Interactive Volume Visualization on a Heterogeneous Message-Passing Multicomputer

Andrei State*, Jonathan McAllister*, Ulrich Neumann‡,
Hong Chen*, Tim J. Cullip*, David T. Chen* and Henry Fuchs*

*University of North Carolina at Chapel Hill
‡University of Southern California

ABSTRACT

This paper describes *VOL2*, an interactive general-purpose volume renderer based on ray casting and implemented on Pixel-Planes 5, a distributed-memory, message-passing multicomputer. VOL2 is a pipelined renderer using image-space task parallelism and object-space data partitioning. We describe the parallelization and load balancing techniques used in order to achieve interactive response and near-real-time frame rates. We also present a number of applications for our system and derive some general conclusions about operation of image-order rendering algorithms on message-passing multicomputers.

1 INTRODUCTION AND PREVIOUS WORK

Volume rendering is a widely used visualization method. Due to the large number of graphics primitives (voxels) which must be visited during the image generation process, real-time (or even interactive) frame rates are difficult to achieve, even on highest-performance graphics engines. Previous work that addressed this computational expense problem includes [9], in which a number of parallelization and load balancing techniques for the special case of a shared-memory architecture were presented; the rendering algorithm used was ray casting with parallel projection.

We describe an equivalent system, *VOL2*, for a distributed-memory architecture. It uses ray casting with perspective projection, a general volume rendering method suitable for a variety of visualization tasks. Ray casting is an image-order algorithm in which volume data is traversed and sampled by rays emanating from the viewpoint; the rays intersect the image plane; they accumulate (integrate) information about the volume data during traversal. The algorithms and principles used as the basis for VOL2 are outlined in [2,4,5,8,10,13,19]. An early experimental precursor of VOL2 was mentioned in [12,19]. An early version of this paper was published as [11].

The remainder of this work is organized as follows: brief overview sections on the hardware platform used and the type of display presented to the user are followed by a detailed description of the internal pipelined-parallel system layout. We then describe the types of visualization modes and graphics primitives supported by VOL2. The largest section is devoted to methods used to obtain interactive and real-time performance levels; these include a technique derived from "frameless rendering" [1]. We conclude with an overview of applications for our system.

2 HARDWARE PLATFORM

VOL2 is implemented on Pixel-Planes 5, a high-performance graphics engine with general-purpose computing nodes (called Graphics Processors or GPs) based on the Intel i860 microprocessor, and special-purpose rendering nodes based on massively parallel SIMD processor-enhanced memories [3]. Each GP has 8 Megabytes of local memory. Each rendering node can execute pixel operations in parallel on a 128x128 pixel raster, which corresponds to 1/20 of the final 512x640 pixel image. All nodes are interconnected via the system's internal 5 Gigabit/sec token ring network. Also connected to the token ring are frame buffers and the Sun-4 host computer.

3 PRESENTED DISPLAY

VOL2 produces successively refined displays by rendering a coarse image while the view parameters are changing, and by gradually increasing the image quality during interaction pauses (Plate 1). Kinetic depth effect is provided by appending to such a successive refinement sequence a series of seven highest-resolution frames; these cyclically displayed cineloop frames present the visualized structures in animated oscillatory rotation (rocking). The user will observe gradually increasing image resolution, followed by increasingly smooth left-right rocking of the displayed high-resolution structures (as each successive cineloop frame is being computed, it is immediately included in the rocking sequence). Additional depth cues are provided by directional lighting with diffuse and specular reflections.

4 RENDERING PIPELINE

The rendering pipeline has six components (Fig. 1): the host, the master GP, ray casters, compositors, splat processors (for screen interpolation), and the frame buffer. The host provides UNIX services and allows users real-time control through an X-window interface and other input devices (joysticks, trackers). The master

*Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175.
(state|mcallist|chenh|cullip|chen|fuchs)@cs.unc.edu

‡Computer Science Department, University of Southern California, Henry Salvatori 338, Los Angeles, CA 90089-0781.
uneumann@usc.edu

1995 Symposium on Interactive 3D Graphics, Monterey CA USA
© 1995 ACM 0-89791-736-7/95/0004...$3.50

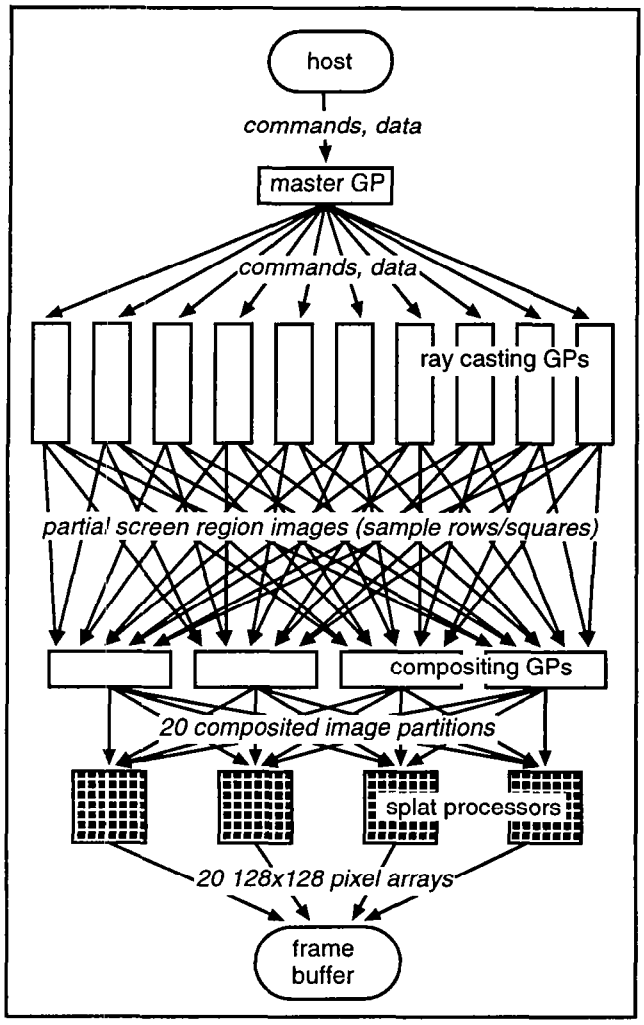

Fig. 1. VOL2 visualization pipeline.

GP is responsible for system synchronization and for load balancing the ray casters. Most of the i860 nodes are allocated as ray casters which compute image samples. Eight i860 nodes are used as compositors which combine the image samples into a final image. This image is sent to rendering nodes operating as splat processors which interpolate the image over the full display resolution and write the result to the frame buffer.

Local memory on each GP can hold only a limited number of voxels (about 6M in 8-bit voxel mode and 1.5M in 32-bit voxel mode). If the data set is too large to be replicated on all ray casting nodes, it is partitioned into slabs at system startup time; the ray casting GPs are partitioned into groups [8]. Each group of ray casting GPs is assigned to a slab of the data set (object-space partitioning, Fig. 2). During rendering, ray casters sample their assigned slabs on an image-space grid, compute partial screen region images (i. e., arrays of partially composited ray segments) and send these to the compositors. The latter combine the partial image samples into final image samples. This is accomplished by front-to-back compositing of the ray segments. Typically 8 nodes are allocated to the compositing task, each responsible for a 640x64 pixel horizontal band of the final 640x512 pixel image. The actual resolution of the computed image varies due to the use of successive refinement; the array of composited image samples sent to the splat processors to generate the fixed resolution (640x512) final image is thus of variable size.

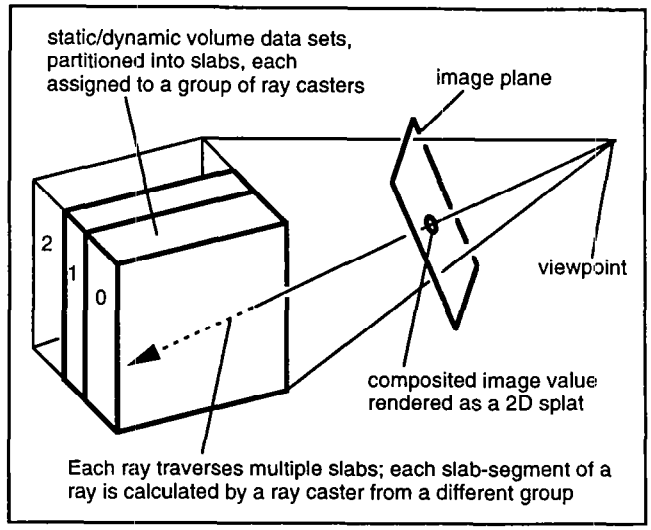

Fig. 2. Static object space data partitioning into parallel slabs.

The SIMD rendering nodes are used as splat processors due to their availability and efficiency at this task [10]. Composited image samples are convolved with a 2D filter kernel to resample the image at frame buffer resolution. Several user-selectable filter kernels are implemented, among them box, bilinear, biquadratic, piecewise quadratic and bicubic filters (Plate 2). The resampled values are sent to the frame buffer for display.

5 RENDERING OPTIONS

The ray caster code implements a number of rendering modes, such as isosurface rendering, direct rendering with and without shading, and maximum intensity projection (MIP). Plate 3 illustrates the visualizations obtained by these modes from the same data. Adding a new rendering mode to VOL2 amounts to writing a new ray caster core function; ray caster core functions are used in the innermost ray casting loop to sample the data set at a specific position along a ray and interpret the sample in a specific way (isosurface search, opacity accumulation, etc.). This modular design allows for easy prototyping and experimentation with new rendering modes without overburdening the programmer with the intricacies of Pixel-Planes 5 multiprogramming.

VOL2 supports wireframe line segments and flat-shaded triangles as graphics primitives. The (antialiased) lines are Z-buffered against isosurfaces and against each other. They are added by the splat processors (Fig. 3) after the image is resampled to frame buffer resolution (lines are only visible within fully transparent areas of the data set). Triangles can be used to add reference geometry to the scene and may penetrate into the volume data set. They are rendered by the ray casting GPs since they must be composited properly with the volume data. Since there are typically few triangles in our applications, their rendering cost is minimized by testing their individual bounding boxes against each screen region to ascertain if rays cast on a particular ray casting GP (i. e., through a specific screen region) will hit any triangles; ray setup involves computing the intersection distance to the polygons to eliminate intersection tests at every ray step.

A cut-plane for the volume data set is also provided. It is textured with the volumetric data (visible in Plate 5). The cut plane can be moved by the user to examine any arbitrarily positioned or oriented cross-section of the volumetric data set.

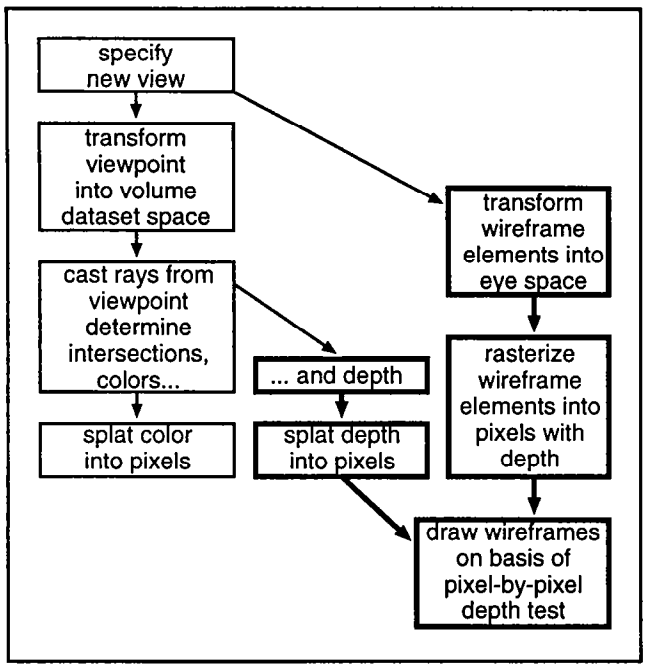

Fig. 3. Algorithm for combining wireframe line segment primitives with volume rendered images.

6 OBTAINING INTERACTIVE PERFORMANCE

The generality of ray-casting (for example, in isosurface rendering the surface thresholds can be changed on-the-fly, since no intermediate geometric primitives have to be generated). has its price. Ray casting is computationally expensive, even for relatively small data sets (1M voxels). We therefore attempted to identify and remove or alleviate VOL2's performance bottlenecks.

6.1 BY-PASS CODE

In order to obtain timing measurements, *by-pass* code was implemented in the master GP, ray casting and compositing nodes. By-pass code is derived from the code normally executing on the computing nodes by removing all compute-intensive operations and retaining only the message-passing instructions, thus preserving a computing node's ability to operate in the system (by essentially "fooling" the nodes it communicates with).

By selectively activating by-pass code for certain nodes, one can determine how fast the rest of the system can be operated. For example, by activating by-pass code for all nodes, we can determine the maximum obtainable system performance for our image generation pipeline layout (Fig. 4); by activating by-pass code for all nodes except the master GP, we can determine at what frame rates the master GP becomes overburdened during system operation (Fig. 5); by activating by-pass code for the ray casters and the master GP, we obtain the maximum speed at which the compositing/splatting/display back-end can operate—compositing performance is fairly independent of image content; it depends mostly on image resolution and the number of object partitioning slabs (Fig. 6).

6.2 LOAD BALANCE AND ADAPTIVE SAMPLING

Unlike the other system components, the ray casting nodes do not exhibit a maximum speed behavior. Their performance depends largely on data set size and image content. While master GP and compositing back-end have to be able to keep up with the ray

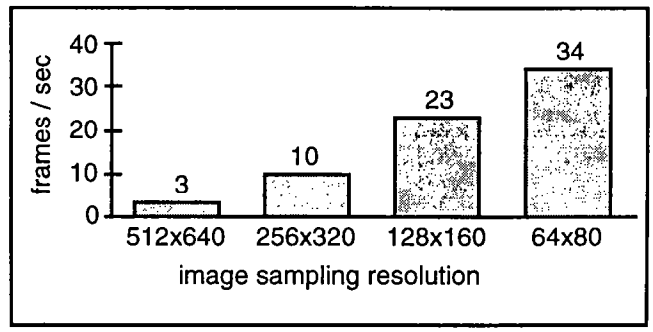

Fig. 4. Rendering pipeline throughput, measured with by-passed image generation code in all pipeline stages; this shows the maximum speed supported by the message-passing framework at different image sampling resolutions. These numbers are independent of the number of computing nodes present in the system.

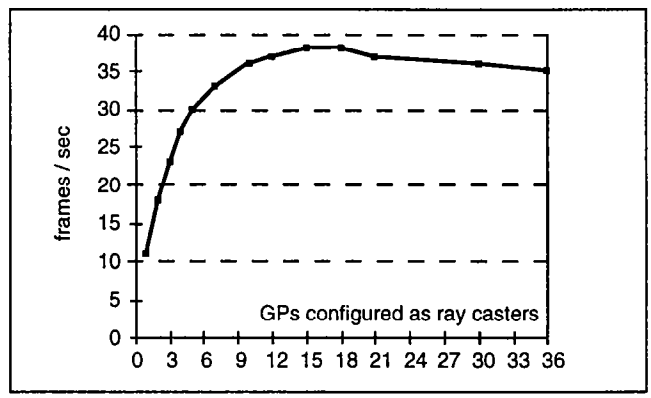

Fig. 5. Master GP maximum performance, measured with by-passed code on all nodes except the master GP. If the system contains few ray casters, the frame rate is low due to screen region processing (however minimal due to by-passing) on a single ray caster. For larger numbers of ray casters, we measure master GP maximum speed.

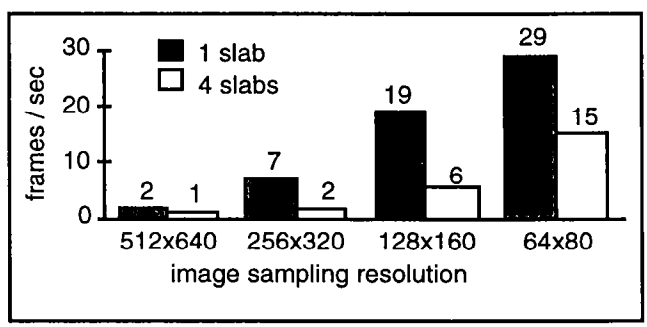

Fig. 6. Performance of the compositing-splatting-display back-end, measured with by-passed image generation code in the ray casting nodes. These numbers are independent of the number of computing nodes present in the system.

casters for the types of data sets and displays VOL2 is normally used for, the ray casters themselves have to be load balanced with respect to each other. To that end, ray casters are dynamically assigned screen regions for image generation processing. Two assignment methods are implemented and are user-selectable. In the *sample rows* approach the master GP assigns sequential rows of samples (Fig. 7, left) to the ray casting nodes on a first-come-first-serve (FCFS) basis. The rows are distributed in order, starting at the top of the image. This provides good load balance, but precludes adaptive sampling since a 2D context is required for it on each ray caster.

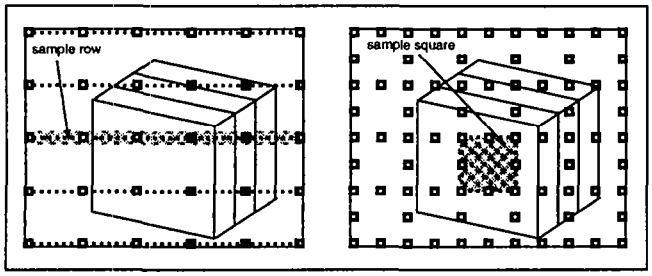

Fig. 7. Image space partitioning for load balancing by sample rows (left) and sample squares (right).

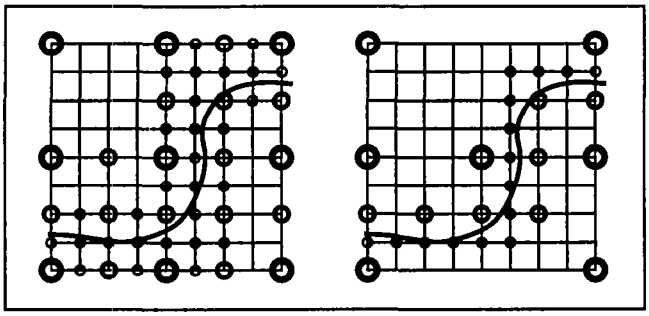

Fig. 8. Conventional adaptive subdivison (left) causes 51 samples to be taken while partial subdivision (right) requires only 29 samples. Samples taken at successive levels of subdivision are represented by progressively finer circles. The curve boundary triggers the subdivision criterion.

In the second load balancing approach (*sample squares*) the master GP distributes a total of eighty 65x65-pixel square screen regions (Fig 7, right) on a FCFS basis to the ray casting nodes. The square region size (1/80th of the final image) includes two edges of replicated rays to support adaptive sampling without seams. Squares are distributed in order of descending cost, where cost is the time taken to render the region in the previous frame (0th order cost prediction). Typically, assignment of squares on the basis of descending cost provides approximately 10-20% increase in frame rate over distribution in screen order.

The sample squares approach is combined with adaptive sampling, implemented as a modified form of recursive square subdivision. It requires fewer rays and provides similar results to that used in [6]. The conventional approach (Fig. 8, left) fully subdivides a square area by computing five new samples when any pair of four corner values exhibit variance above a user-defined threshold. Our partial subdivision approach (Fig. 8, right) computes new samples only between varying sample pairs with the center sample taken if any samples within a square vary. The example shows that the new approach requires fewer samples than the full subdivision method. A triangular subdivision method [15] has similar economy, but is less well suited to square regions.

For the isosurface ray caster, an additional optimization technique is used in combination with adaptive sampling: the ordered sequence of isosurfaces encountered along each ray is encoded in the ray sample. The encoded values are also compared during adaptive sampling; differences between neighboring rays trigger adaptive sampling along contours and isosurface intersection curves even if the threshold criterion is not met, thus enforcing accurate edge and intersection curve display.

Plate 4 shows a bar graph of the ray casting GP workloads normalized to the highest load (these and other types of test displays have proven very useful for observing the behavior of our system). Both the sample rows and sample squares approaches

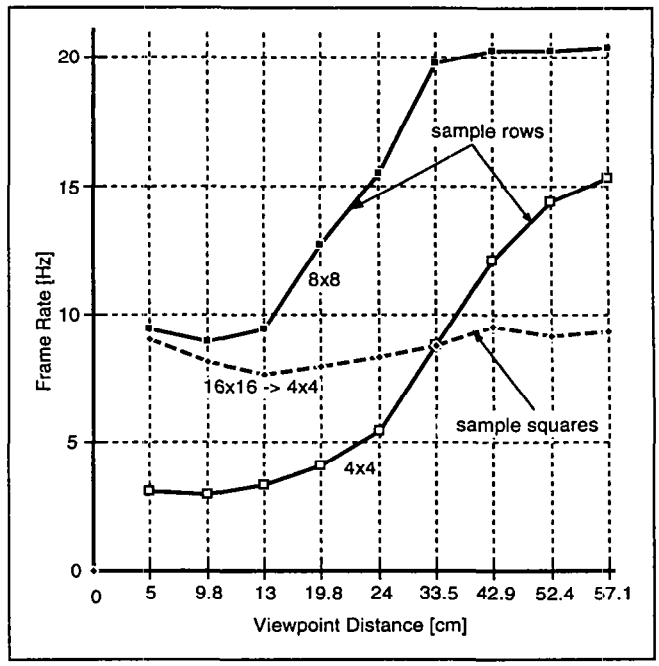

Fig. 9. Frame rate comparison between sample rows and sample squares for varying volume data set size in image space. (Full screen images are produced by 5 cm viewpoint distance, 60 cm distance produces approximately 1/16 screen coverage.) Line partitions cast rays every 8x8 or 4x4 pixels. Square regions are adaptively sampled initially at one ray per 16x16 pixels and refined up to one ray per 4x4 pixels. These measurements were taken on a system containing a total of 21 GPs, of which only 4 were allocated as compositors. Figures for larger systems with 8-compositor allocation are slightly higher.

produce good load balance with the former giving better performances for images with low screen coverage and the latter approach giving better performance for full-screen images, as well as more consistent frame rates over varied image sizes (Fig. 9).

6.3 PARTIAL UPDATING

In addition to the multiple successive refinement levels, the user can select a partial updating mode to increase the frame rate. Partial updating is loosely based on the frameless rendering technique described in [1]. This causes a new frame to be displayed as soon as a user-selected fraction of the image samples have been updated. Update levels of 25%, 50%, and 100% are currently implemented. For example, if the partial updating fraction is 25%, each sample is updated once every 4 frames. When user interaction pauses, an image at the lowest successive refinement level will have filled in after four frames.

Rather than updating a randomly distributed set of samples, we update the samples on a regular grid, which has the benefit that the bookkeeping required to ensure every sample eventually gets replaced if samples are chosen randomly all but disappears; a simple modulus of the sample coordinates with the frame number tells whether to cast a ray for a given sample on a given frame.

Partial updating implies incremental image modification, requiring the array of screen samples to be preserved from one frame to the next. In our implementation, this array is stored on the compositing nodes. Note that due to image partitioning for dynamic load balancing of the ray casters, it would be difficult to preserve the (fragmented) images on the ray casters; the existence of a compositing step in our pipeline proved advantageous for the implementation of partial updating.

6.4 OTHER OPTIMIZATIONS

A number of standard techniques are used to speed up ray casting. The voxels are stored with 13-bit pre-computed normals. A shading table is computed at the start of every frame that encodes the Lambertian coefficient for the given light direction(s) as a function of the surface normal. Voxel shading is efficiently performed by lookup into this table. Pre-computed threshold bits at each voxel accelerate ray processing by flagging whether an 8-voxel cell has "interesting" material within it. The highest value in each cell is also pre-computed and used to speed up ray casting. Rays are terminated when an opacity threshold is reached.

7 APPLICATIONS

VOL2 has been used as a rendering engine (both as a separate stand-alone server and embedded in a more complex system) for a number of research projects:

7.1 INTERACTIVE RADIATION THERAPY PLANNING

VOL2 is used as a visualization tool within VISTAnet, a collaborative project whose principal application is interactive radiation therapy planning (IRTP); the goal is to deliver lethal radiation to cancerous tissue, while keeping the doses received by healthy tissue at non-lethal levels. The treatment strategy is to intersect multiple treatment beams onto a predetermined 3D target region of a patient's anatomy, a complex task requiring comprehension of shape and sensitivity of the anatomy. VISTAnet is an experimental tool enabling 3D IRTP through rapid radiation dose computation (on a Cray Y-MP™ supercomputer) combined with interactive radiation dose visualization. Cray and Pixel-Planes 5 are linked by a near-gigabit communication network.

During an interactive session, a physician user specifies anatomy data sets and defines or modifies treatment beam parameters. These are transmitted to the Cray, which computes the dose distribution produced within the anatomy by the current treatment beam configuration and sends the dose data over the high-speed network to Pixel-Planes 5, where a combined image of anatomy, treatment beams, and resulting dose is generated; the physician examines the rendering and continues to adjust the parameters. The current processing rate is several such adjustments per second for anatomy data sets containing about 1M voxels. The display (Plate 5) must hence be able to quickly convey the treatment plan's characteristics to the user.

A special ray caster core function was added to VOL2 for operation under VISTAnet; it performs isosurface rendering of anatomy and dose data sets. For the anatomy, user-defined thresholds in the CT data and pre-defined organ or tumor segmentation data are both used for on-the-fly isosurface search during ray traversal; simultaneously, the radiation dose data set is traversed in search of up to three radiation dose isosurfaces, also with user-defined thresholds. Proper compositing of the dose, anatomy and organ or tumor surfaces must be ensured, especially when multiple surfaces lie between ray samples (Fig. 10); each surface's distance from the previous sample point along the ray is computed and sorted to establish the correct order for compositing. Wireframe outlines for the radiation treatment beams are rendered using VOL2's line segment primitives.

The (dynamically changing) radiation dose data set is received asynchronously from the Cray (via the Network Interface Unit or NIU, also attached to the Pixel-Planes 5 token ring and providing access to the external VISTAnet Gigabit network). An incoming

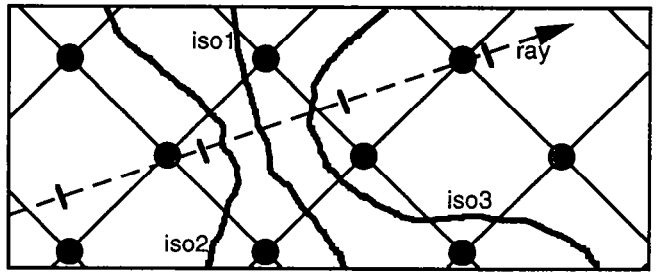

Fig. 10. Isosurfaces between samples along a ray are encountered in algorithmic order but must be sorted for compositing. Surfaces detected in <1,2,3>-order must be composited in <2,1,3>-order.

radiation dose preempts ongoing rendering for the current frame and switches context to a different task which distributes the new radiation dose to all ray casters as it is received. The distribution scheme follows the data set slab partitioning scheme described.

VISTAnet is described in more detail in [12,15].

7.2 INTERACTIVE 3D ULTRASOUND VISUALIZATION

The dynamic data set updating capabilities developed for VISTAnet are also used in an experimental augmented-reality ultrasound visualization system (Plate 6). For this system we have allocated a number of computing nodes to a volume reconstruction task: video images from an ultrasound machine are resampled into a volume data set, which is then transmitted to the ray casters for near-real-time image generation [17]. This system also required the incorporation of virtual-reality-type head and hand tracking support.

7.3 STEREOSCOPIC DISPLAY

Support for stereoscopic visualization using field-sequential stereo display on a large rear-projection screen was added to VOL2 for virtual reality experiments, as was the capability to generate such displays for head tracked viewing; this includes off-center perspective projection and the ability to position the viewpoint inside the volumetric data set.

7.4 OFF-LINE IMAGE GENERATION

Finally, VOL2 has also been used as an off-line rendering tool for simulated augmented-reality ultrasound visualization [17] and as an image precomputation tool for an experimental head-motion parallax visualization system [16].

8 CONCLUSIONS

The methods used to obtain the current performance (pipelined system layout, load balance between pipeline stages as well as between parallel nodes of individual pipeline stages) were successful—VOL2 has even been used as a skeleton for other Pixel-Planes-5-based parallel image-order renderers: polygon-based interactive ray tracing and interactive image-based morphing; both take advantage of the sophisticated, finely tuneable control over the performance/image quality tradeoff provided by the VOL2 framework.

We consider the by-pass code method one of the most useful lessons learned while building this system. This technique is generally applicable to the design of parallel/pipelined image-order renderers and has proven extremely useful as a tool to detect and eliminate performance bottlenecks in a complex multicomputer-based real-time rendering system.

9 FUTURE WORK

It has been extremely difficult to achieve VOL2's current frame rates and interactive response characteristics. The performance/resolution tradeoff is particularly unsatisfactory since it weakens kinetic depth cues. The system does indeed provide both interactive frame rates and strong kinetic depth, but not simultaneously (and hence not interactively), due to insufficient computational power. We expect significant performance improvements from an implementation of a general-purpose volume rendering algorithm on next-generation graphics multicomputers [7].

10 ACKNOWLEDGMENTS

We acknowledge the significant algorithm and software development efforts of John Rhoades, Qin Fang, Matt Lavoie, Jim Symon and Suresh Balu. This work was supported by NSF and ARPA under Cooperative Agreement NCR-8919038 with CNRI ("VISTAnet: A Very High Bandwidth Prototype Network for Interactive 3D Imaging"), by BellSouth, and by GTE. Additional funding was provided by ARPA ISTO contract DAEA 18-90-C-0044 ("Advanced Technology for Portable Personal Visualization").

REFERENCES

1. Bishop, Gary, Henry Fuchs, Leonard McMillan and Ellen J. Scher Zagier. "Frameless Rendering: Double Buffering Considered Harmful," Proceedings of SIGGRAPH '94 (Orlando, FL, July 24–29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, pp. 175–176.

2. Drebin, Robert A., Loren Carpenter, and Pat Hanrahan. "Volume Rendering," Proceedings of SIGGRAPH '88 (Atlanta, GA, August 1–5, 1988). In *Computer Graphics*, 22, 4, (August 1988), ACM SIGGRAPH, New York, pp. 65–74.

3. Fuchs, Henry, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs and Laura Israel. "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," Proceedings of SIGGRAPH '89 (Boston, MA, July 31–August 4, 1989). In *Computer Graphics*, 23, 3 (August 1989), ACM SIGGRAPH, New York, 1989, pp. 79–88.

4. Levoy, Marc. "Volume Rendering by Adaptive Refinement," *The Visual Computer*, 1990, 6, pp 2–7.

5. Levoy, Marc. "Design for a Real-Time High-Quality Volume Rendering Workstation," *Proceedings of the Chapel Hill Volume Visualization Workshop* (Chapel Hill, NC, May 1989), pp. 85–92.

6. Levoy, Marc. "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, May 1988, pp. 29–37.

7. Molnar, Steven, John Eyles and John Poulton. "PixelFlow: High-Speed Rendering Using Image Composition," Proceedings of SIGGRAPH '92 (Chicago, IL, July 26–31, 1992). In *Computer Graphics*, 26, 2 (July 1992), ACM SIGGRAPH, New York, 1992, pp. 231–240.

8. Montani, C., R. Perego and R. Scopigno. "Parallel Volume Visualization on a Hypercube Architecture," Proceedings of the 1992 Workshop on Volume Visualization (Boston, MA, October 19–20, 1992), special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992, pp. 9–16.

9. Nieh, Jason and Marc Levoy. "Volume Rendering on Scalable Shared-Memory MIMD Architectures," Proceedings of the 1992 Workshop on Volume Visualization (Boston, MA, October 19–20, 1992), special issue of *Computer Graphics*, ACM SIGGRAPH, New York, 1992, pp. 17–24.

10. Neumann, Ulrich. "Interactive Volume Rendering on a Multicomputer," Proceedings of the 1992 Symposium on Interactive 3D Graphics (Cambridge, MA, March 29–April 1, 1992), special issue of *Computer Graphics*, ACM SIGGRAPH, 1992, pp. 87–93.

11. Neumann, Ulrich, Andrei State, Hong Chen, Henry Fuchs, Tim J. Cullip, Qin Fang, Matt Lavoie and John Rhoades. "Interactive Multimodal Volume Visualization for a Distributed Radiation-Treatment Planning Simulator," Technical Report TR94-040, University of North Carolina at Chapel Hill, Computer Science Department, June 1994.

12. Rosenman, Julian, Edward L. Chaney, Tim J. Cullip, James R. Symon, Vernon L. Chi, Henry Fuchs and Daniel S. Stevenson. "VISTAnet: Interactive Real-Time Calculation and Display of 3-Dimensional Radiation Dose: An Application of Gigabit Networking," *Int. J. Radiation Oncology Biol. Phys.*, 25, Pergamon Press Ltd., 1992, pp. 123–129.

13. Sabella, Paolo. "A Rendering Algorithm for Visualizing 3D Scalar Fields," Proceedings of SIGGRAPH '88 (Atlanta, GA, August 1–5, 1988). In *Computer Graphics*, 22, 4, (August 1988), ACM SIGGRAPH, New York, pp. 51–58.

14. Shu, Renben and Alan Liu. "A Fast Ray Casting Algorithm Using Adaptive Isotriangular Subdivision," Proceedings of Visualization '91 (San Diego, CA, October 22–25, 1991), Gregory M. Nielson and Larry Rosenblum, Editors, IEEE Computer Society Press, Los Alamitos, CA, October 1991, pp. 232–238 and 426.

15. State, Andrei, Julian Rosenman, Henry Fuchs, Tim J. Cullip and Jim Symon. "VISTAnet: Radiation therapy treatment planning through rapid dose calculation and interactive 3D volume visualization,"*Visualization in Biomedical Computing 1994* (Rochester, MN, October 4–7, 1994), Richard A. Robb, Editor, Proc. SPIE 2359, 1994, pp. 484–492.

16. State, Andrei, Suresh Balu and Henry Fuchs. "Bunker View: Limited-range head-motion-parallax visualization for complex data sets," *Visualization in Biomedical Computing 1994* (Rochester, MN, October 4–7, 1994), Richard A. Robb, Editor, Proc. SPIE 2359, 1994, pp. 301–306.

17. State, Andrei, David T. Chen, Chris Tector, Andrew Brandt, Hong Chen, Ryutarou Ohbuchi, Mike Bajura and Henry Fuchs. "Case Study: Observing a Volume Rendered Fetus within a Pregnant Patient," *Proceedings of Visualization '94* (Washington, DC, October 17–21, 1994), R. Daniel Bergeron and Arie Kaufman, Editors, IEEE Computer Society Press, Los Alamitos, CA, pp. 364–368 and CP-41.

18. Westover, Lee. "Interactive Volume Rendering," *Proceedings of the Chapel Hill Volume Visualization Workshop* (Chapel Hill, NC, May 1989), pp. 9–16.

19. Yoo, Terry S., Ulrich Neumann, Henry Fuchs, Stephen M. Pizer, Tim Cullip, John Rhoades and Ross Whitaker. "Direct Visualization of Volume Data," *IEEE Computer Graphics and Applications*, 12, 4, Los Alamitos, CA, July 1992, pp. 63–71.

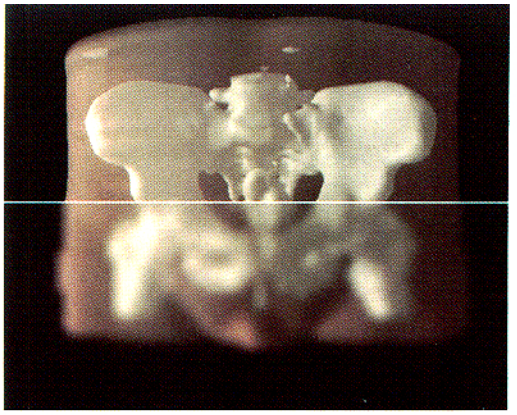

Plate 1. Isosurface rendering of human pelvis. Bottom: adaptive sampling from every 16th down to every 4th pixel, 10 frames/sec. Top: adaptive sampling from every 4th down to every pixel, 1 frame/sec.

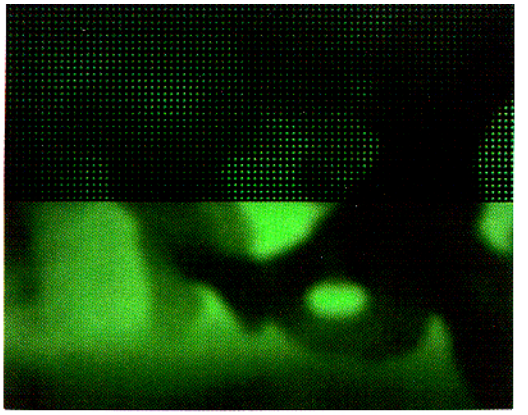

Plate 2. Top half: low resolution ray casting samples computed every 8 pixels. Bottom half: the result of splatting the samples onto a 512x640 image using a Gaussian kernel.

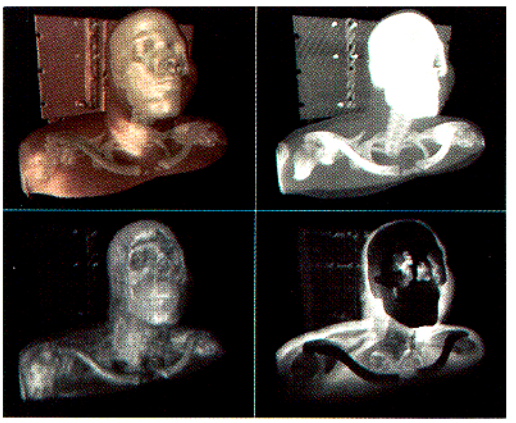

Plate 3. Rendering modes. Top left: twin isosurfaces. Top right: maximum intensity projection. Bottom left: direct rendering with shading (Levoy rendering). Bottom right: direct rendering without shading.

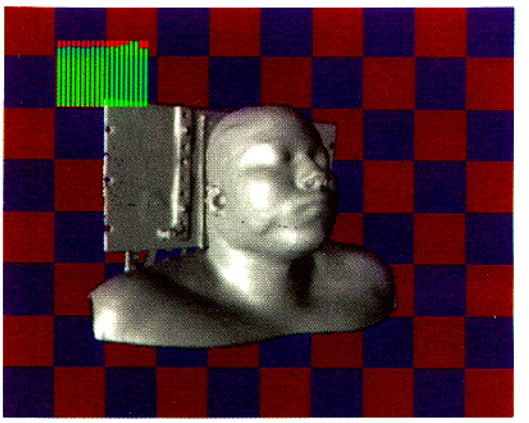

Plate 4. Load balancing in sample squares image partitioning method. The bar graph at the upper left shows load balancing among the ray casting GPs. The red/blue background shows the sample squares.

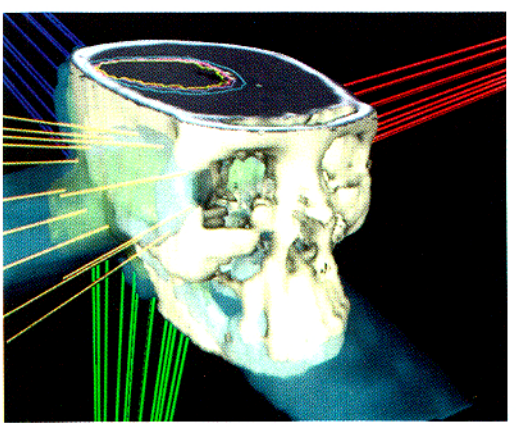

Plate 5. Visualization for interactive radiation therapy planning. The wireframe treatment beams intersect in a tumor located behind the right ear. The textured cut plane shows radiation dose isocurves. The blue isosurface shows a user-specified radiation dose threshold.

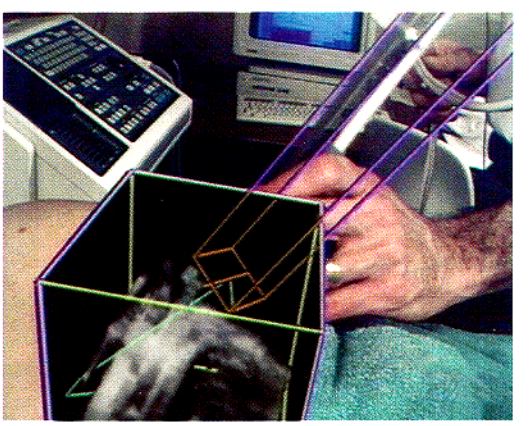

Plate 6. Augmented-reality ultrasound system (view within head-mounted-display). Several GPs perform volume reconstruction (on-line resampling of 2D ultrasound slices into a 3D volume). The ultrasound probe, shown in wireframe, appears to emit volume material as it is being moved through the box.