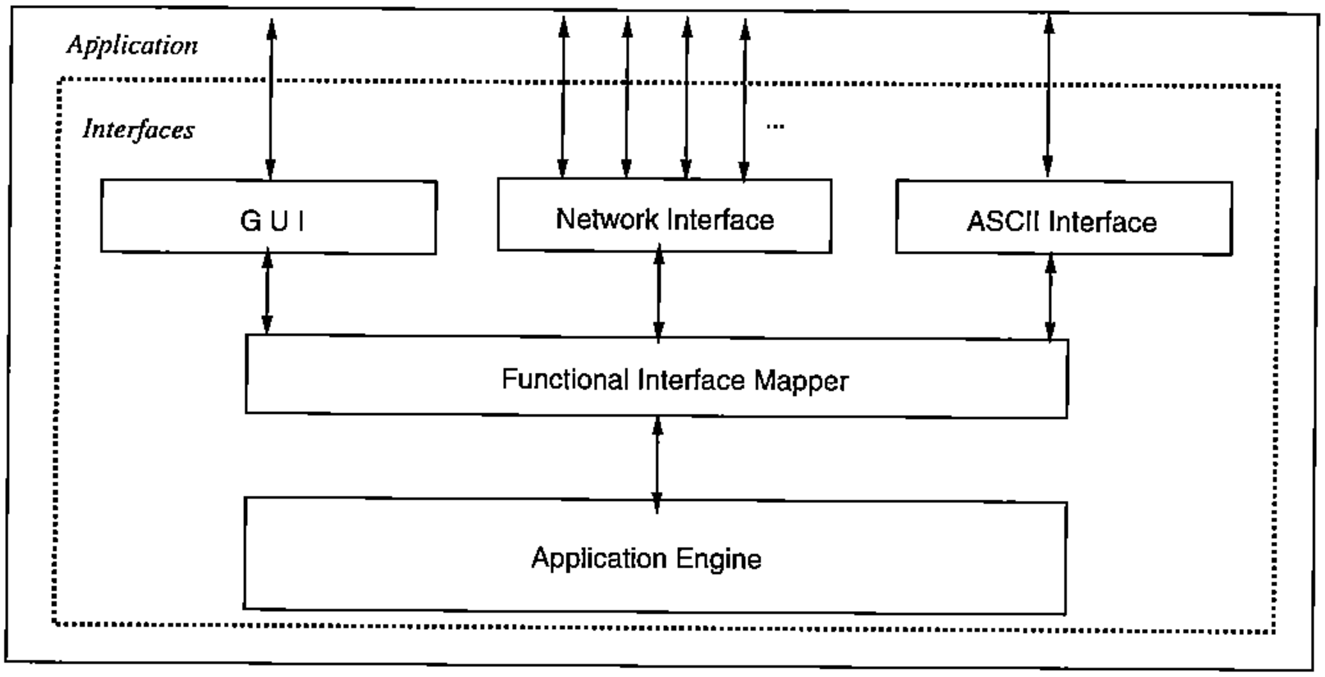

Figure 5: Architecture of SHASTRA Applications

can easily get congested. However, if we do not send a large enough amount of data, communication overhead dominates the total cost of distribution. A balance, therefore, needs to be struck between the two. In practice we have observed the balance to be sensitive to ambient network traffic. Our current system runs on an ethernet (10Mbps), and would benefit greatly from deployment and use of very high bandwidth network technologies like ATM.

2.7 Heterogeneity Issues

In a heterogeneous computing environment, SHASTRA applications achieve hardware independence by building on top of high level abstractions, above the greatest common denominator. We assume the availability of the X Window System (X11R5) for user interfaces. Multi platform development is cumbersome because high performance graphics platforms have different graphics models and APIs. As a solution in SHASTRA, platform independence is achieved by building applications atop abstract libraries which hide hardware specifics. These abstract libraries can be easily extended to standardized interfaces as they evolve.

The XS graphics and windows library was developed to provide a machine independent interface to routines for 3D graphics. The use of this abstraction provides us with source code level portability across multiple platforms, without compromising on speed or quality of graphics [2]. The current suite of libraries supports graphics using X11, SGI/GL, HP/STARBASE and Windows 3.1.

Platform heterogeneity woes in the realm of data representation are obviated by using the SHASTRA protocol for data transport, which uses XDR to encode data in a device independent manner.

3 Shared Work Spaces

SHASTRA is a collaborative multimedia scientific manipulation environment in which experts in a co-operating group communicate and interact to solve problems. The SHASTRA environment consists of a group of interacting applications. Some applications are responsible for managing the distributed environment (the Kernel applications), others are responsible for maintaining collaborative sessions (the Session Managers), yet others provide specific communication services (the Service Applications), while yet others provide scientific design and manipulation functionality (the SHASTRA Toolkits). Service applications are special purpose tools for multimedia support – providing mechanisms of textual, graphical, audio and video rendition and communication. Different tools register with the environment at startup providing information about what kind of services they offer (Directory), and how and where they can be contacted for those services (Location). The environment provides mechanisms to create remote instances of applications and connect to them in client-server mode (Distribution). In addition, the environment provides support for a variety of multi-user interactions (Collaboration). It provides mechanisms for starting and terminating collaborative sessions, and joining or leaving them. The infrastructure is described in detail in [1].

3.1 A Collaborative Visualization Tool

POLY is a 3-D rendering and visualization tool in the SHASTRA environment. New SHASTRA toolkits use POLY as their 3D graphics interface, since it isolates 3D graphics object manipulation, rendering and visualization functionality. POLY provides a variety of mechanisms for visualization of multi-dimensional data. It understands a number of graphical object formats, which it converts to an internal form for efficient display and transport. It has a user interface that supports manipulation of graphical objects. At its network interfaces, POLY interoperates with other SHASTRA toolkits, and provides a very high level abstraction for manipulation of such data. The Motif based GUI is used to manipulate visualized objects in multiple XS graphics windows.

The user interface of the visualization system is shown in Figure 6. The top image is a rendering of the upper torso of Freddy. The skeletal structures are opaque and shaded, while the rest of the structures have been assigned different levels of transparency. The bottom image shows a surface rendering of a human head with a cutaway of the skull to show part of the brain surface.

The SHASTRA environment for collaborative visualization consists of a collection of instances of POLY. A collaborative session is initiated by one of the POLY users in the environment. This user becomes the group leader and specifies to the local Kernel the list of POLY users that will be invited to participate in the session, and becomes the group leader. The Kernel instantiates a Session Manager, which starts a session with the group leader as its sole participant, and then invites the specified users of concurrently executing remote POLY sessions to participate. Users that accept are incorporated into the session. Any POLY instance not in the conference can request admittance, and join. A participant can leave an ongoing session at any time. Users can be dynamically invited to join or removed from

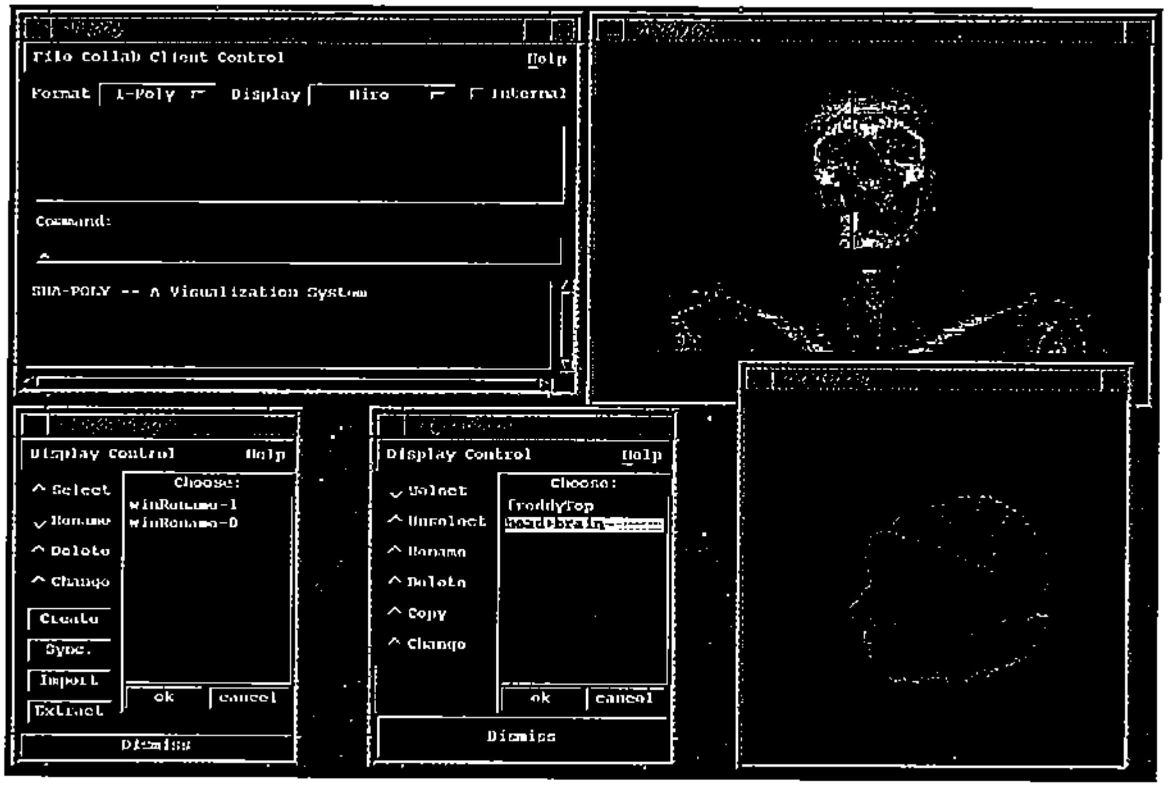

Figure 6: Using the SHASTRA Application called POLY for Collaborative Visualization

conferences by the group leader or his designees.

The hybrid computation model for conferences in SHASTRA consists of a centralized Session Manager for each session, which regulates the activity of multiple instances of POLY. Though this model suffers from problems of scale due to the centralized Session Manager, it performs well for typical group sizes. An important benefit derived from the replication is in the realm of platform heterogeneity – the application instances are responsible for dealing with particular platform idiosyncracies. In addition, since the conference consists of cooperating applications, the notion of private and shared workspace and private and shared interaction is easily supported. The centralization of the Session Manager for a collaborative session accords us the benefit of centralized state. The Session Manager serves as a repository of shared objects. This makes it easy to accommodate late joiners of sessions to come up to date quickly. It also eases the task of serialization of input actions for multi-point synchronous interaction, and constraint management for mutual consistency.

A permissions based regulatory subsystem permits control of data flow at runtime, providing a variety of interaction modes. Collaboration in SHASTRA can occur in the REGULATED (Turn-taking or Master-Slave) mode or in the UNREGULATED (Free Interaction) mode. In the REGULATED mode, users take turns by passing a baton. The collaboration infrastructure of SHASTRA has a two tiered permissions based regulatory subsystem used to control interaction primarily in the UNREGULATED mode. SHASTRA permissions control 'Access' to a view of the conference, local viewing controls to 'Browse' a view, rights to 'Modify' conference state, and rights to 'Copy' shared objects.

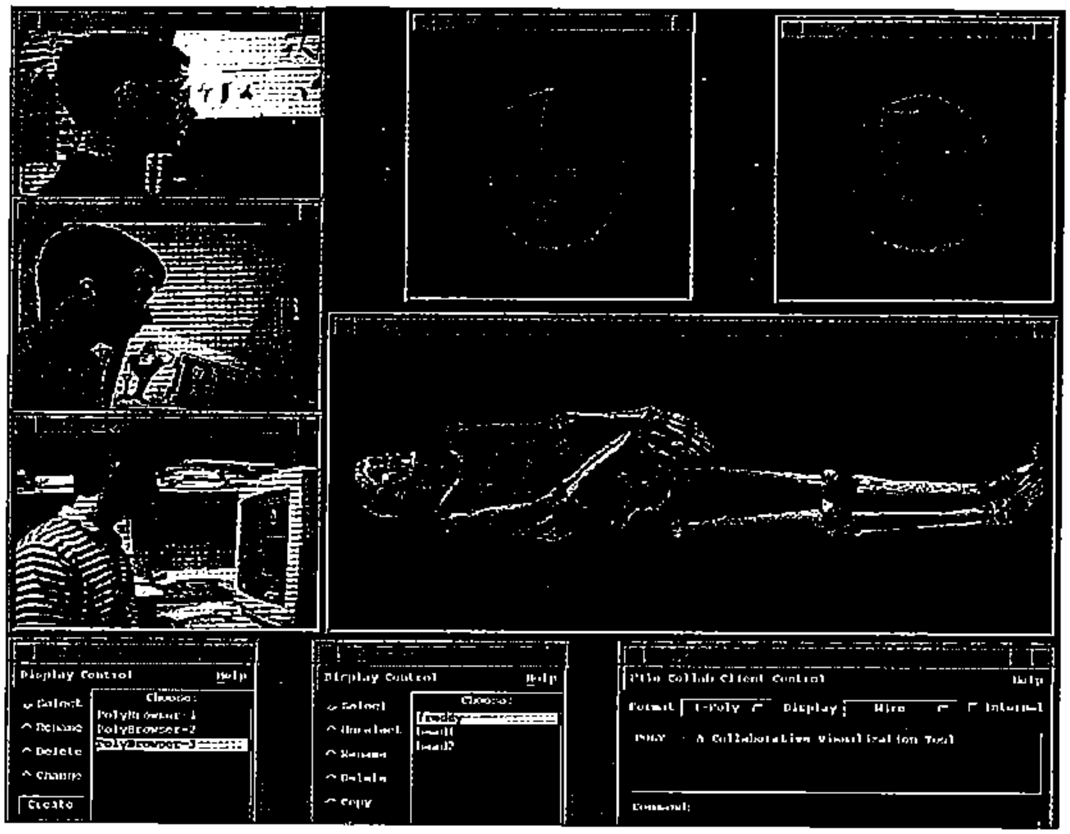

Figure 7: One Site in a Collaborative Visualization

The session manager allows only one user to manipulate "hot spots" in the shared space – where there is a possibility of contention – at any particular instant. It uses the first-come-first-served paradigm to decide which user gets temporary exclusive control. The baton passing facility of the system can be used to take turns to adjust visualization parameters. Alternately, designers can use the auxiliary communication channels – like audio, video, and text by initiating PHONE, VIDEO or TALK sessions – to regulate access, and for arbitration. All operations are performed via the (central) session manager which is responsible for keeping all sites up-to-date, so that the users have a dynamically changing and continuously updated view of the action in the shared windows.

3.2 Collaborative Volume Visualization

Every participating POLY instance creates a shared window in which all the cooperative interaction occurs. Users introduce graphics objects into the session by selecting them into the Collaboration Window. The Session Manager is responsible for providing access to the objects at all participating sites which have the Access permission, and for permitting interaction relevant to the operation at sites which have Modify permission for the collaboration. Collaborating users can twiddle visualization modes and parameters, and adjust viewing modes and direction. The system provides telepointers in the shared windows. It also provides indications of remote presence which describe the viewing location of remote users in the collaborative session. Figure 7 and 8 depict two sites in a three way collaborative

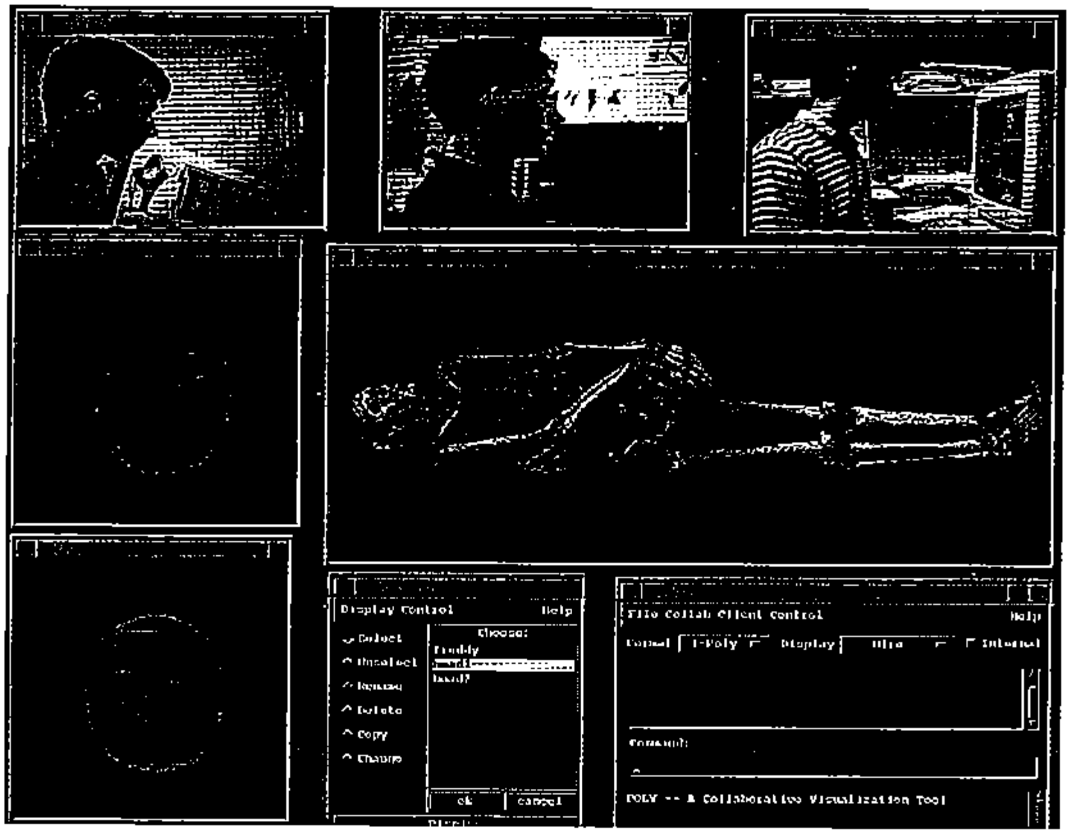

Figure 8: Another Site in a Collaborative Visualization

visualization. The entire rendering of Freddy is shared by all collaborating sites – they share the data set, the viewing location, as well as visualization control parameters. The collaborators share data sets and viewing location in the other two renderings of MRI data sets of the human head. However, they use different cutaways to examine different parts of the data.

At one extreme, the SHASTRA implementation for Collaborative Visualization can be used by a single user to perform scientific visualizations, just like in a non-collaborative setting. Allowing other users to join the session with only Access and Browse permissions sets up the environment like an electronic blackboard to teach novice users the basics of the process. An appropriate setting of collaboration permissions and turn-taking can be used to allow hands on experience with the task. In conjunction with the audio and video communication services of SHASTRA, this becomes a powerful instructional environment. Collaborative sessions using POLY are a valuable tool for review and analysis of problem solutions. Multimedia communication facilities permit a rapid exchange of rationales for choices, interpretations of analyses and iterative improvement.

4 Implementation Details, Conclusions and Future Work

We have used the distributed and collaborative environment to render large data sets efficiently. Our computing environment for the distributed rendering tasks consisted of an IRIS Indigo R4000 and Sun

4/50 (Sparc IPX) workstations with 32Mb RAM, linked by a 10Mbps ethernet. The data sets used are stored on remote file systems, and are accessed through NFS. Preliminary measurements of total time taken to render different volumetric data sets (made during conditions of normal network traffic) are very encouraging, and indicate that there is close to a linear speedup achieved by using multiple workstations to render large volume data sets. (All numbers are whole seconds of real time the user has to wait before the final image is available. Note that this includes network latencey, process swapping, and NFS access time.) The skull data set (512 x 512 x 113 short integers, 56.5 Mb) was rendered on an Indigo in 255 seconds. Using only two remote servers, the rendering took 195 seconds, and using only four remote IPX servers it took 165 seconds. Corresponding times for the head data sets (512 x 512 x 109 short integers, 54.5 Mb) are 330 seconds, 300 seconds and 260 seconds respectively. Freddy, the cadaver data set (512 x 512 x 920 short integers, 460 Mb), took 2340 seconds on an Indigo, 1640 seconds using only two IPX servers, and 1390 seconds using four IPX servers.

We are currently fine tuning the distributed algorithm from the point of view of memory access patterns to minimize swapping, and distributed data size to reduce network traffic and congestion and load balancing to get better performance. Also, currently, multiple distinct views of the same volume data set are rendered separately. We are adding an optimization that can be made when two views differ only in the cutaways. We can identify the parts of the final image that are common to both views, and render them only once. We are exploring other partitioning and distribution strategies in order to further improve the speed of rendering. Specifically, we are in the process of integrating the distributed rendering algorithm with the brokering and load balancing facilities of SHASTRA, to make optimal use of network computational power.

Acknowledgements We are grateful to the of research centers of Johns Hopkins University (E. Fishman and D. Ney) for the human cadaver data set Freddy, SUNY Stony Brook (A. Kaufman) for one of the human head sets, and the University of North Carolina for anonymous ftp access to the skull and the other human head data set.

References

[1] V. Anupam and C. Bajaj. Collaborative Multimedia Scientific Design in SHASTRA. In *Proc. of the ACM Multimedia'93 Conference*, pages 447–456. ACM Press, 1993.

[2] V. Anupam, C.Bajaj, A. Burnett, M. Fields, A. Royappa, and D. Schikore. *XS: A Hardware Independent Graphics and Windows Library* . Computer Science Technical Report, CAPO-91-28, Purdue University, 1991.

[3] B. McCormick, L. DeFanti and M. Brown. Visualization in Scientific Computing. *Computer Graphics*, 21:complete issue, 1987.

[4] B. Collins. Data Visualization. In R. Martin, editor, Directions in Geometric Computing, pages 31 – 80. Information Geometers Press, 1993.

[5] E. Farrel and R. Zappulla. Three dimensional data visualization and biomedical applications. *CRC Critical Reviews in Biomedical Engineering*, 16(4):323–363, 1989.

[6] D. Gordon and R. Reynolds. Image Space Shading of 3-Dimensional Objects. *Computer Vision, Graphics, and Image Processing*, 29:361–376, 1985.

[7] J. Kajiya and B. Vol Herzen. Ray Tracing Volume Densities. *Computer Graphics*, 18:165–174, 1984.

[8] A. Kaufman. *Volume Visualization.* IEEE Computer Society Press, New York, 1990.

[9] L. Harris, R. Robb, T. Yuen and E. Ritman. Non-invasive Numerical Dissection and Display of Anatomic Features. In Recent and Future Developments in Medical Imaging. SPIE, 1987.

[10] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[11] R. Drebin, L. Carpenter and P. Hanrahan. Volume Rendering. *Computer Graphics*, 22:65–74, 1988.

[12] P. Sabella. A Rendering Algorithm fro Visualizing 3D Scalar Fields. *Computer Graphics*, 22:51–58, 1988.