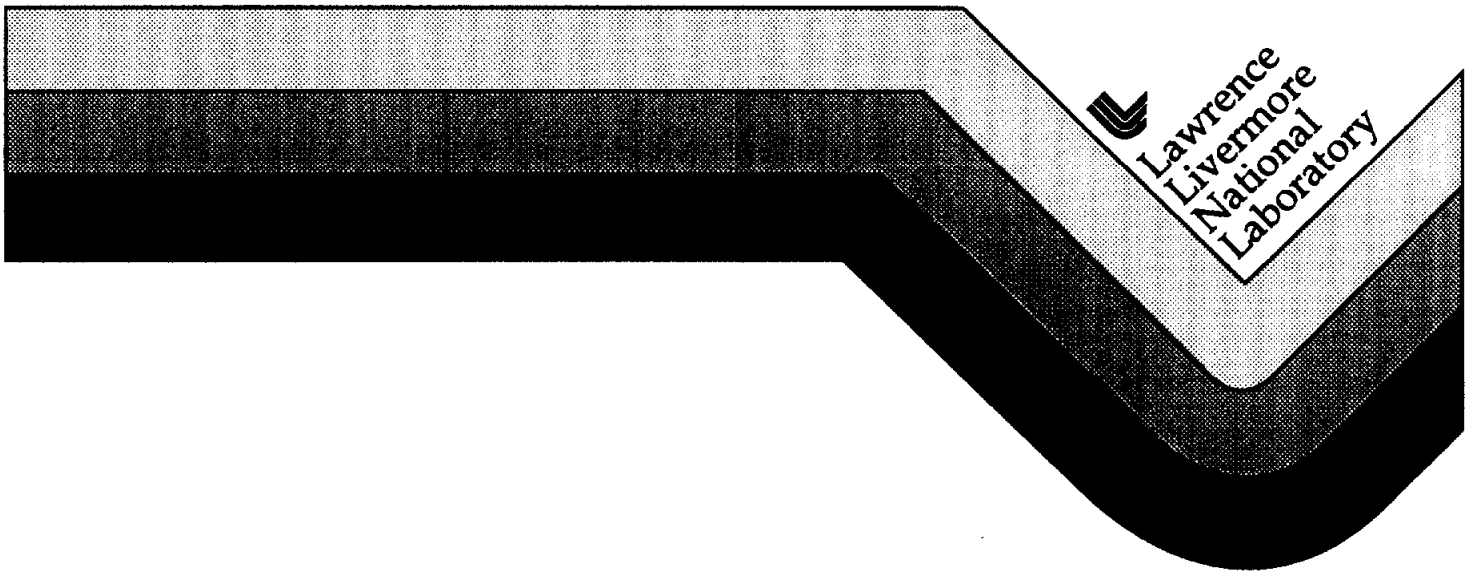
UCRL-CR-126386
S/C#291836

Perm Web: Remote Parallel and Distributed Volume Visualization

C.M. Wittenbrink
K. Kim
J. Story
A. Pang
K. Hollerbach
N. Max

January 1997



DISCLAIMER

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

PermWeb: Remote Parallel and Distributed Volume Visualization

Craig M. Wittenbrink
Baskin Center for Computer Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064

Kwansik Kim, Jeremy Story
and Alex Pang
Baskin Center for Computer Engineering and Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064

Karin Hollerbach and Nelson Max
Institute for Scientific Computing Research
Lawrence Livermore National Laboratory
Livermore, CA 94550

In this paper we present a system for visualizing volume data from remote supercomputers (PermWeb). We have developed both parallel volume rendering algorithms, and the World Wide Web software for accessing the data at the remote sites. The implementation uses Hypertext Markup Language (HTML), Java, and Common Gateway Interface (CGI) scripts to connect World Wide Web (WWW) servers/clients to our volume renderers. The front ends are interactive Java classes for specification of view, shading, and classification inputs. We present performance results, and implementation details for connections to our computing resources at the University of California Santa Cruz including a MasPar MP-2, SGI Reality Engine-RE2, and SGI Challenge machines. We apply the system to the task of visualizing trabecular bone from finite element simulations. Fast volume rendering on remote compute servers through a web interface allows us to increase the accessibility of the results to more users. User interface issues, overviews of parallel algorithm developments, and overall system interfaces and protocols are presented. Access is available through Uniform Resource Locator (URL) http://www.cse.ucsc.edu/research/slvg/.

Key Words and Phrases: permutation warping, load balancing, interactive viewing, remote frame-buffer viewing, Java, World Wide Web, parallel volume rendering.

1 INTRODUCTION

Visualization of simulated and collected volumetric data is important in many fields. Recent examples of the data sets used for research and development include the Visible Human[1] and global ocean circulation data. Much of the research in visualization has been made accessible by providing immediate accessibility to tools and data through the World Wide Web (WWW). Others have investigated the interfacing of renderers through

the WWW,[2] and other applications in graphics investigating dissemination and interactivity such as virtual frog dissection[3] and immersive virtual worlds.[4] The development of WWW tools is an interesting topic. Card et al.[5] investigate limitations and current capabilities of the WWW. We are specifically interested in investigating and developing tools for the following capabilities:

1. remote viewing by experimental scientists to compare trabecular bone strength or architecture findings with measurements.
2. remote searching by scientists to find the closest visual sample match to web published data for selection of reasonable material properties.
3. remote evaluation by surgeons and radiologists of patient data, for diagnosis or pre-operative scenarios.
4. effective, rapid demonstration of finite element analysis of trabecular bone and other simulations.

Volume rendering algorithms provide a realistic, complete, but computationally expensive means of visualizing sampled 3D data such as those found in medical imaging and computer simulations. Rendering is the creation of images from models, whether those models are geometric or volumetric. By rendering slices and volumes of data, and changing rendering parameters, medical imaging allows searching for different tissues or abnormalities.

Our work involves integrating a parallel renderer with the finite element studies of human joints at the Lawrence Livermore National Laboratory (LLNL).[6–10] Such tools for visualization will make the validation, and refinement of the model simpler, and the demonstration of the results faster and easier. Our collaboration with LLNL has allowed us to use up-to-date data sets where we have proven their efficiency.

We are researching a combination of strategies based on tiling and permutation warping for efficient memory access and data communication that takes advantage of the massively parallel computational resources, or hundreds to thousands of processors. We have (a) extended this strategy to data dependent optimizations for massively parallel calculations, (b) provided remote control through the World Wide Web (WWW), (c) and applied these tools to the task of visualizing trabecular tissue from finite element simulations. The combined use of parallel permutation warping with WWW access results in our system name, PermWeb.

We have extended our permutation warping techniques[11] to take advantage of data dependent coherency for large speedups. Speedup studies have been done at the University of California, Santa Cruz using the MasPar MP-2 4096 node machine. Our volume rendering speedups through efficient network and memory accesses have been published,[11–13] but their utility has not been widely seized upon. Our volume rendering algorithm uses permutation warping to achieve linear speedup and linear storage on parallel machines. The algorithm supports arbitrary view directions, large data sets, large parallel machines, and high order filters, combined features not supported by other data parallel algorithms. Extensions to the algorithm make it superior in speed, while supporting superior filtering qualities to parallel variants of the shear warp factorization algorithm. The ability to do load balancing, culling, and adaptive ray termination are possible with the massively parallel MIMD (multiple instruction stream multiple data stream) and SIMD (single instruction stream multiple data stream) machines.

We developed remote frame viewing solutions, in order to effectively use the remote supercomputers at Santa Cruz. For wider availability of limited high end computational resources, we have provided a WWW interface for users to request volume renderings of 3D data sets. This interface allows users to specify viewing parameters. In future implementations we are considering providing web interfaces for setting transfer functions. The web tools can also be a vehicle to allow remote steering of simulation together with visualization of intermediate results.

Fast volume rendering on remote compute servers through a web interface allows us and the scientific community in general to improve the way data are analyzed and visualized. On-line rendering of simulations is important to reduce/remove time lost waiting for the results of flawed simulations, and to promote faster understanding of correct results. Together with the WWW interface we hope for as wide availability of the tools

and research results as possible. In this paper we present the software architecture for PermWeb, describe the necessary components, and also discuss implementation details and tradeoffs. We also discuss a variety of user interfaces, and performance issues for PermWeb. We briefly describe our parallel volume rendering algorithms, project methodology, motivating application, and system.

1.1 Background: Volume Rendering and Permutation Warping

Volume rendering is a class of algorithms that compute the interaction of light in a volume of light-scattering-particles. For a more in depth discussion of volume rendering see Blinn,[14] Kajiya and von Herzen,[15] and Levoy.[16] The final output is a 2D array of pixel intensities which can be calculated in many different ways indicated by the numerous input variables: volume data, light sources, view transform, classification function, and shading function. An illustrative categorization of possible algorithms is by viewing transform. The viewing transform converts the initial shading intensities and the opacities to the three dimensional screen space by resampling. Existing parallel algorithms may be grouped into four categories determined by their viewing transforms: backwards, multi-pass forwards, forwards splatting, and forwards wavefront. A backwards viewing transform is ray tracing, where the eye point is transformed back into the object space to determine the spatial positions within the data space that affect the view ray. Nieh and Levoy,[17] and Yoo et al.,[18] have developed backwards (ray tracing) volume rendering algorithms for parallel computers. Multi-pass forwards algorithms transform in the opposite direction, and move data from the object space to the screen space, for example as is done in the shear warp algorithm.[19–21] Forwards splatting takes each voxel, and contributes their results to the screen, and though highly parallelizeable has artifacts from out of ordered compositing.[22] Forwards wavefront parallel algorithms are similar to the multi-pass forwards algorithm, but often order the data in a more structured fashion, such as that used in the line drawing algorithm.[23] Our permutation warping[24] approach computes a backwards mapping algorithm with optimal storage and deterministic communication on shared or distributed memory machines.

Permutation warping is essentially a processor assignment technique that provides a general approach for efficient parallel transform algorithms. Permutation warping is better than prior parallel algorithms because it is memory efficient, processor efficient, general, and accurate. The algorithm calculates a volume rendering output, and gives specific memory layout and communication requirements necessary for the exclusive read exclusive write parallel random access machine (EREW PRAM). Processors are assigned sample points, requiring thousands to millions of processors. There are schemes for efficient virtualization to fewer processors as well.

Our algorithm consists of the following three steps:

1. Processors classify and shade, reading neighboring data as necessary.
2. Each processor resamples the opacities, and intensities, to be aligned with the view rays. If done in a straight forward fashion this would require many rounds of communication, but we have developed a permutation warp that requires only one communication.[13] We resample in the object space (OS) near where the points lie, and then send the resampled data to their screen space positions. The method uses a rule that calculates processor assignments for the viewing transform as we initially investigated in Wittenbrink et al.[13] Figures 1 and 2 illustrate the transforms calculated by the processors. Figure 1 shows virtualized permutation warping. The rotated screen space view is overlapped with the non-rotated object space (left). Subimages are computed and then communicated to the aligned screen space (middle), and the sorted subimages are shown in bold in preparation for final compositing (right). Figure 2 shows the upright cube on the left as the original data space, and the rotated The machine is visualized as processors with object space assignments (left) and processors with screen space assignments (right). The object space processors are connected to the right, where the screen space is the upright cube, and the original object space is rotated in relation to the screen. The lines drawn from left to right show communication required to permute the data to the appropriate processors in a provable one-to-one single communication step.

A processor does permutation warping by:

(a) Calculating processor assignments;

(b) Calculating the reconstruction point;

(c) Performing resampling and reading the values of its neighboring processors; (The number of neighbors used determines the filter order.)

(d) Sending resampled values to screen processors. Figure 1 shows the aligned subimages in two-dimensions. Figure 2 shows view spaces in three-dimensions, where lines are drawn from a processor, and the processor it is communicating to. The calculated view is the rotated cube, and processors in the center of the cube do not communicate, as there is a small angle of rotation for this example.

3. A parallel product evaluation combines resampled intensities and opacities. Binary tree operations are used, similar to a parallel product, for the associative compositing.

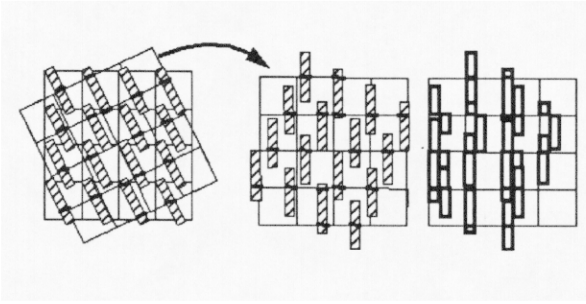


Figure 1: Virtualized permutation warping.

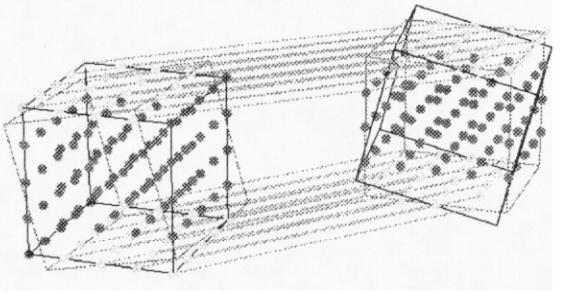


Figure 2: Permutation communication.

1.2 Project Methodology

This research project encompasses three areas: parallel algorithms research, system development, and distributed software architecture research. This paper is an overview of the project, and we briefly discuss here the methodology for the algorithm, system, and architecture research. The parallel algorithm research methodology is to incorporate the advantages of data dependent optimizations into our scalable algorithms. There are more recent sequential algorithms that suggest immediate algorithmic improvements. The methodology for the systems development has been to prototype functioning systems for the driving applications and to optimize and tune the performance. There is more detail of the system in Sections 3 and 4. The methodology for the software architecture development has been to abstract the lessons learned from the prototyping, and to also try to anticipate newer developing technologies. The software architecture is described further in Section 3. To provide full discussion of the project we mention briefly here the approach and developments of the parallel algorithms research.

Several enhancements are possible for the permutation warping algorithm: load balancing, region culling, and local sub-cube adaptive termination. These enhancements are the subject of another paper.[25] Our prior algorithmic studies of permutation warping have shown that it is time and space optimal for resampling on the EREW PRAM.[11–13,24] The MIMD study that we carried out showed that there was poor load balancing for certain view angles. An effective load balancing strategy is a must for MIMD variants of the permutation warping algorithm, and our current work includes altering the mapping of permutations to virtual addresses which allows the use of slackness, and also redistribution of processor's work for SIMD and MIMD. The permutation assignment can be done from virtual processor address to virtual processor address, and the virtual assignments can be dynamically altered to load balance the processing as it occurs.

In related work the shear warp factorization algorithm,[19] has been able to reduce the amount of work necessary in computing a volume rendered output from regular volumes by nearly an order of magnitude. Parallel versions of this program have been presented,[21] and the primary speedup is through straight forward parallelization of the sequential algorithm. But Lacroute et al.[21] and other studies[20] show that the possible speedup to higher numbers of processors is limited because of the decomposition. Permutation warping can scale further as we demonstrated on the SIMD implementations, and it is also possible to use culling which has improved the shear warp approach. Culling through octree encoding, and other compression schemes has been investigated, using the parallel permutation warping algorithm yielding improvements of up to 400%.

Because in permutation warping, each subvolume is rendered as if it were a separate volume rendering job, some amount of adaptive termination along a ray may be performed. In addition, as the subvolume's contributions are computed, an amount of work may be terminated by either doing front to back parallel evaluation, or through communication of termination signals, once accumulated opacity has reached the threshold set. The adaptive termination in the subvolume is a straightforward addition, while the adaptive termination across subvolumes can be done in a variety of ways. The improvement is expected to partially depend on the architecture. Future research shall generalize the tiling and permutation warping for rendering on other supercomputers as well as continue to examine additional data performance enhancements.

2 SCIENCE MOTIVATION

The development of tools for visualization is most useful when driven by true application requirements. The science motivation for our remote rendering tools is to assist in scientific evaluation for finite element studies. Our driving application is to investigate trabecular bone structures. We briefly describe our application, and the questions that are being investigated to elucidate the system requirements and decisions in the following section.

Many bones have an outer layer of compact cortical bone and an inner core of porous cancellous (or trabecular) bone. Although its specific architecture can vary significantly, trabecular bone has a cellular structure with space between the network of bone trabeculae. Variation in the architecture of the trabecular bone depends upon loading conditions of the bone as well as health of the individual. Trabecular bone is always less dense than is cortical bone, with volume occupancy of less than 70% typically being defined as trabecular. Mechanical properties depend on overall bone density as well as specific architecture and strength of the existing trabeculae. In healthy subjects, bone density increases with use. Changes in trabecular architecture reflect increases in loading normally experienced by the bone. Similarly, mechanical properties of trabecular bone are anisotropic; directions normally subjected to higher loads exhibit, for example, thicker, stronger trabeculae that more closely resemble plate-like structures. Lightly loaded directions form more open networks. Changes in trabecular architecture and strength can also be related to diseases such as osteoporosis. Since variability in strength among trabecular bone samples is high, non-invasive methods are quite useful. Important research involves using 3D visualization to assist in the measuring of bone architecture, diagnosing, and studying bone disease. The science questions we are interested in relate to comparison of empirical studies on the bone structures to simulations and evaluation of hypothesis on structure and function. Because of the complex three-dimensional structure of the bone, volumetric rendering is useful. Because the simulation studies are done on massively parallel platforms, the combined rendering allows for amortization of data set movement that speeds up visualization and therefore investigation.

3 SYSTEM DESCRIPTION

The system is intended to support science investigation, and we make certain assumptions about the users of the system. We assume the users have networked computers, desire access from multiple platforms and platform types, and require access to shared supercomputing resources. The scenario includes internetworked computers within an intranet that share a joint rendering server. For our specific project, we have researchers in different locales (Santa Cruz, Palo Alto, Livermore, and Berkeley), different platforms (Hewlett-Packard, Silicon Graphics, and personal computers), and with different parallel servers available (MasPar, Meiko Scientific, Silicon Graphics Challenge, and Cray T3D).

A system to achieve our science goals with our modest support and personnel required using as many off-the-shelf components as possible. We therefore leveraged all user client software by assuming Netscape Navigator or Microsoft Explorer type front ends to the system. We also assumed that we had access to a web server, and that most networking transmission would use the WWW standards. Because the servers are at different sites we assumed the system would be a multiprocess system. Figure 3 shows an overview of the software architecture of the system. The four main processes are the *Web Server*, *Render Request*, *Render Server*, and *Child Render Work*. Only the last three are custom software, and many components of the custom programs are built from freeware.

The custom software developed is also indicated by the source files shown in Figure 3. The rectangular source files include: render.html, tcpclient.c, tcpserv.c myppmtogif.c, and render.c. The dashed lines show the source file dependencies to programs. The front end, because of the use of standard WWW browsers, is coded in hypertext markup language (HTML). This language specifies a hypertext document, and has embedded images. Figures 6 and 7 show the appearance of two of our front ends.

The tcpclient.c program is invoked as a common gateway interface (CGI) program, so it may run on a different platform than the web server that provides access to the *render.html* forms interface. The additional processes *Render Server* and *Child Render Work* are from the same program that continually runs, waiting for requests to create a volume rendering. The volume data is shown to be in *bone_volume.vol*, which is stored in memory for fast access to the rendering server. The results are always compressed before being sent back to the client (*myppmtogif.c*), and the server requires a volume renderer (*render.c*).

The software architecture is made clear through a brief explanation of its operation, as shown in the figure with the numbered circles. In (1) a user makes a request via the WWW to the *Web Server*. The *Web Server* returns the *render.html* front end forms interface. Once the user has selected the desired parameters, he chooses a button "render", which (2) invokes the CGI process. Typically CGI processes are found in the cgi-bin directory, and may be referred to as cgi-bin scripts. In (3) the cgi-bin script contacts the *Render Server*, which (4) causes the *Render Server* to fork off *Child Render Work*. In this way a single , server handles requests from many users which can be controlled by choosing how many connections the server will accept. In (5) *Child Render Work* accesses the source volume, *bone_volume.vol*, renders it into an image, compresses the image to the GIF format, (6) sends the result to through the socket to *Render Request*, and exits.

Render Request upon receipt of the image, in (7) writes it to a file, *result.gif*, and in (8) returns the appropriate uniform resource locator (URL) to the requesting client. The page returned to the client includes the URL for the image, and is typically in the form of a WWW page with the cgi-bin script as the page: http://Web Server2/cgi-bin/Render Request/?params. *Web Server2* is indicated if in fact a different server is used than for the initial request. The parameters indicate the location where the actual viewing parameters are passed to the server. The system is simple, efficient, and uses WWW technology including front end clients, servers, protocols, to provide access to rendered information. The advantage of our system is a small amount of code to be developed for general and flexible use.

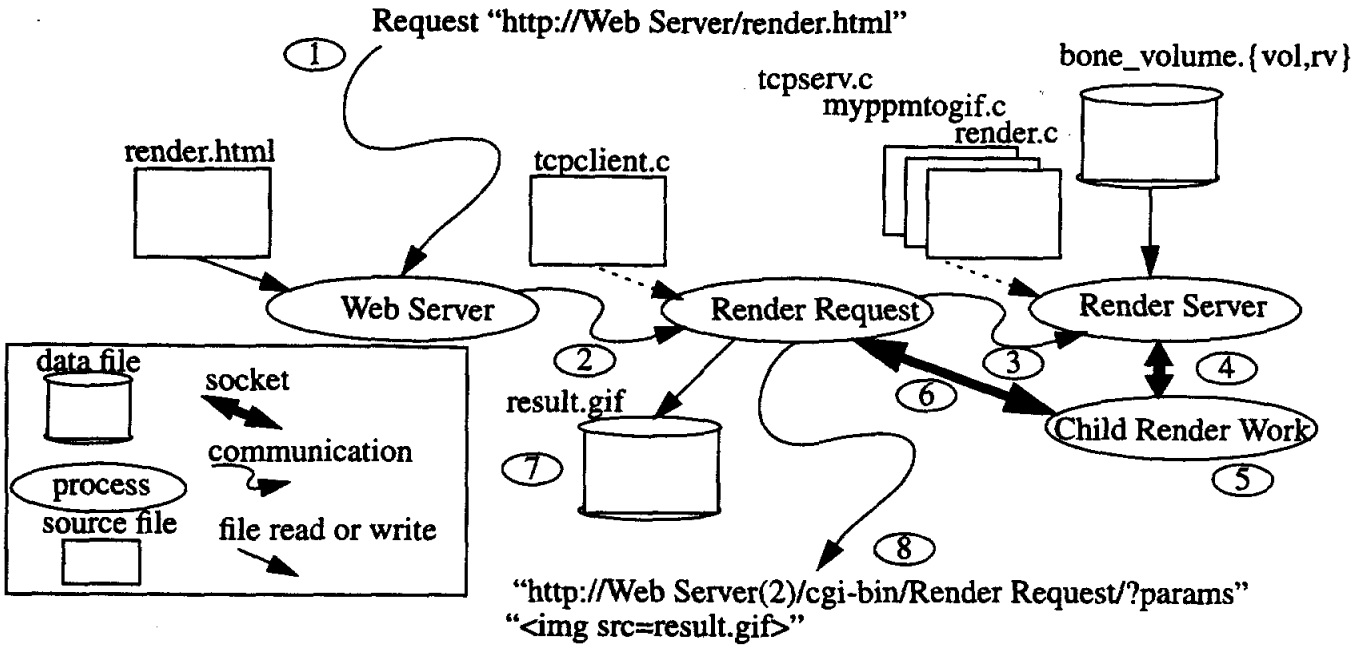


Figure 3: PermWeb architecture

4 RESULTS

We have developed several interfaces and experimented with different approaches to interfacing to renderers. We have four available interfaces for volume rendering, web page "http://www.cse.ucsc.edu/research/slvg". The user can try out different viewing parameters and/or data sets. The web clients link to the renderer (either a workstation or the MasPar MP-2) using the software architecture in the previous section. We briefly review some of the implementation decisions, lessons learned, and approaches.

For the first implementation the cgi-bin scripts are written in C and csh-scripts. A WWW client passes viewing parameters either through an HTML forms or, for the second implementation, a Java interface, which is link to a cgi-bin script to call the renderer. The Java classes[26] provide more interactivity in specifying the classification and viewing parameters, while maintaining the wide portability and accessibility of the web clients. The Java/WWW client provides a two dimensional interactive graphics interface, and the same approach looks promising for a fully interactive 3D volume renderer, allowing access to a variety of parallel platforms, and datasets for a large number of users. The third implementation interfaces to the MasPar, and the fourth implementation uses Java to animate prerendered images.

Because WWW servers often have security difficulties, it is important to limit access and flexibility as much as possible. CGI scripts are a possible security problem, especially when providing access to code in development. Therefore one security issue is how to provide development access to cgi-bin scripts. We have used different servers, placing cgi-bin scripts on a more restricted server to limit possible damage.

The system architecture can render from a network request without ever touching disk for data access. But, we have found in prototype and development, many of the common building blocks and volume renderers read and write from files. Our current implementation writes several intermediate files, a .ppm image, a .gif image, and the .gif image on the cgi-bin script server. In addition the memory caching of the source volumes has not been implemented, but is an easy extension to reduce latency of request to final image.

The two renderers we have experimented with are the MasPar rendering software that we developed, and are

extending,[25] and the publicly available shear warp volume renderer of Lacroute and Levoy.[19] Figure 6 and 7 show WWW/HTML/Java front ends to our implemented system. The MasPar renderer uses permutation warping as described earlier, and runs on the University of California Santa Cruz, MasPar MP-2, 4096 node machine. The machine has a DECstation front end, which runs the renderer. In this instance, it was more straight forward to have the server use a system call to run a separate volume rendering program on the MasPar, and retrieve the resulting image. The first algorithm is the permutation warping algorithm, and this is the base algorithm we used for our algorithm enhancements. The second major algorithm uses octree encoding of subvolumes to assist in compressing, and accelerating the visualization process.

The algorithm research is an investigation of the time, quality tradeoffs possible with the permutation warping approach. The results show that there are up to 400% run time advantages in using volume coherency to accelerate a SIMD implementation.[25] The MasPar code is in MPL, a parallel variant of the C language. Due to the lack of parallel I/O on our MasPar, the current bottleneck is not the rendering rate, but the time to read the volume data from disk.

The shear warp renderer (*webrender*) uses the VolPack library, and is a slightly modified example from the library that takes view parameters and input file, so that different input files can be used. We have run the shear warp on a Silicon Graphics Indigo2, again using a system call to have the *Render Server* call *webrender* to render the image. The input file types for the MasPar renderer are raw binary volumes in column, slice major ordering. The Shear Warp rendering program used the preprocessed volume in the VolPack defined .rv format. We have written converters to convert the bone data set into these two volume data formats.

One of the key components to achieve acceptable performance is the compression of the result image. By using the freeware GIFENCOD by David Rowley, which uses Lempel-Zif compression, a .ppm image can quickly be converted to a .gif image. The remote viewer than gets the image more quickly, even with the overhead of compressing the image. We have experimented with parallelizing the GIFENCOD code, and also coded the compression on the MasPar, but only have preliminary results. A centralized color hash table is used and further exploration of this is required.

In addition to the HTML front end, Java[26] has been investigated as a means to provide more interactivity to users. Figure 7 shows a prototyped Java front end. The user selects different views by moving the sliders which are indicated by the plotted circles of the center of view, and the object center. A Java class object is developed, compiled to Java byte code, and made available to the web server. It would be straight forward to develop a 3D interface for selecting viewpoints, and even additional parameter selections such as classification and shading.

The current performance of the MasPar access is on the order of many seconds. Depending on the loading of the MasPar, which is a queued system, there may be a delay to get in the execute queue. The file read time of the source volume dominates the total execution time, which could be fixed either by using a parallel I/O or memory caching of the volume using a continually running server. The current performance of the shear warp is typically a few seconds. Because of the relatively fast file access of the Indigo2, and the smaller source volumes used, the performance is reasonable for remote users. The precompressed, run-length encoded, source volumes used by shear warp help in reducing the file read latency. The anticipated performance given full optimization of the system software would be 5-10 frames per second. To get better performance than this may require a web browser plug in for a more direct connection to the rendering server, bypassing the multiple step socket paths used with the current system.

There are more and more advanced web technologies becoming available, and it may be effective to use digital video web browser plug in, and implement a custom render server that communicates with one of these browsers. In this way, the transmission of images would be standard, and likely Java class animated front ends, in coordination with a third party video plug in, would provide good performance with small development costs.

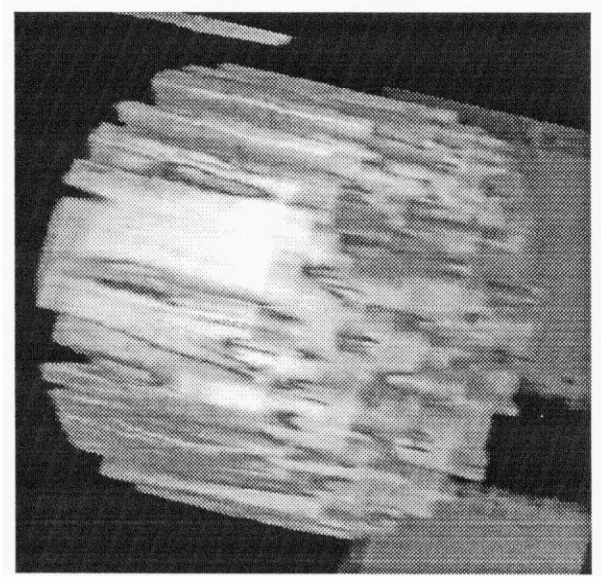


Figure 4: Rendering of trabecular bone, view 1.

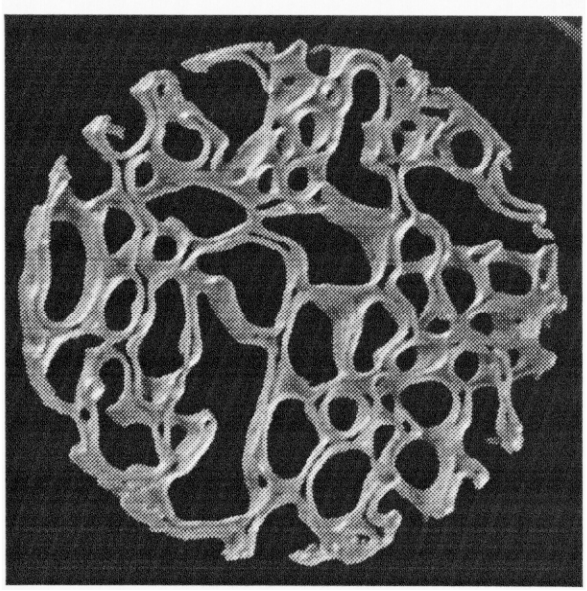


Figure 5: Trabecular bone, view 2.

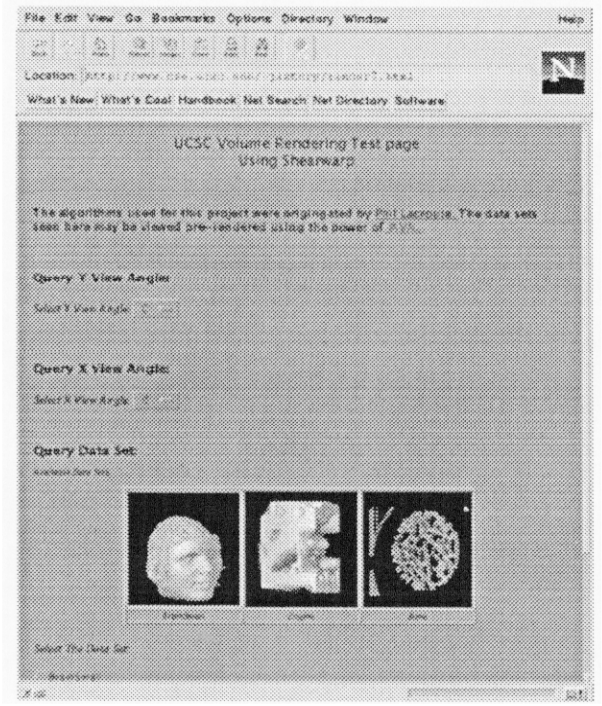


Figure 6: PermWeb HTML front-end.

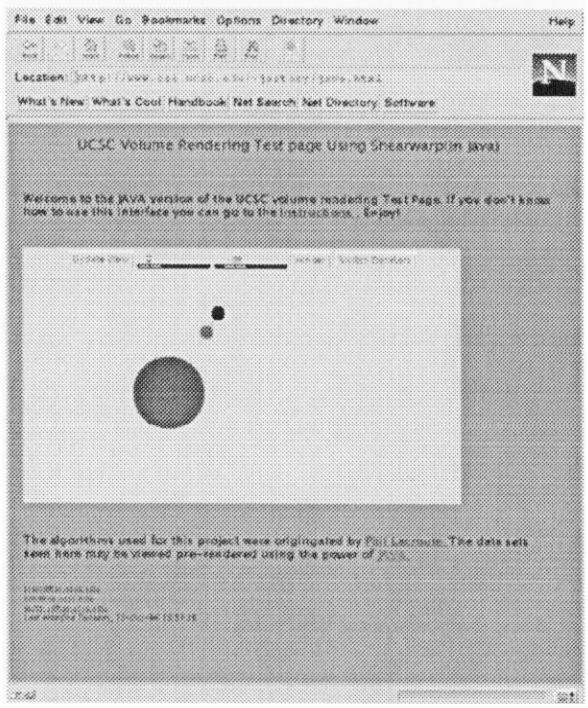


Figure 7: PermWeb Java front-end.

5 SUMMARY AND CONCLUSIONS

We have detailed a software architecture for remote volume rendering using many standardized WWW components, and a few custom components. The system was designed to address several key constraints for scientific inquiry including measurement simulation comparison, multiple study comparisons, and evaluation in research or diagnosis scenarios. Key to the performance of our system is image compression, and in-memory rendering and transfer. We have developed multiple front ends using this system architecture including Java interactive front ends for specifying viewing parameters. We have interfaced the system to multiple renderers, including our parallel permutation warping code on the MasPar, and the VolPack library code of Lacroute et al. We have investigated performance issues in reducing the latency of the rendering, and improving the system. Future research includes the development of coupled compression renderers, parallel compression algorithms, heterogeneous platform parallel rendering, and further data dependent optimization parallel rendering. The data dependent optimizations of permutation warping are the subject of another paper.[25] The lessons learned are that a small scale software project can easily leverage the WWW technologies available to make remote visualization systems. Anticipated developments in digital video web browser plug-ins, newer network technologies, and more flexible multimedia types will make such an architecture more and more effective at providing solutions for distributed visualization systems.

ACKNOWLEDGEMENTS

We would like to acknowledge our collaborators, especially the faculty and students of the Santa Cruz Laboratory for Visualization and Graphics. We would also like to thank Dr. Paul Skokowski for his interest in coupling the visualization and finite element research. This work is supported by the Lawrence Livermore National Laboratories, grant ISCR-LLNL B291836.

6 REFERENCES

[1] Dave Sims. Applications: Putting the visible human to work. *IEEE Computer Graphics and Applications*, 16(1):14–15, January 1996.

[2] C.S. Ang, D.C. Martin, and M.D. Doyle. Integrated control of distributed volume visualization through the world-wide-web. In *Proceedings Visualization '94*, pages 13–20, Washington, DC, October 1994. IEEE.

[3] David W. Robertson and William E. Johnston. Using the world wide web to provide a platform independent interface to high performance computing. In *COMPCON*, pages 3–7, San Francisco, CA, March 1995. IEEE.

[4] John R. Vacca. 3D worlds on the web. *Computer Graphics World*, 19(5):43–50, May 1996.

[5] Stuart K. Card. Special report, cg and viz in the gii: subreport, visualizing retrieved information: A survey. *IEEE Computer Graphics and Applications*, 16(2):63–67, March 1996.

[6] Hollerbach Karin and Hollister A. Computerized prosthetic modeling. In *Biomechanics*, September 1996.

[7] P.-L. Bossart, H. E. Martz, and K. Hollerbach. Finite element analysis of human joints: image processing and meshing issues. In *Proceedings of ICIP-96*, pages Vol. II–285–288, Lausanne, Switzerland, September 1996. IEEE.

[8] Karin Hollerbach. Modeling human joints and prosthetic implants. In *Science and Technology Review*, pages 19–21, September 1996.

[9] K. Hollerbach, D. Schauer, and Ashby A.E. Modeling the biomechanics of human joints and prosthetic implants. Technical Report UCRL-TB-118601-Rev.1, University of California, Lawrence Livermore National Laboratory, Livermore, CA, 1995.

[10] C. Nielsen, K. Hollerbach, S Perfect, and K Underhill. A computational method for comparing the behavior and possible failure of prosthetic implants. In *1995 IEEE Intl. Conf. of the Engineering in Medicine and Biology Society*. IEEE, 1995.

[11] Craig M. Wittenbrink and A. K. Somani. Permutation warping for data parallel volume rendering. *Journal of Parallel and Distributed Computing*, submitted, 1995. available as Technical Report, UCSC-CRL-96-33.

[12] Craig M. Wittenbrink and Michael Harrington. A scalable MIMD volume rendering algorithm. In *Proceedings IEEE 8th International Parallel Processing Symposium*, pages 916–920, Cancun, Mexico, April 1994.

[13] Craig M. Wittenbrink and A. K. Somani. 2D and 3D optimal parallel image warping. *Journal of Parallel and Distributed Computing*, 25(2):197–208, March 1995.

[14] Jim Blinn. Light reflection functions for simulations of clouds and dusty surfaces. In *Computer Graphics*, pages 21–29, July 1982.

[15] J. T. Kajiya and B. Von Herzen. Ray tracing volume densities. In *Proceedings of SIGGRAPH*, pages 165–174, July 1984.

[16] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(5):29–37, May 1988.

[17] J. Nieh and M. Levoy. Volume rendering on scalable shared-memory mimd architectures. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 17–24, October 1992.

[18] T. S. Yoo, U. Neumann, H. Fuchs, S. M. Pizer, T. Cullip, J. Rhoades, and R. Whitaker. Achieving direct volume visualization with interactive semantic region selection. In *Proceedings IEEE Visualization '91*, pages 58–65, San Diego, CA, October 1991.

[19] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH 94*, pages 451–458, Orlando, FL, July 1994.

[20] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 7–14, Atlanta, GA, Oct 1995. IEEE.

[21] Philippe Lacroute. Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization. In *Proceedings of the 1995 Parallel Rendering Symposium*, pages 15–22, Atlanta, GA, Oct 1995. IEEE.

[22] L. Westover. Footprint evaluation for volume rendering. In *Computer Graphics*, pages 367–376, August 1990.

[23] P. Schroder and G. Stoll. Data parallel volume rendering as line drawing. In *Proceedings of 1992 Workshop on Volume Visualization*, pages 25–32, Boston, MA, October 1992.

[24] Craig M. Wittenbrink and Arun K. Somani. Permutation warping for data parallel volume rendering. In *Proceedings of the Parallel Rendering Symposium*, pages 57–60, color plate p. 110, San Jose, CA, October 1993.

[25] Craig M. Wittenbrink, Kwansik Kim, and Alex T. Pang. Data dependent optimizations for permutation volume rendering. Technical Report UCSC-CRL-96-24, University of California, Santa Cruz, December 1996.

[26] David Flanagan. *Java in a Nutshell*. O'Reilly & Associates, Sebastopol, CA, February 1996.