Real-Time Incremental Visualization of Dynamic Ultrasound Volumes Using Parallel BSP Trees

William F. Garrett, Henry Fuchs, Mary C. Whitton, and Andrei State

PUBLISHED IN THE PROCEEDINGS OF VISUALIZATION '96

Department of Computer Science[1]
University of North Carolina at Chapel Hill

ABSTRACT

We present a method for producing real-time volume visualizations of continuously captured, arbitrarily-oriented 2D arrays (slices) of data. Our system constructs a 3D representation on-the-fly from incoming 2D ultrasound slices by modeling and rendering the slices as planar polygons with translucent surface textures. We use binary space partition (BSP) tree data structures to provide non-intersecting, visibility-ordered primitives for accurate opacity accumulation images. New in our system is a method of using parallel, time-shifted BSP trees to efficiently manage the continuously captured ultrasound data and to decrease the variability in image generation time between output frames. This technique is employed in a functioning real-time augmented reality system that a physician has used to examine human patients prior to breast biopsy procedures. We expect the technique can be used for real-time visualization of any 2D data being collected from a tracked sensor moving along an arbitrary path.

CR Categories and Subject Descriptors: E.1 [Data Structures]: *Trees,* I.3.1: [Hardware Architecture]: *Three-dimensional displays,* I.3.7 [Three-Dimensional Graphics and Realism]: *Color, shading, shadowing, and texture* and *Visible line/surface algorithms,* J.3 [Life and Medical Sciences]: *Medical information systems.*

Additional Keywords and Phrases: Augmented reality, ultrasound echography, 3D medical imaging, BSP tree.

1 INTRODUCTION

Ultrasound echography is a popular imaging modality for many medical applications including fetal and cardiological examinations. We are developing an augmented reality system for use in ultrasound-guided needle biopsy of the breast (Plate 1). As the anatomy being examined is inherently three dimensional, the physician would often like to see the data rendered as a volume rather than as the 2D cross-sectional images (slices) generated by most ultrasound machines. Although 3D ultrasound acquisition systems are available commercially, they are not widely used. The 3D systems are problematic for applications such as breast biopsy because the apparatus may obstruct the physician's access to the patient's body. 2D acquisition systems are likely to remain common many years, so it is fruitful to investigate real-time rendering of data from this class of sensors.

Our goal is to provide the physician with a volume representation of ultrasound data displayed in real time as she scans a patient with a hand-held 2D ultrasound probe. The rendering portion of our system must address three problems: the arbitrary position and orientation of the ultrasound data, the continuous capture of new data that must be combined with older data for display, and the need for real-time operation. Plate 2, showing a hand scanned in a tank of water, illustrates how 3D shape can be recognizably represented in opacity accumulation images of multiple slices. Plate 3 illustrates the challenge of rendering a combination of older and newer data when scanning a moving target, in this case a closed hand opening during the scan.

A hand-held ultrasound probe can be moved freely with six degrees of freedom. Although each slice is in a known and fixed position relative to the ultrasound probe at the time it is captured, slices can have arbitrary position and orientation with respect to each other. Figure 1 shows that slices may intersect and samples may be non-uniformly spaced due to variations in the speed of probe movement. In contrast, other popular 3D imaging modalities, e.g. CT and MRI, produce parallel slices of data with known geometric relationships between slices. Compositing 2D slices to make a 3D image from data collected with a hand-held ultrasound probe requires properly rendering an irregular array of data after deriving the slice positions and orientations from the (tracked) probe location.

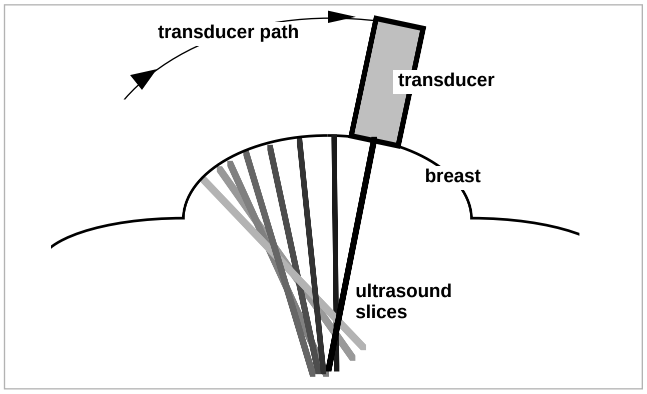

Figure 1: Intersecting and unevenly spaced ultrasound slices. Lighter shading indicates older slices.

The second problem is the real-time, continuous nature of the source data. Accumulating and displaying all the data collected at 30 frames/sec in a scanning session would lead to uninterpretable images. The system must manage an "active set" of data by continually adding new slices and eliminating older ones. The user should be able to control the number of slices in the active set. The system should also (optionally) visually distinguish older and newer data so that newer data is emphasized and shape and position changes are more noticeable.

Third, for real-time augmented reality applications the system must run at a minimum of 10 frames per second in stereo. This constraint made us look beyond traditional volume

[1]CB# 3175 Sitterson Hall, Chapel Hill, NC 27599-3175, tel. 919-962-1700
Email: {garrett, fuchs, whitton, state}@cs.unc.edu

visualization methods which require computationally expensive data resampling.

In [State 1996] we described an augmented reality system that merges ultrasound rendering with live images of the physical environment. The work reported here enabled the 3D real-time volume visualization in that system, but was described only briefly in that paper. The method is extended and analyzed here.

2 BACKGROUND

Our work builds on previous research in volume rendering, ultrasound visualization, and BSP trees. Of particular relevance is research on using textures and hardware texture accelerators to render volumes and research using polygonal primitives to represent ultrasound slices.

2.1 Volume Rendering

Volume rendering methods, like rendering methods in general, can be divided into two categories: *backward-mapping* methods (e.g., ray casting [Tuy 1984], where the image plane is mapped onto the data, and *forward-mapping* methods (e.g., splatting [Westover 1990]) where data is mapped to screen space. Most methods require (re)sampling data to a regular 3D grid or compiling adjacency information for irregular grids. Moreover, to render with translucency (e.g., “Levoy rendering” [Levoy 1988] or simple opacity accumulation), the volume must be sampled in back-to-front or front-to-back order for proper compositing.

Most computers are not powerful enough to resample and render large, screen-filling datasets (256^3 voxels) at interactive rates (10 Hz or better) but many modern computers do offer hardware acceleration for rendering textured polygons. Cullip and Neumann [Cullip 1994] proposed a simple method for using the texture-rendering capabilities of a Silicon Graphics RealityEngine for volume rendering. Stein, Becker, and Max [Stein 1994] demonstrate how the volume rendering method of cell projection [Shirley 1990] can be implemented with hardware-assisted texture mapping. Cabral, Cam, and Foran [Cabral 1994] provide some of the mathematical foundations for generating volume-rendered images with texture-mapping hardware.

2.2 Ultrasound Visualization Systems

Thune and Olstad [Thune 1991] presented a system for capturing time-varying 3D ultrasound data using a restricted motion (rotation only) ultrasound probe and rendering images off-line. Sakas and Walter [Sakas 1995] built a system for visualizing 3D ultrasound data, characterized by the use of a motor-driven ultrasound probe, multi-step filtering, and very high visual quality. Their technique includes space-filling interpolation between ultrasound slices during volume reconstruction.

State, et. al. [State 1994] generated a sequence of images showing a moving observer’s view of a reconstructed volume (a fetus) within a pregnant patient. The researchers collected time-stamped data from a tracker on a freely movable hand-held ultrasound probe, a tracker on the user’s head, and images from a head-mounted video camera. The animation, although showing lower volume reconstruction quality than [Sakas 1995], simulated what a real-time augmented-reality visualization running on a more-powerful future machine might display to a user.

Bajura, Fuchs, and Ohbuchi [Bajura 1992] introduced the concept of rendering ultrasound slices as polygon-like objects in an early real-time augmented reality system. The system displayed intersecting, opaque primitives via z-buffering. The researchers exploited the hardware accelerated sphere primitive of Pixel Planes 5 and approximated a textured polygon by rendering an array of small, intersecting spheres each centered on a sample point on the ultrasound slice and colored with the value of that sample point.

Ohbuchi, Chen, and Fuchs [Ohbuchi 1992] developed a system that incrementally resampled and rendered (via ray casting) ultrasound slice data. This work was expanded to a primitive augmented-reality system that ran at near-real-time frame rates (~1Hz) on the Pixel-Planes 5 graphics multicomputer [State 1994, State 1995]. The present work can best be described as improving on the results of that system by using new rendering algorithms and a different hardware platform to achieve real-time frame rates (10-15 Hz).

2.3 BSP Trees

A binary space partition (BSP) tree is a data structure for managing planar polygons in 3D. Intersecting polygons are divided into non-intersecting fragments as the tree is built. The structure can be traversed to produce non-intersecting primitives in low-to-high visibility order for any given viewpoint [Fuchs 1980].

BSP trees are most appropriate for static geometry with a moving viewpoint; the tree is built once and traversed many times [Fuchs 1983]. One serious drawback in using BSP trees with a changing data set is that while adding new objects requires only inserting the new primitive(s) into the tree (an inexpensive operation), removing geometry may require rebuilding the entire tree (discussed in greater detail in 4.1). [Chrysanthou 1996] shows how the rebuilding can sometimes be avoided by recombining the subtrees that remain after a piece of geometry is removed.

2.4 Contribution

Our work makes two contributions. First, it demonstrates real-time volume representations of sets of arbitrarily oriented slices of ultrasound data using BSP trees and texturing on a standard, commercial high-end graphics workstation. Second, it presents a method of parallel BSP trees to manage a dynamically changing set of ultrasound data and to minimize variation in per-frame BSP tree management times for more consistent overall frame rendering times.

3 RENDERING TEXTURED SLICES

Ultrasound echography data is captured from the scanner as a live, gray scale video image and is placed in texture memory at the start of the rendering loop in our application. Although the echography data is both generated by the ultrasound machine and captured by our system as 2D imagery, the samples (pixels/texels) are not planar samples. The region actually sampled is nominally wedge shaped (with the thicker end away from the ultrasound probe) due to the spreading characteristics of the sound waves used in echography. Thus each value in the 2D ultrasound image represents contributions from the values through the wedge.

In the interest of rendering speed, our rendering method does not reconstruct the volume or otherwise address the issue of data values in the space between slices. We model each ultrasound slice as a planar polygon and render it directly, with the ultrasound video image applied to it as a translucent texture. Ultrasound scans often comprise many closely-spaced slices, which diminishes the effect of neglecting inter-slice regions. Thus, although composed only of 2D polygons, our real-time images produce recognizable 3D structures.

The precise size, shape, position, and orientation of the slice of ultrasound data relative to the ultrasound probe is predetermined by a one-time calibration procedure [State 1994]. We track the ultrasound probe with a highly accurate mechanical tracker (FARO Technologies Metrecom IND-1). The probe calibration data, combined with the real-time tracking information, gives the 3D position and orientation of each polygon representing an ultrasound slice.

Proper opacity-accumulation compositing (one of the rendering modes implemented in our system) requires that slice polygons be non-intersecting and be presented in low-to-high visibility order (i.e., if polygon A obscures polygon B from the current viewpoint, B must be rendered before A). We use the BSP tree data structure for the slice polygons because it meets both these requirements.

We generate our volume representations by rendering polygons from the BSP tree with the appropriate textures applied. Using the various texturing modes of the Silicon Graphics RealityEngine^2 we can duplicate traditional volume rendering algorithms such as opacity accumulation and maximum intensity projection. We also use the texture lookup tables to adjust the brightness and opacity mappings for better discrimination of the target in our visualization.

One of the stated goals of the visualization is to provide a method of weighting data by age so that moving targets can be viewed. We implemented our system with exponential age-based attenuation (Plate 3).

Since the image is completely re-rendered from the BSP tree during every frame, we are able to attenuate slice image intensities and opacities as a rendering effect. Each slice polygon is time-stamped as the slice is captured from the ultrasound system. As each slice is inserted into the BSP tree and split into fragments, the slice's time stamp is propagated to the fragments. During tree traversal for rendering, the time stamps are used to compute each fragment's age and attenuate the polygon's brightness accordingly. The original slice data, stored in texture memory, are not modified.

4 DUAL BSP TREES

BSP trees work well for static data sets viewed from dynamically changing viewpoints. Our goal, however, is real-time visualization of the *n* most recently captured slices from a continuous data stream. This active set of data changes every frame, one new slice arrives and one old slice expires at each time step.

4.1 Continuously Captured Data

While adding new geometric primitives to a BSP tree requires only inserting the new primitives into the tree, removing a geometric primitive often requires rebuilding the entire sub-tree beneath the nodes that are deleted. As an alternative to rebuilding, expired slices can be flagged as “invisible” and not rendered during traversal, but such an operation doesn’t reduce the number of nodes in the tree. The constantly growing number of nodes causes greater fragmentation of newly-inserted geometry. Inserting into large trees is expensive, so the tree must occasionally be rebuilt to eliminate the expired nodes.

Despite the drawback of periodic rebuilding, BSP trees are attractive for our application since most of the slice polygons in one frame are present in the next. For example, if the active data set comprises 100 slices, 99% of the contents (99 slices) are displayed in the next frame. It pays to maintain a structure across time steps.

We solve the problem of deleting and rebuilding by maintaining two parallel BSP trees, out of phase in time. Consider that we want to render the *n* most recent slices every frame. When we start the system, we initialize an *active tree* and insert one slice into it in each of the first *n-1* frames (time steps) of the system. At frame *n-1*, we start a *replacement tree* and insert slices into both trees through frame *2n-2*. Note that after the slice insertion at time step *2n-1* the replacement tree will have *n* slices — the number needed to render the active set. Since rendering takes place after insertion, the first tree can be reinitialized at the end of step *2n-2* and the replacement tree becomes the active tree. Figure 2 illustrates this process for active set size *n*=4.

After startup all slices are inserted into both the active and replacement trees. Because of the way the trees overlap, each has a maximum of *2n-2* slices inserted before it is reinitialized. Each tree is active for display for *n-1* frames and flags *n-2* slices as expired before the trees are swapped and the older tree is reinitialized.

Time Step	Slices in Tree A	Slices in Tree B	Slices Displayed	Tree Used for Display
1	1	-	1	A
2	1-2	-	1-2	A
3	1-3	-	1-3	A
4	1-4	4	1-4	A
5	1-5	4-5	2-5	A
6	1-6	4-6	3-6	A
7	7	4-7	4-7	B
8	7-8	4-8	5-8	B
9	7-9	4-9	6-9	B
10	7-10	10	7-10	A
11	7-11	10-11	8-11	A
12	7-12	10-12	9-12	A
13	13	10-13	10-13	B
14	13-14	10-14	11-14	B
15	13-15	10-15	12-15	B
16	13-16	16	13-16	A
17	13-17	16-17	14-17	A
….	….	….	….	….

Figure 2: Behavior of dual BSP tree with an active set of *n*=4. Except for start-up in steps 1 through 3, the system always displays the 4 most recent samples. After start-up, the active BSP tree switches every third step. The double lines between cells indicate when the trees are reinitialized.

It is easy to see that the value of *n* (the number of slices displayed in each frame) can be changed interactively. If *n* is increased, we simply delay switching to the replacement tree and continue growing the existing trees to the new values of *n* and *2n*. Decreasing *n* can be accommodated by switching and starting the replacement tree early.

4.2 Analysis of Dual BSP Trees

A major requirement of our real-time system is to have a consistent update rate for the images presented to the user. Using a single BSP tree data structure for a changing data set requires that the tree be rebuilt occasionally. Figures 3 and 4 show that tree rebuilding (every *n*th frame after startup) causes spikes in the per-frame tree management time, thus making the frame update rate uneven.

The dual-tree approach amortizes the cost of rebuilding by making two insertions per frame: one into the active tree and one into the replacement tree. While each frame time is slightly longer, this scheme results in more even computational load per frame and a lower upper-limit on the per-frame tree management time.

The performance data in Figures 3 and 4 show the time (in milliseconds) required to insert a new slice in the BSP tree for each frame. Slice geometry data were captured from a typical ultrasound scanning session and were used to test the single- and dual-tree systems. The slice set is shown in Plate 4. The dual-tree system operated as described in 4.1. The single-tree system incrementally built a BSP tree to *2n* slices, at which point the tree was destroyed and a new tree of size *n* was built using the original unfragmented slice data.

Figure 3 shows that the maximum insertion time for any one frame was less than 5 ms with the dual-tree scheme but more than 30 ms with the single-tree scheme. Figure 4, which reflects the same sequence of tracking data but uses a larger active set, shows maximum times of 10 ms and 130 ms, respectively. In our visualization system, the total time budget to generate a new stereo frame is 100 ms, and managing the BSP tree(s) is only one of many tasks that must be handled. Reducing the maximum insert time from 30% to 5%, or 130% to 10%, of the frame time is a significant improvement in system performance. The lower variations and lower maximum times of the dual-tree approach are preferable in a real-time system where consistent update rate is important.

The memory space required by the dual tree system demonstrated in Figure 3 was only 20% greater than the storage required for the single BSP tree. This increase is negligible, as the overhead of representing slice geometry is tiny compared to the storage required for the textures that embody the ultrasound data. Each node (slice fragment) in the tree contains a list of vertices with texture coordinates, a few scalar properties, and a pointer to a stored texture map. While the geometry for each fragment occupies approximately 200 bytes, each texture occupies 64 kilobytes.

An issue not fully explored above is the impact of changing the size at which the BSP tree is rebuilt in the single-tree scheme. Here we rebuild after *2n* inserts, but we could easily choose to rebuild after *3n*, *1.7n*, etc. instead. We sidestep this question for two reasons. First, changing the maximum size of the single tree alters only the average per-frame computation time, not the peak time. The peaks occur at rebuilding time and

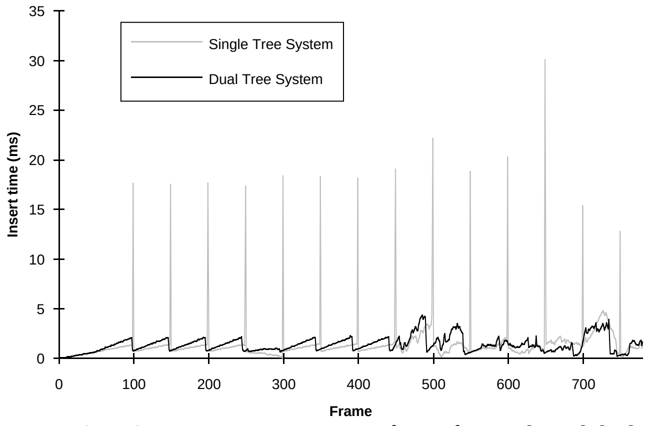

Figure 3: BSP tree insertion time per frame for single and dual tree systems with an active set of 50 slices.

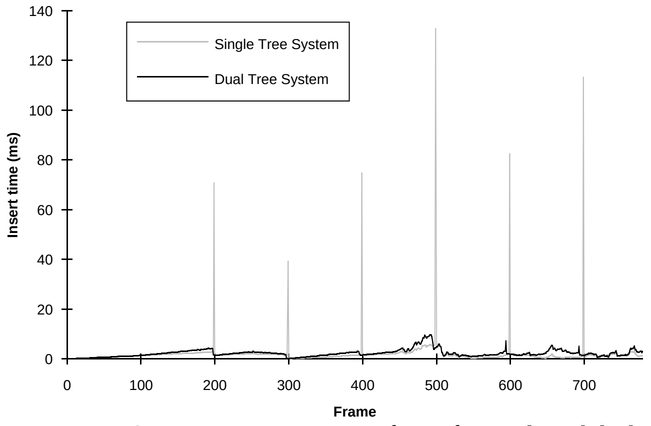

Figure 4: BSP tree insertion time per frame for single and dual tree systems with an active set of 100 slices.

depend on the minimum tree size (i.e., the size of the active set) rather than the maximum size. Second, a serious exploration of this parameter space would require testing with many more data sets and is not within the scope of this work. We experimented with a few values and found that the average times usually varied by only 5%.

5 GENERALIZATION TO *b* PARALLEL BSP TREES

In our system we use dual BSP trees, but we also investigated a generalization to *b* parallel BSP trees. With two trees, just before swapping, the active tree holds *2n-2* slices and the replacement tree holds *n-1* (where *n* is the number of slices to be shown at any time). With more than two trees, trees are deleted and created more frequently and fewer inserts are made to any one tree before it is reinitialized (so the maximum size of the trees decreases). Figure 5 shows how *b* trees are managed in parallel and Figure 6 summarizes their characteristics.

Increasing the number of trees in the system presents a performance tradeoff: an extra insertion must be performed every frame for each tree added, but each additional tree decreases the maximum size any tree is allowed to reach before being deleted. The question is, essentially, whether it is better to have numerous small trees or few larger ones. The answer

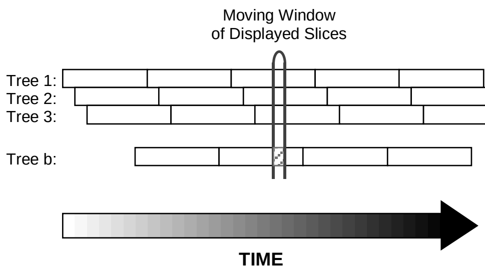

Figure 5: Rendering from and maintaining a system of *b* parallel BSP trees.

#trees	2	3	4	b
max. size of tree	$2n-2$	$\frac{3n-3}{2}$	$\frac{4n-4}{3}$	$\frac{b(n-1)}{b-1}$
tree added every #frames	$n-1$	$\frac{n-1}{2}$	$\frac{n-1}{3}$	$\frac{n-1}{b-1}$
inserts per frame	2	3	4	b

Figure 6: Characteristics of parallel BSP tree systems.

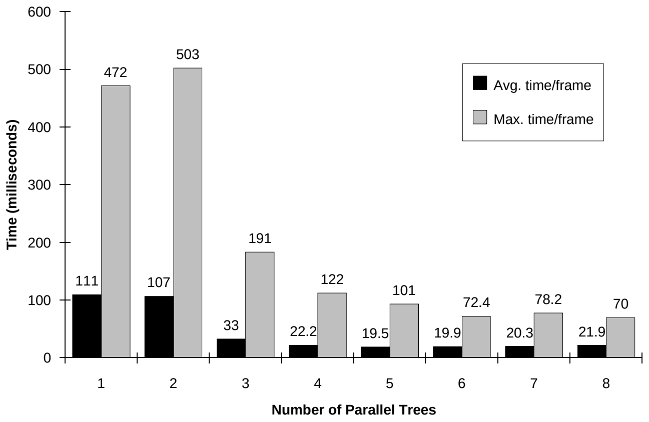

Figure 7: Average and maximum times per frame to insert geometry into systems of 1-8 trees. The geometry data set used here is different from that used in Figures 3-4 and is shown in Plate 5.

depends on the nature of the geometry data and how much it costs to insert it into the trees.

[Fuchs 1980] shows that BSP trees can have as many as O(n^3) internal nodes (fragments) after n inserts. Insertion time is roughly proportional to the number of internal nodes; in the worst case, an incoming slice must be checked and split against each fragment in the BSP tree. Geometry data and BSP tree statistics compiled from several ultrasound scans show that, in our application, BSP trees grow at between O(n) and O(n^2) during a typical scan, and that the cost of insertion varies directly but with the internal size of the tree. When the growth is faster than O(n), a system of more than two trees is optimal.

Again, the growth of the tree depends on the geometric nature of the data. Figure 7, below, shows the results for various values of b for a different set of geometry data than used in previous comparisons. The numbers suggest that a choice of b=5 or b=6 (i.e., 5 or 6 parallel BSP trees) is optimal in this case. The geometry data that produced these results, contrived to produce high levels of fragmentation, is shown in Plate 5.

6 CONCLUSIONS AND FUTURE WORK

We have presented a method and demonstrated a system for incrementally rendering 2D ultrasound data in real time to create a recognizable visualization of a 3D anatomical target. Small cysts and other imaging targets look three-dimensional; we can see needle tracks in a training phantom (Plate 1).

Representing 2D ultrasound slices as planar, textured polygons allows us to avoid resampling the data and allows us to take advantage of hardware accelerated texturing in the Silicon Graphics RealityEngine^2. We chose a BSP data structure to meet the requirement that primitives be presented in low-to-high visibility order for proper compositing. We manage the continuous data stream and reduce maximum tree management per frame times via parallel, time-shifted BSP trees.

Many issues remain. 3D volumetric display of anatomical features in an augmented-reality environment is a new metaphor for physicians. Issues such as number of slices to display, decay factor for older slices, and user interface need to be explored.

Since we do not use space-filling interpolation between ultrasound slices, the intensity and thus the useful visual content of the rendered image varies greatly depending on whether slices are viewed mostly face-on or mostly edge-on. We need to address this fundamental problem of the 2D primitives we render.

We continue to work on improvements in image quality. Ultrasound images tend to be fairly noisy, exhibiting problems such as speckle and reflection. Improving rendering quality and discernability of the target anatomy while maintaining real-time update rates is the challenge ahead of us.

ACKNOWLEDGMENTS

We thank David Chen, Arthur Gregory, David Harrison, Mark Livingston, Etta Pisano (MD), and Chris Tector for their help with the system. We thank the anonymous reviewers for their comments and criticism. PIE Medical Equipment B.V. generously provided us with an ultrasound machine. This work was supported in part by ARPA DABT63-93-C-0048. Approved by ARPA for Public Release—Distribution Unlimited. Additional support was provided by the National Science Foundation Science and Technology Center for Computer Graphics and Scientific Visualization (NSF prime contract 8920219).

REFERENCES

[Akeley 1993]
Akeley, Kurt. “RealityEngine Graphics.” *Proceedings of SIGGRAPH '93* (Anaheim, CA, August 1-6, 1993). In *Computer Graphics Proceedings*, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. 109-116.

[Bajura 1992]
Bajura, Michael, Henry Fuchs, and Ryutarou Ohbuchi. “Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient.” *Proceedings of SIGGRAPH ’92* (Chicago, Illinois, July 26-31, 1992). In

Computer Graphics 26, 2 (July 1992), 203-209.

[Cabral 1994]
Cabral, B., Cam, N., and Foran, J. "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware." *Proceedings of the 1994 Symposium on Volume Visualization* (Washington, DC, October 17-18,1994), pp. 91-98.

[Chrysanthou 1996]
Chrysanthou, Yiorgos. "Shadow Computation for 3D Interaction and Animation." *Ph.D. Thesis, University of London* (January 1996).

[Cullip 1994]
Cullip, Timothy and Ulrich Neumann. "Accelerating Volume Reconstruction With 3D Texture Hardware." *UNC Technical Report* TR93-027 (May 1994).

[Fuchs 1980]
Fuchs, Henry, Zvi Kedem, and Bruce Naylor. "On Visible Surface Generation by a Priori Tree Structures." *Proceedings of SIGGRAPH '80* (July 1980). In *Computer Graphics* 14, 3 (July 1980), 124-133.

[Fuchs 1983]
Fuchs, Henry, Gregory Abram, and Eric Grant. "Near Real-time Shaded Display of Rigid Objects." *Proceedings of SIGGRAPH '83* (July 1983). In *Computer Graphics* 17, 3 (July 1983), 65-72.

[Herman 1979]
Herman, Gabor and Hsun Kao Liu. "Three-Dimensional Display of Human Organs from Computed Tomograms." *Computer Graphics and Image Processing,* 1979, 1-21.

[Lengyel 1995]
Lengyel, Jed, Donald Greenberg, and Richard Popp. "Time-Dependent Three-Dimensional Intervascular Ultrasound." *Proceedings of SIGGRAPH 95* (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics Proceedings,* Annual Conference Series, 1995, ACM SIGGRAPH, pp. 457-464.

[Levoy 1988]
Levoy, Marc. "Display of Surfaces from Volume Data." *IEEE Computer Graphics and Applications* 8, 5 (May 1988), 29-37.

[Nelson 1993]
Nelson, Thomas and Todd Elvins. "Visualization of 3D Ultrasound Data." *IEEE Computer Graphics and Applications* (November 1993), 50-57.

[Ohbuchi 1992]
Ohbuchi, Ryutaro, David Chen, and Henry Fuchs. "Incremental Volume Reconstruction and Rendering for 3D Ultrasound Imaging." SPIE Vol. 1808 *Visualization in Biomedical Computing 1992,* 312-323.

[Ohbuchi 1994]
Ohbuchi, Ryutarou. "Incremental Acquisition and Visualization of 3D Ultrasound Images." Ph.D. Thesis. UNC 1994-0362 (1994).

[Sakas 1995]
Sakas, Georgios and Stefan Walter. "Extracting Surfaces from Fuzzy 3D-Ultrasound Data." *Proceedings of SIGGRAPH 95* (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics Proceedings,* Annual Conference Series, 1995, ACM SIGGRAPH, pp. 465-474.

[Shirley 1990]
Shirley, Peter and Allan Tuchman. "A Polygonal Approach to Direct Scalar Volume Rendering." *Computer Graphics* 24, 5 (November 1990), 63-70.

[State 1994]
State, Andrei, David 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 IEEE Visualization '94* (Washington, DC, October 17-21, 1994).

[State 1995]
State, Andrei, Jonathan McAllister, Ulrich Neumann, Hong Chen, Timothy Cullip, David Chen, and Henry Fuchs. "Interactive Volume Visualization on a Heterogeneous Message-Passing Multicomputer." *Proceedings of the 1995 Symposium on 3D Interactive Graphics* (Monterrey, CA, April 9-12, 1995), pp. 69-74.

[State 1996]
State, Andrei, Mark Livingston, William Garrett, Gentaro Hirota, Mary Whitton, Etta Pisano, and Henry Fuchs. "Technologies for Augmented-Reality Systems: Realizing Ultrasound-Guided Needle Biopsies." *Proceedings of SIGGRAPH 96* (New Orleans, Lousiana, August 4-9, 1996). In *Computer Graphics Proceedings,* Annual Conference Series, 1996, ACM SIGGRAPH.

[Stein 1994]
Stein, Clifford M., Barry Becker, and Nelson Max. "Sorting and Hardware Assisted Rendering for Volume Visualization." *Proceedings of 1994 Symposium on Volume Visualization* (Washington, DC, October 17-18,1994), pp. 83-89.

[Thune 1991]
Thune, Nils and Bjørn Olstad. "Visualizing 4-D Medical Ultrasound Data." *Proceedings of Visualization 1991* (San Diego, CA, October 22-25, 1991), 210-215.

[Tuy 1984]
Tuy, Heang, and Lee Tan Tuy. "Direct 2-D Display of 3-D Objects." *IEEE Computer Graphics and Applications* 4, 10 (November 1984), 29-33.

[Westover 1990]
Westover, Lee. "Footprint Evaluation for Volume Rendering." *Proceedings of SIGGRAPH '90* (August 1990). In *Computer Graphics* 24, 4 (1990), 367-376.

[Watkin 1993]
Watkin, K., L. Baer, S. Mathur, R. Jones, S. Hakim, I. Diouf, B. Nuwayhid, and S. Khalife. "Three-Dimensional Reconstruction and Enhancement of Arbitrarily Oriented and Positioned 2D Medical Ultrasonic Images." *IEEE Canadian Electrical and Computer Engineering: Proceedings* (1993), 1188-1195.

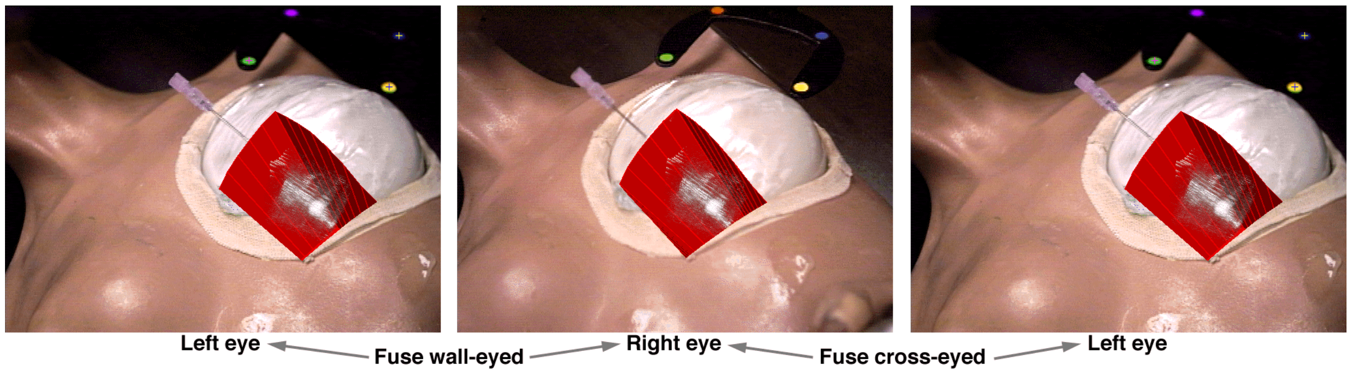

Plate 1. Head-mounted display view from an augmented reality system designed to assist a physician with ultrasound-guided needle biopsy of the breast. A cyst aspiration needle has been inserted into a simulated lesion within the breast phantom. The needle is visually aligned with its scanned image. The lesion is the white blob inside the computer-generated opening (red) within the breast.

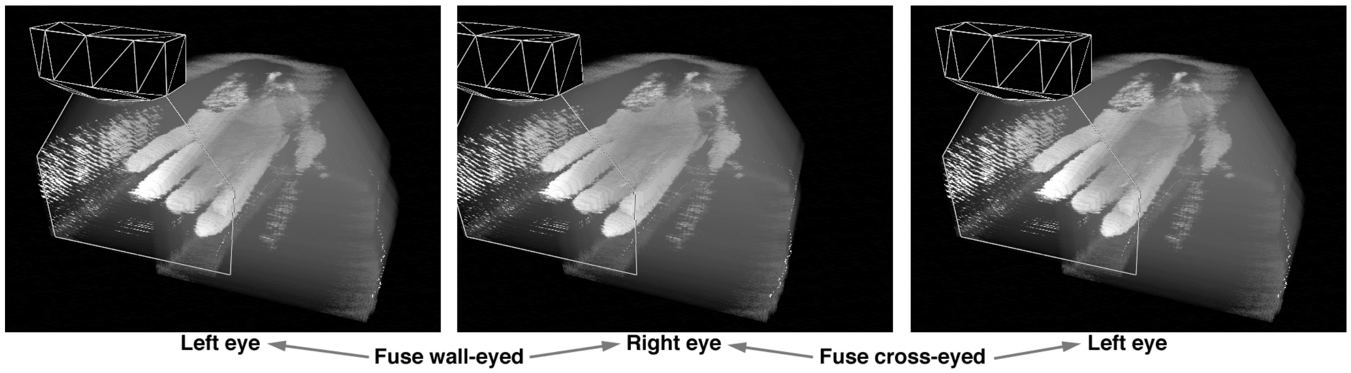

Plate 2. Scanning a motionless human hand (left hand, palm up). The wireframe object near the top is the ultrasound probe and the wireframe polygon below it shows the slice imaged most recently. The volume consists of over 100 planar, translucent, textured polygons "emitted" during a 10-second sweep along a U-shaped path.

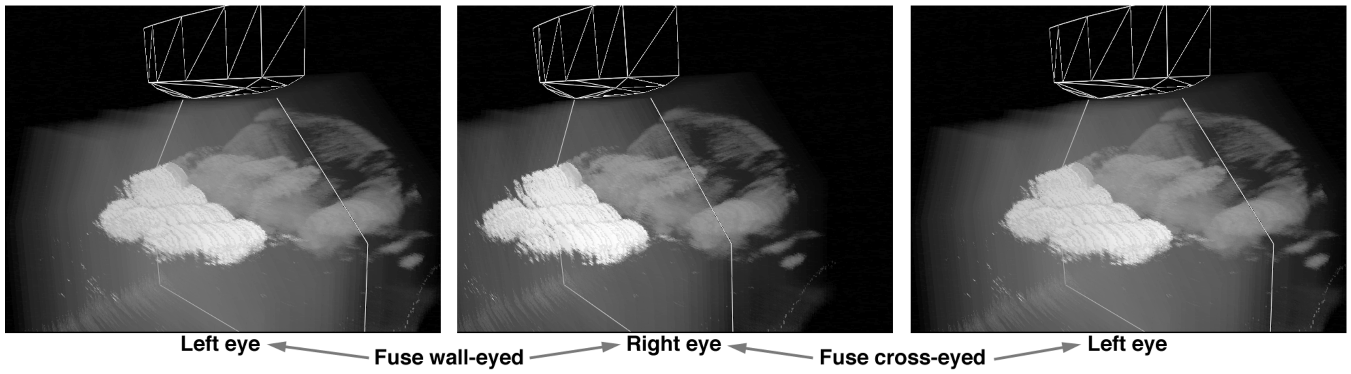

Plate 3. Scanning a moving human hand. The sweep started at the wrist and scanned the closed fist; then the hand opened and the probe scanned the fingertips. Polygon intensity is progressively attenuated by age, displaying the (older) fist faintly and the recently-imaged outstretched fingers brightly. This "3D radar" effect depicts decreasing confidence in regions of space that have not been scanned recently.

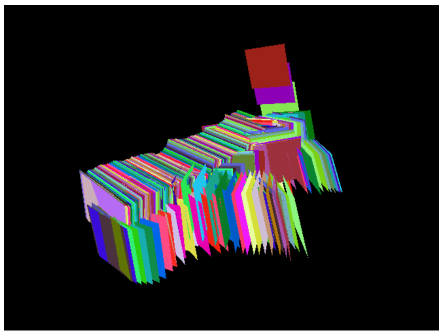

Plate 4. A slice geometry data set captured during a typical ultrasound scanning session. Slices are drawn as simple colored polygons to emphasize intersections.

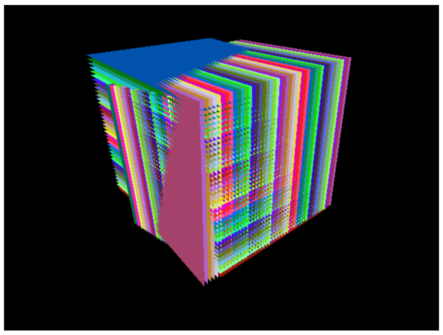

Plate 5. A slice geometry dataset contrived to produce numerous intersections.