Interactive Visualization of 3D Medical Image Data

Bradley M. Hemminger, Timothy J. Cullip†, Michael J. North

Departments of Radiology and Radiation Oncology†
University of North Carolina at Chapel Hill, NC 27599-7510
(919) 966-2998 bmh@rad.unc.edu

1. Introduction

Over the past fifteen years the field has progressed from coarse 3D computer graphic renderings to high quality 3D visualizations of medical image data. The rendering times for creating these images have also significantly decreased from periods of days in 1980 to tens of seconds on standard workstations in 1992. However, very limited clinical use has been made of 3D renderings at this time. Probably the most significant reason for this is the lack of interactive user control of the renderings. 3D visualizations hold great promise if the user can interactively segment and classify the data, and interactively choose viewpoint and rendering parameters to optimally visualize information present in the datasets.

In 1993, for the first time, commercially available workstation hardware is capable of real-time 3D volume rendering of medical image data in a clinically useful fashion. Our experience at UNC using non-interactive and interactive renderings indicates that three primary elements are necessary for a clinically successful real-time interactive volume rendering tool: 1) the user must be able to interactively classify the data to visualize difficult to define "surfaces", such as soft-tissue/soft-tissue boundaries in the chest and abdomen, as well the more easily defined boundaries such as bone/soft-tissue and skin/air; 2) the user must be able to interactively and conveniently choose the parameters determining the rendering of the scene (i.e. arbitrary viewpoint, zoom, lighting conditions, rendering algorithm); and 3) the ability to cut away the volume to reveal occluded portions of the anatomy (using cutting planes or sculpting tools).

A few prototype very high speed rendering engines have been developed that support realtime volume rendering, including Pixel Planes[1], and the Princeton Engine[2]. While these systems have allowed research into realtime 3D volume rendering, there have not previously been commercially available systems supporting the speeds necessary for interactive volume rendering. This paper will concentrate on the two new commercially available systems, the Denali from Kubota Pacific Computer Inc. (Kubota) and the Reality Engine based systems from Silicon Graphics Inc. (SGI), which we found capable of real-time, clinically useful, volume rendering.

The second section of this paper will provide background on what 3D volume rendering is, and describe the state of the art in volume rendering as applied to medical imaging. The third section of the paper will discuss several aspects of realtime volume rendering: hardware requirements for realtime volume rendering of medical image volumes; descriptions of Denali and Reality Engine systems; benchmarks of these two systems

and analysis of their performance; and, a complete description of the algorithm we have implemented on the Reality Engine. Finally, the fourth section of the paper discusses the preliminary results of using our realtime volume rendering application, SeeThru, for clinical cardiothoracic surgery planning at UNC Hospitals.

2. Background

As computers with video displays became accessible to researchers during the 1970s and 1980s, people began to experiment with displaying medical image data on them. Initially, most work was to display collections of individual 2D images from film on video displays, especially images from the newer, digital modalities (nuclear medicine, CT, MR, PET). As computing power increased, developments in the field of computer graphics led to the design of algorithms that would depict 3D structures on 2D display screens so that the human observer would have a sense of the 3D scene. The appearance of a 3D scene from a static 2D image was accomplished using visual cues such as occlusion, perspective, shading. and stereo (when using stereo display and glasses). These results were then applied to medical image data that could be acquired as a 3D volume. Initially, this was mainly sets of parallel 2D slices in 3D space, which were then treated as a 3D volume sample. More recently, acquisition techniques have been developed allowing for the acquisition of actual volumes samples. In either case, these sets of volume elements, usually referred to as voxels, make up a 3D volume that can be displayed using computer graphic techniques on a 2D screen.

The original methods used to display the 3D scene as a 2D image on the video display are now referred to as *surface rendering* methods. These methods required the determination of a surface of interest from the 3D volume. Identifying the surface of an object of interest required generating a contour on each 2D slice. This meant hand contouring the object on each slice, or using image processing techniques (thresholding, region growing/dilation) to semi-automate the process, and then following this by a method to create a polygonal surface from the contours[3]. These methods had two significant disadvantages. First, they required time consuming contouring steps, often requiring highly skilled medical personnel to do the contouring and follow-up editing. Second, they drastically reduce the information content of the study by making a binary decision as to where surfaces lie, thus reducing the dataset from a 3D volume of density values, to a list of surfaces. Reducing the 3D volume to a list of polygonal surfaces creates a very compact representation, that could be rendered fairly quickly on graphics workstations, which generally had hardware support for rendering polygonal type objects.

In the last several years, as computer power has increased and hardware prices have declined, experimentation with actually rendering the entire data volume has become possible. Rendering directly the entire volume without the intermediate step of defining surfaces is generally referred to as *volume rendering*. Volume rendering methods model the voxels as objects in space and calculate the interaction of light with these objects as seen from the observer's viewpoint. The treatment of voxels as different types of objects (point samples, blobs, cubes, etc.) with different possible reflectance and scatter characteristics has lead to the creation of many different volume rendering techniques. A general discussion of volume rendering can be found in

Drebin[4], and a recent survey of volume rendering techniques applied to medical imaging can be found in Yoo[5].

The 2D images representing 3D volumes are even more realistic to the observer if they can be interactively rotated in space via user control due to kinetic depth effect.[6,7] Since this requires at least 5-10 frames per second update rates to maintain the visual precept of a single object in continuous motion, current workstations have not been capable of realtime volume rendering. However, as computation times have decreased to seconds and minutes, the *cine* presentation has been increasingly popular. This involves the precalculation of multiple angles of view along a rotation of the object. These precomputed individual views are then displayed as a cine sequence over the predefined rotation, and this provides kinetic depth effect benefits similar to those of interactive control of the rotation.

Currently, as seen at RSNA 1993, commercially available presentations generally support

- non-interactive high quality 3D image renderings
- pre-computed cine presentations of low and high quality renderings
- near-realtime presentations of less complex volume rendering methods such as MIP, IsoSurface, and rendering by intermediate processing to polygonal surfaces via Marching Cubes or similar algorithms.

However, a very significant development occurred in 1993. For the first time, commercially available off the shelf workstation hardware is capable of realtime high quality volume rendering. The two commercial systems supporting realtime rendering are the Kubota Denali and the SGI Reality Engine.

3. Realtime Volume Rendering Systems

The requirements of a realtime volume rendering systems can be estimated for an average CT study of 512x512x64 (2^{24}) voxels. In order to render the image data, the volume renderer essentially has to sample at the resolution of all the voxels, or perhaps slightly finer. When sampling, the desired (x,y,z) position at which the volume should be sampled generally does not fall directly on existing voxels, and must be properly estimated from its nearest neighbors. This requires performing an operation referred to as a trilinear interpolation. The simplest 3D neighboring sampling requires accessing each of the 8 adjacent neighbors and results in performing about 23 additions and 12 multiplications per sample. Finally, in order to have interactive rendering we need at least 10 frames a second. Thus a lower bound estimate of required operations would be

$$2^{24} \textit{ samples } * \textit{ about 30 low level operations } * \textit{ 10 frames second}$$

or about 5 Giga-operations per second. The best measure of speed is the resulting number of frames per second for a certain rendering method. Because this depends on several variables, it is often more convenient to use the number of 3D trilinear interpolations per second that the renderer can perform to compare systems since this is the basic common operation. Thus to achieve ten frames per second for the above CT study a system should be capable of approximately $2^{24} * 10$, or 160 million trilinear interpolations per second.

The major problem in accomplishing this goal is the size of the dataset to be held in memory (2^{24} for our CT example), and the rate which it must be sampled. Most

methods that have attacked this problem have done so by some form of parallelization, either in screen space or in image space. This essentially divides the problem into smaller pieces that can be rendered more quickly, reducing the size of the problem. The tradeoffs are the partial redundancy in copies of the image voxels and communication between subparts during rendering, especially during the recombining of information. As computer power increases and prices drop, at some point we would expect to see direct hardware support for holding an entire volume of data as a single entity and rendering it directly. One method of accomplishing this is utilizing the 3D texture map memories on high end graphics engines. These 3D texture memories were developed to allow texturing of 3D objects in addition to 2D ones, to further support the needs of highly realistic rendering of synthetic scenes. Importantly, however, these 3D texture memories are essentially the 3D volume resampling mechanism needed for directly rendering the entire volume. The sampling of the volume from texture memory is accomplished by sampling planes through the entire volume texture perpendicular to the viewpoint, and compositing the results of each sampled plane to create the final 2D image. Because the planar sampling from texture memory and the compositing have hardware support, the renderings can be accomplished in realtime. We refer to this technique as *planar texture volume sampling*, and a description of our algorithm as implemented on the SGI Reality Engine is given in section 4.

3.1 Kubota Denali

The Kubota Denali is comprised of Transform Engine Modules (TEM) and *Frame Buffer Modules* (FBM) modules. The Denali system supports having up to 6 TEMs and up to 20 FBMs. The TEMs perform the first stages of the graphics rendering pipeline, geometry transforms, lighting and shading calculations, and scan conversion. The FBMs are responsible for per pixel operations. To achieve realtime performance the full volume is partitioned into pieces which are stored on separate FBMs for parallel processing. The pixel engines on each FBM support 600,000 trilinear interpolations per second, allowing a maximally configured Kubota (20 FBMs) to achieve approximately 12 million trilinear interpolations per second. The Denali has direct support for manipulating and rendering volume datasets in the Kubota Volume Extension to the Kubota X server. At the time of our benchmarking the system in October 1993, the Kubota Volume Extension supported projection summing (maximum intensity projection, minimum intensity projection, and ray sum projection) and IsoSurface rendering methods. In follow-up discussions with Kubota, they have indicated that opacity based compositing methods are under development.

3.2 SGI Reality Engine

The SGI Reality Engine currently supports the largest 3D texture memory size. On the Reality Engine volumes can be held in the 3D texture memory on the raster manager (RM) boards. Parallelization of work can be done through the use of multiple raster manager (RM) boards, each of which has its own independent copy of the entire texture memory. The Reality Engine supports 1, 2 and 4 board configurations. Our initial work has been done using the RM4 boards which hold 4 Mbytes of texture memory data. In January of 1994, SGI released their RM5 boards which hold 16 Mbytes of texture memory. We benchmarked a single Reality Engine RM4 board at approximately 37 million trilinear interpolated samples per second from the 3D texture memory, and the two raster board system at twice that amount. Using four boards should quadruple the single board rate. We have implemented a planar texture volume

resampling method that supports several rendering methods including MIP, Xray Projection, opacity-based compositing, and gradient-based compositing. The Reality Engine allows several different configurations of texture memory, leading to many different possible rendering combinations. Under our current implementation, the rendering times are all equivalent because the size of information stored and manipulated for each voxel is 16 bits, so for the benchmarks we used the opacity based method.

Only the portions of the Denali and Reality Engine archictectures pertinent to realtime volume rendering have been described. The Kubota system is described more completely in their technical white paper documents,[8,9,10] and their realtime volume rendering is more completely described by Guan.[11] A technical description of the SGI Reality Engine is available as a technical report,[12] and the upcoming paper[13] will describe the application of the Reality Engine to volume rendering. Additionally, more in-depth discussions of realtime volume rendering archictectures, such as Pixel Planes,[1] Pixel Flow,[2] and Princeton Engine,[2,24,25] can be found in computer graphics literature. Finally, both the Pixel Planes (under Division) and the Princeton Engine (under David Sarnoff Labs) projects are also being turned into commercial products.

3.3 Benchmarks

While trilinear interpolations per second may provide the most convenient measure for comparison between systems, the best measure of performance is the actual frames per second achievable. This section describes our testing of the two systems to measure frames per second across a variety of variables. While we have tried to match the rendering tests between the Denali and Reality Engine to be as similar as possible, there are several small differences between the tests on the different systems. Thus, one should not use these results to directly compare the two systems. Additionally, the best test is to take your sample datasets and render them on each system yourself using the technique and parameters appropriate to your application. Due to space considerations, only the results of a few representative datasets, taken from the larger set of all benchmarks performed, are presented.

Kubota Denali

These results are based on two days of tests during October of 1993 at the UNC department of Radiology. The results for four different studies are given in tables 1 through 4. The studies were rendered using either MIP or IsoSurface algorithms, with either Point or Trilinear sampling. The resulting image was rendered to one of 256^2, 512^2, or 1024^2 window sizes. In general, the resulting rendered image occupied about half to two thirds of the window size. Additionally, the configuration of the Denali was varied through 6 different configurations (by adding or removing TEM cards and/or FBM cards). The rendering application used was the *volumetric demo* program provided by Kubota to demonstrate the volume rendering capabilities of their system. Tables 1 and 2 show benchmark data for two datasets, with the rendering methods, output window size, and sampling method varied. Tables 3 and 4 show two different datasets with rendering parameters constant, and only the configuration of the Denali varied. Finally, the frames/sec value for the Denali are calculated based on stopwatch measurements, not the more accurate method of having timestamps directly from the program, because we could not modify the demo program. Each measurement is the average of 3 repeats of the same test. One should expect error in stopwatch

measurements, on the order of 0.02% to 0.06% (assuming .3 second error on pressing stopwatch for start and end, for the measured intervals of 10 to 30 seconds that were utilized).

Configuration	*Dataset Size*	*Window Size*	*Algorithm*	*Sampling*	*Fr/Sec*
6/20	512x512x64	1024x1024	MIP	Trilinear	0.63
6/20	512x512x64	1024x1024	MIP	Point	1.20
6/20	512x512x64	512x512	MIP	Trilinear	2.30
6/20	512x512x64	512x512	MIP	Point	3.20
6/20	512x512x64	256x256	MIP	Trilinear	7.64
6/20	512x512x64	256x256	MIP	Point	11.98
6/20	512x512x64	1024x1024	IsoSurface	Trilinear	0.55
6/20	512x512x64	1024x1024	IsoSurface	Point	0.90
6/20	512x512x64	512x512	IsoSurface	Trilinear	1.60
6/20	512x512x64	512x512	IsoSurface	Point	2.80
6/20	512x512x64	256x256	IsoSurface	Trilinear	6.40
6/20	512x512x64	256x256	IsoSurface	Point	9.88

Table 1

Configuration	*Dataset Size*	*Window Size*	*Algorithm*	*Sampling*	*Fr/Sec*
6/20	256x256x128	1024x1024	MIP	Trilinear	0.50
6/20	256x256x128	1024x1024	MIP	Point	1.00
6/20	256x256x128	512x512	MIP	Trilinear	0.98
6/20	256x256x128	512x512	MIP	Point	1.89
6/20	256x256x128	256x256	MIP	Trilinear	4.05
6/20	256x256x128	256x256	MIP	Point	7.13
6/20	256x256x128	1024x1024	IsoSurface	Trilinear	0.41
6/20	256x256x128	1024x1024	IsoSurface	Point	0.78
6/20	256x256x128	512x512	IsoSurface	Trilinear	0.97
6/20	256x256x128	512x512	IsoSurface	Point	1.46
6/20	256x256x128	256x256	IsoSurface	Trilinear	3.67
6/20	256x256x128	256x256	IsoSurface	Point	6.14

Table 2

Configuration	*Dataset Size*	*Window Size*	*Algorithm*	*Sampling*	*Fr/Sec*
6/20	256x256x32	512x512	MIP	Trilinear	4.92
6/10	256x256x32	512x512	MIP	Trilinear	3.07
3/10	256x256x32	512x512	MIP	Trilinear	2.95
1/10	256x256x32	512x512	MIP	Trilinear	3.03
1/20	256x256x32	512x512	MIP	Trilinear	4.00
2/20	256x256x32	512x512	MIP	Trilinear	4.42

Table 3

Configuration	*Dataset Size*	*Window Size*	*Algorithm*	*Sampling*	*Fr/Sec*
6/20	128x128x55	512x512	MIP	Trilinear	5.45
6/10	128x128x55	512x512	MIP	Trilinear	3.30
3/10	128x128x55	512x512	MIP	Trilinear	3.23
1/10	128x128x55	512x512	MIP	Trilinear	2.91
1/20	128x128x55	512x512	MIP	Trilinear	3.72
2/20	128x128x55	512x512	MIP	Trilinear	4.59

Table 4

SGI Reality Engine

Benchmarks on the SGI Reality Engine were performed using our prototype renderer developed to take advantage of the large 3D texture memory on the SGI Reality Engine systems. This renderer is described in the next section. Because we were using our own program, we were able to output timestamps to exactly measure the rate at which the renderer operated. Timestamps were set to record the time duration required for 10 frame updates. By interactively manipulating the 3D dataset via the mouse interface to cause screen redraws at near continuous rates, measurements were made of ten second frame updates. At least five measurements were made for each condition, and the average is reported. Figures 5 and 6 show benchmark results for 256x256x32 and 128x128x64 size datasets, respectively. Two parameters of the rendering were varied i figures 5 and 6: the sampling rate and the zoom factor. The sampling rate was measured at 1.0 (sampling step size equal to voxel size), 0.50 (sampling at half voxel size steps), and 0.25 (sampling at quarter voxel size steps). The zoom factor was measured at 0.90 and 0.70, which corresponded to the viewed image occupying the entire display window, and the viewed image occupying approximately three quarters of the display window. The display window in all cases was 512^2.

Configuration	*Dataset Size*	*Window Size*	*Zoom*	*Sampling*	*Fr/Sec*
RE1 1 RM4	256x256x32	512x512	0.90	1.00	5,84
RE1 1 RM4	256x256x32	512x512	0.70	1.00	7.86
RE1 1 RM4	256x256x32	512x512	0.90	0.50	2.79
RE1 1 RM4	256x256x32	512x512	0.70	0.50	4.11
RE1 1 RM4	256x256x32	512x512	0.90	0.25	1.38
RE1 1 RM4	256x256x32	512x512	0.70	0.25	1.85

Table 5

Configuration	*Dataset Size*	*Window Size*	*Zoom*	*Sampling*	*Fr/Sec*
RE1 1 RM4	128x128x64	512x512	0.90	1.00	4.47
RE1 1 RM4	128x128x64	512x512	0.70	1.00	6.96
RE1 1 RM4	128x128x64	512x512	0.90	0.50	2.26
RE1 1 RM4	128x128x64	512x512	0.70	0.50	3.60
RE1 1 RM4	128x128x64	512x512	0.90	0.25	1.14
RE1 1 RM4	128x128x64	512x512	0.70	0.25	1.76

Table 6

Configuration	*Dataset Size*	*Window Size*	*Zoom*	*Sampling*	*Fr/Sec*
RE2 2 RM4s	256x256x32	512x512	0.90	1.00	8.26
RE2 2 RM4s	256x256x32	512x512	0.70	1.00	11.39
RE2 2 RM4s	256x256x32	512x512	0.90	0.50	4.12
RE2 2 RM4s	256x256x32	512x512	0.70	0.50	5.41
RE2 2 RM4s	256x256x32	512x512	0.90	0.25	1.77
RE2 2 RM4s	256x256x32	512x512	0.70	0.25	2.94

Table 7

Configuration	*Dataset Size*	*Window Size*	*Zoom*	*Sampling*	*Fr/Sec*
RE2 2 RM4s	128x128x64	512x512	0.90	1.00	8.03
RE2 2 RM4s	128x128x64	512x512	0.70	1.00	11.82
RE2 2 RM4s	128x128x64	512x512	0.90	0.50	4.23
RE2 2 RM4s	128x128x64	512x512	0.70	0.50	6.23
RE2 2 RM4s	128x128x64	512x512	0.90	0.25	2.13
RE2 2 RM4s	128x128x64	512x512	0.70	0.25	3.12

Table 8

3.4 Analysis of Systems and Benchmarks

Analysis and discussion is given below for each of the variables tested during the benchmarks.

System configuration

The majority of the testing was all done with the maximum Denali configuration of 6 TEMs and 20 FBMs (tables 1 and 2). The final tests, shown in tables 3 and 4, list the six different configurations tested. While we did not test all possible variations (1-6 TEMs and 5,10, or 20 FBMs), the tested configurations show the pertinent trends for realtime volume rendering. The most important factor was the number of FBMs, as would be expected since increasing the FBMs increases the parallelization. The second, and much less important factor, was the number of TEMs. Most of this increase in performance was going from 1 to 2 TEMs and not in adding additional TEMs. Thus,

it may be more cost effective to purchase a 2/20 system which has nearly the performance of the 6/20 with only one third of the TEMs.

The Reality Engine was only tested on 1 and 2 RM4 board configurations. Doubling the number of raster boards could, at best, double the rendering rate. In our tests we found that in all cases performance was linearly faster, but slightly less than 2 times faster, when comparing the one raster board to the two raster board system. The slight difference is probably due to the effect of constant overhead in the processing. We have used a four RM5 board system in technical demonstrations at two conferences, but have not had the opportunity for controlled testing. Our impression is that it is faster; however, with slightly less of a speedup than was seen in going from a 1 board to 2 board system.

Dataset size

A limitation of the current systems that use the 3D texture memory for planar texture volume resampling is the size of the dataset. The Reality Engine used in our tests had RM4 boards which hold 4 Mbytes. The minimum usage per voxel is 16 bits as in our benchmarked algorithm (other methods such as storing 3D gradient would require more per voxel space), so this implies a maximum of 2 Mega-voxels, or equivalently a 128x128x128 volume. The new RM5 boards hold 16 Mbytes and thus could hold 256x256x128 or 512x512x32 volumes, which is nearly the complete CT study we used as an example earlier. On the other hand, systems like the Kubota, which partition the volume memory into separate pieces for parallel processing may be capable of holding larger datasets. The maximum Denali with 20 FBMs can hold 80 Mbytes of information. Kubota indicates that about half of this can be used for the volume dataset (the other half for intermediate results and depth buffer), and that assuming 16 bit voxels, volumes of 256x256x253 or 512x512x64, can be handled. Note that for both systems, larger datasets can be handled by partitioning the complete volume into smaller blocks, and rendering the blocks separately. The tradeoff is that realtime update rates are usually lost because there is a significant delay in loading data from host memory to the faster raster boards (Reality Engine) or FBMs (Denali), and because more voxels are being rendered. If successive refinement techniques are acceptable for an visualization, then larger datasets may be handled satisfactorily by interactively rendering a lower resolution version of the dataset, and updating it with a full resolution version between interactions.

Window size

The output window size on the Denali can be arbitrarily sized. It was tested at 1024^2, 512^2, and 256^2 sizes. In general the image rendered in the window occupied one half to two thirds of the window size. While the rendering times were less for smaller sized windows compared to larger size windows, there was not a consistent proportional relationship as one might expect. This may be due, in part, to our not being able to completely specify the desired sampling on the volumetric demo program. It did not provide feedback or user control over the number of pixels per plane on resampling. Additionally, the Denali can perform fast 2D interpolation to resample the resulting window from one size to another size, making it ambiguous whether the volume was sampling at higher resolution or if the resulting window was simply resized via interpolation.

On the Reality Engine the scale (zoom) variable controls the size of the image. Based on the choice of zoom values, one would expect a factor of 1.65 ($0.90^2/0.70^2$) change between the two settings. The measured factor matches this fairly closely, generally being in the range of 1.3 to 1.6.

Rendering methods

On the Kubota, only MIP and IsoSurface methods were available to be tested. While these are simpler rendering algorithms, both opacity based and gradient based methods should be possible using planar resampling with the larger overall volume divided into smaller pieces and distributed among the FBMs. Kubota is currently working on an opacity renderer.

Our renderer for the Reality Engine performs planar texture volume resampling and supports Radiograph projection, MIP, opacity-compositing, and gradient-compositing methods. We choose to use the opacity-compositing method for benchmark measurements because it has been the most successful clinically, and because other methods like MIP and IsoSurface could be implemented at the same or better frame rates.

Sampling method

In general we have found that using trilinear interpolations for sampling the volume result in significantly improved image quality, and that point resampling is not acceptable. Because other renderers or hardware benchmarks may list only the faster, lower quality point sampling rates, we included point and trilinear sampling times on the Kubota to show the difference in frame rate between the two methods.

On the SGI, the sampling variable corresponded to the number of sample planes used in sampling the volume. We choose to test three values: 1.0, 0.5 and 0.25. Sampling every pixel guarantees that each pixel is having an effect on the resulting image. One would expect that if the original volume is sampled properly, then sampling at Nyquist rate of half steps (0.5 sampling rate) should give a fairly optimal reconstruction. We found the image quality to be satisfactory, but somewhat coarse with 1.0 sampling, always acceptable with 0.5, and slightly better with 0.25. In our experimentation with rates of less than 0.25 we did not often see an improvement; however, we have not evaluated large numbers of cases at sampling rates of less than 0.25 because of lack of interactivity. As expected, the rendering times were linearly proportional to the sampling resolution (i.e. using twice the number of sampling planes was twice as slow).

Rendering speed

Earlier, we established 5-10 frames a second as our goal. More specifically, 5 frames is marginal, while 10 frames a second or better yields the effective appearance of continuity as the object is manipulated. To compare the systems we choose the standard process we have found useful clinically: rendering a 256x256x32 dataset using opacity blending and trilinear sampling to a 512x512 window. This choice of window size, combined with choosing a zoom of 0.70 on the Reality Engine yielded the most similar resulting image size, as well as the most similar amount of sampling (image quality). On the Kubota we substituted MIP renderings for the opacity method. With these settings, the Kubota can achieve marginal realtime speeds (4.92 frames/sec) when

rendering. It also has the ability to render larger datasets (such as our example 512x512x64 study), but at less than realtime speeds (2.30 frames/sec). The Reality Engine with one raster board can achieve between marginal and good realtime rates (7.86 frames/sec), and with two raster boards it can achieve slightly better than good realtime rates (11.39 frames/sec).

4. Realtime Volume Rendering using Volume Texture Memory

We have previously published the algorithm for planar texture volume resampling, named *Voltex*, and described an implementation for rendering volume datasets in realtime on the SGI Reality Engine[15]. This section is an updated description of the algorithm from the original paper[15]. More recently, similar algorithms have been described by the technical staff of both Kubota[11] and SGI[13]; additionally, example volume rendering software is provided with those systems.

Viewing a volume from arbitrary positions requires reconstructing and resampling the volume along rays that extend from the view-point through the image plane pixels. If the resampled points are constrained to lie in planes, a textured polygon can be used to resample volume data that is loaded into the texture memory of the Reality Engine (figure 1). Polygons are embedded in the texture which is resampled at each point on the polygon projecting to a pixel. There are two alternatives for selecting the

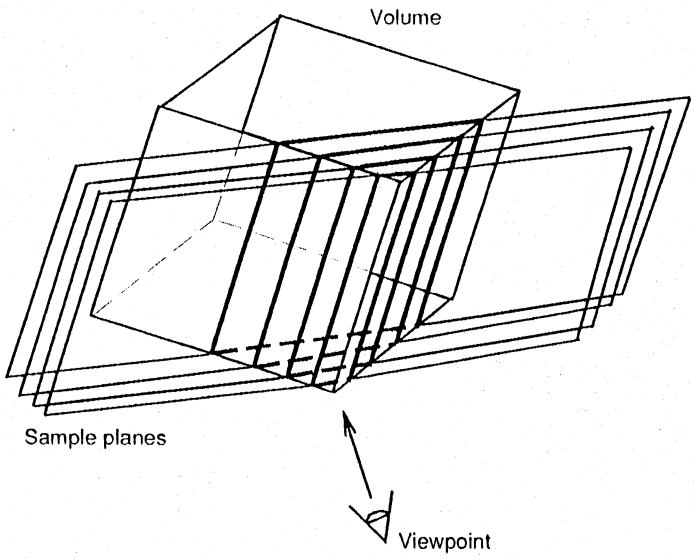

Figure 1. Planar volume sampling is shown, with eye point on bottom, cube depicting volume dataset that is stored in texture memory, and four sample planes showing planar samples. The dark lines show the actual polygons sampled from the texture volume. These are composited across all the planes sampling the volume.

orientations of the resampling polygons. Resampling can be done on polygons aligned with the object-space axes or on polygons aligned with the image-space axes. In either case, the resampled values behind each pixel are combined to produce a color for that pixel. The combining method is often a compositing operation, but it may be other operations as required by the visualization application.

Polygons aligned in object-space are defined to lie within the volume and rendered with GL library calls. This method is complicated slightly by the need to re-orient the sampling polygons in the plane most parallel to the view-plane as the view-point changes. This is accomplished by examining the view matrix and explicitly creating polygons for the six cases that arise[16]. Polygons aligned in image-space must be clipped to the boundaries of the volume to ensure valid texture coordinates. Polygons are defined in image-space and transformed by the inverse viewing matrix into object-space where the clipping occurs. Clipped polygons are then rendered with the usual GL library calls.

Radiographs

A digitally reconstructed radiograph of medical volume data is produced by combining the resampled values behind each pixel to approximate the attenuation integral

$$\text{pixel intensity} = 1.0 - \exp(-S \bullet u_i \bullet d)$$

where u_i are the resample values behind a pixel and d is the spacing between sample values. Note that d is constant for all samples behind a pixel, but due to perspective, it varies from pixel to pixel. The resampled u_i terms are summed at each pixel, and the d factors are applied by using an additional full-screen polygon with a 2D texture corresponding to the d values required for each pixel. The summation results may be viewed directly or the exponential required to mimic a radiograph may be computed at each pixel by using a lookup table.

The Reality Engine has a maximum precision of 12-bits per frame buffer and texture component. The summation could easily overflow that unless the sample values are properly scaled. Our implementation maintains 12-bit volume data values in the texture memory and scales each resampled value by a user controlled "exposure" value ranging from zero to one. The scaled samples are then summed and clamped if they exceed the 12-bit range. In practice, it has been easy to find suitable exposure control settings for the data sets tested.

Opacity-Based Rendering

The summation of samples produces radiograph images. Compositing samples produces images with occlusion. Only one texture component is required for the linear attenuation coefficient used to produce radiographs. Two 8-bit texture components can represent the raw data and a precomputed shading coefficient. The resampled data component values are used as indices into an opacity lookup table. This lookup uses the texture hardware for speed. The shading coefficient is a function of the original data gradient and multiplies the sample opacity to produce images of shaded features as shown in figure 2.

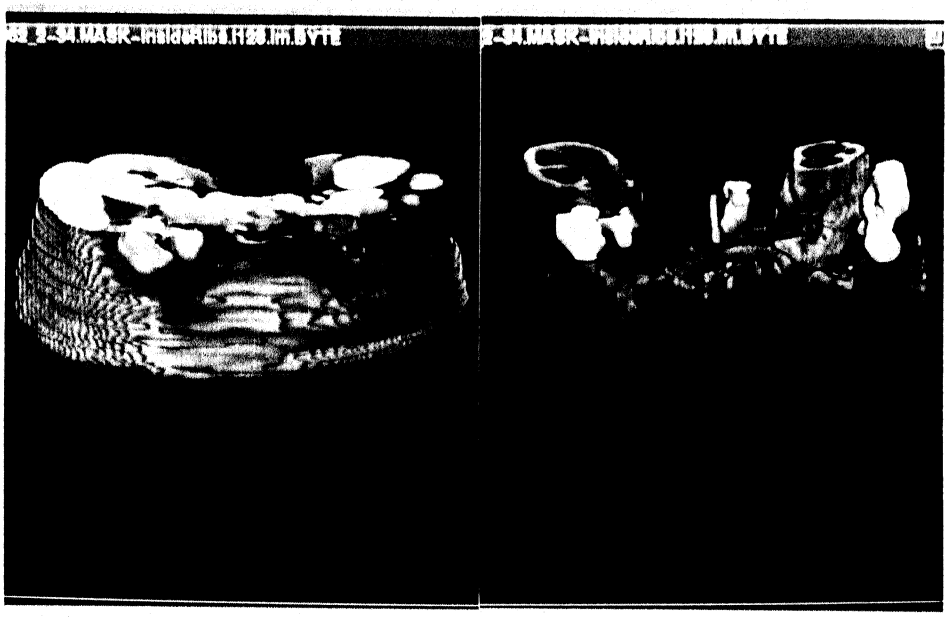

Figure 2. Two classifications of the same view of a spiral CT abdomen study demonstrate the power of interactive classification. The classification setting on the left image is set to visualize surfaces of organs on the interior of the body. The image on the right is the result of classifying to see interior vessels within the organs, especially the vascular structure in the liver (on bottom left). The volume has been rotated to an upside down position with the kidneys on top. The classifications were produced directly from the CT image data from the scanner in under 5 seconds.

The precomputed shading fixes the light position(s) relative to the volume. For more general lighting by sources fixed in image-space, the shade texture component must be replaced by three components containing the normalized data gradient. Unfortunately, the resampled gradient on the polygons is not normalized and normalization is a computationally expensive process requiring a square root. Lighting without normalization is possible, but this has not yet been tried to see how serious the artifacts are.

5. Clinical Experience using *SeeThru* Visualization Application

SeeThru@ is an application developed in the Department of Radiology at UNC for visualizing 3D medical image volume datasets. It is based on the Voltex rendering algorithm. Previous work has been done in visualizing surfaces that are easily definable, including bone/soft-tissue,[17,18] contrast in vessels in angiographic studies,[19,20] and skin for plastic surgery. SeeThru was developed to test whether we could effectively address the problem of visualizing difficult to define surface

Pixel Planes and the SGI led us to believe this was feasible if we could provide effective interactive control of three things: the parameters of the rendering, the parameters of viewing, and tools for cutting or cropping away parts of the study. After some experimentation in early 1993 using SeeThru on phantoms and clinical test cases, we made the tool available for clinical use in the fall of 1993. At this time a wide range of pilot cases has been done in the chest, abdomen, head and breast using both CT and MRI. Our most successful effort to date has been in thoracic surgical planning using spiral CT contrast studies and SeeThru to visualize the internal soft-tissue structures of the patient.

The most important result is that realtime high quality rendering makes a significant difference in the usable and acceptability of the visualization tool. Unless an application can be accessible quickly and conveniently from the original scan data, and can provide realtime control of rendering, viewing, and cutting, then it will not be clinically utilized. Two examples from our clinic demonstrate this. First, we have had research tools available for 10 years that can produce the same or higher quality pictures, but they were not interactive in rendering speed, nor were they quickly and conveniently available to the clinicians. These tools have never been used clinically. Second, our current CT machines support visualizations such as MIP, in both static and cine loop presentations directly on the scanner console. Except for experimental work, neither of these clinical tools are utilized by the radiologists or involved clinicians. This is for three reasons: the rendering is not in realtime, the quality of the rendering is not always good enough, and the visualization often requires classification and region extraction preprocessing steps that must be performed prior to being able to visualize the volume of interest. With SeeThru, on the other hand, our experience has been very positive. For example, in thoracic surgery planning after we performed an initial case as an example, the surgery department has requested a visualization for every complex surgery case performed since that time. The results of the thoracic surgery planning will be covered more completely in another paper.[21]

In our initial six months of clinical experience we have learned several important things about our rendering tool, and also about the visualization task. These are summarized below.

<u>Information learned about choice of rendering tool and parameters</u>

- Effective and easy to use computer human interface tools are required for manipulating the volume. The major operations are specifying rotations and cutting/clipping planes. For rotation, for example, user preference seems to be (first to last): using passive interface props[22] where actions on a 3D prop directly correspond to actions on the 3D screen object; using virtual sphere interface[23] where 2D mouse movements kinesthetically correspond to rotations of the 3D object on the screen; specifying motions via unbounded dial knobs; specifying motions via slider bars.

- Classification via opacity based compositing control eliminates need for segmentation in many of studies.

- Interactive segmentation tools will still be needed to visualize objects that cannot be isolated by classification and/or cutting; additionally, clearly segmented objects would improve the visualization in some types of studies.

- Opacity based rendering has provided the best visualizations to date. Especially important is the ability to interactively classify to see certain elements of the volume.

- Gradient based methods are not effective at finding, and visualizing surface for soft-tissue/soft-tissue interfaces because of lack of clear well defined boundaries.

- An important addition would be to have control over light source. Currently, the light source is fixed in relationship to the volume. This is to reduce storage and processing cost by not storing the normal at each voxel location (i.e. this would require two extra bytes to properly store 3D gradient, which would at least halve the current performance).

- Stereo may improve performance; however, other studies have shown that given the kinetic depth effects of interactive rotation, stereo does not significantly enhance the 3D perception[6]. Often, though, the clinician may want to concentrate on a single view, rather than have the object in motion; thus, this scenario may benefit from a stereo presentation. We have not tested using stereo at this time because it halves the frame rate, and reduces the resolution of the y axis.

- Realtime rendering of the volume makes exploration possible. This is often important in trying to find a visualization to answer a specific clinical question. Often, a 3D static visualization or cine loop failed to answer the clinical question, while the ability to interactively find the best, or sufficient visualization that allowed finding a satisfactorily answer to the clinical question. One initial drawback of this is that, unless users are provided excellent computer human interaction methods, they may spend tens of minutes searching out optimal visualizations, compared to the seconds to few minutes generally required by our expert users.

<u>Information learned in using realtime visualization clinically</u>

- Experienced radiologists do not benefit as much except for complex or uncommon anatomy, as they are well trained at reconstructing the expected anatomy in 3D from 2D cross sections.

- Other people, without significant experience at reconstructing 3D from 2D slices, seem to benefit significantly from seeing and interacting with the 3D visualization. Surgeons seemed to be especially comfortable using the 3D visualization tools, perhaps because it is closely tied to their clinical experience in manipulating the true 3D objects (i.e. the patient's body).

- In addition to using the 3D visualization for understanding the anatomy, the surgeons made significant use of the 3D representation as a model to communicate with other surgeons and medical staff in planning the operation.

- The typical scenario using SeeThru to assist in surgical planning was to classify the volume to see appropriate areas of interest; then to cut away and rotate the volume to get the best view; then cut back and forth through the 3D volume to see interior objects better, especially cutting in and out from an angle similar to the planned surgical entry.

- We found that being able to present a separate window with multiplanar reformatting information to be more preferred than to present this data on top of a volume slice. Presenting the 2D slice (intensity windowed version of density values) on the 3D volume (rendered to realistically simulate physical lighting model) takes away from the 3D perception of the object, and does not present the 2D slice at high resolution. Also, the addition of separate multiplanar 2D views essentially provide a combination 2D and 3D presentation environment combining views depicting both the 3d object and 2d density presentations. Since the 2D views support interactive access to all views from that angle, this combination presentation may serve as a replacement for seeing tiled presentations of the images on the lightbox or monitor, as they are commonly clinically presented at this time.

- The ability to conveniently bring up the study immediately, without preprocessing steps, and the ability to see and interact with the volume in 3D to visualize information not seen on 2D slices were the reasons clinicians gave for using SeeThru.

- Radiologists and clinicians generally preferred not to use successive refinement because of constant changing of the objects due to the refining made comprehending the 3D nature of complex objects even more difficult. Thus, for medical imaging, while successive refinement may useful in helping the user keep track of global position while manipulating the object, it was detrimental to comprehending the object in 3D, which was the main objective of the medical users.

6. Summary

This paper describes the process of directly rendering 3D medical image volumes in realtime. A description is given of two commercially available systems that support realtime volume rendering. A complete description of our planar texture volume resampling algorithm for accomplishing realtime rendering on such archictectures is provided. Benchmarks, and our analysis of them, are reported for the realistic medical image datasets on the two commercial systems. In our pilot clinical work using our SeeThru visualization package, we have found realtime volume rendering under user control to be an effective clinical tool, especially for surgical planning.

7. Acknowledgments

This work was supported in part by NIH R01 CA 44060, and NIH P01 47982.

8. References

1. Fuchs H, Poulton J, Eyles J, Greer T, Goldfeather J, Ellsworth D, Molnar S, Turk G, Tebbs B, Israel L, "Pixel-Planes 5: A Heterogenous Multiprocessor Graphics System Using Process Enhanced Memories," Computer Graphics (Proc. Siggraph 1992), Vol 26, No. 2, pp 2231-240, July 1992.

2. Taylor HH, Mezrich RS, Shahidi R, Knight S, "Real-Time Interactive Volumetric Rendering System for Medical Imaging", Radiology, Vol 189P, Nov 1993, (SS)131.

3. Fuchs H, Kedem ZM, Uselton SP, "Optimal Surface Reconstruction for Planar Contours", CACM 20, 1977.

4. Drebin RA, Carpenter L, Hanrahan P, "Volume Rendering", Computer Graphics, Vol 22, No 4, pp65-74, August 1988.

5. Yoo TS, Fuchs H, "Three Dimensional Visualization using Medical Data: 3D Medical Visualization from Acquisition to Application", Course 21, Siggraph 1993.

6. Sollenberg RL, Milgram P, "Effects of Stereoscopic and Rotational Displays in a Three-Dimensional Path Tracing Task", Human Factors, 35(3), pp483-499, 1993.

7. Todd JT, Akerstrom RA, Reichel FD, Hayes W, "Apparent rotation in three-dimensinal space: Effects of temporal, spatial, and structural factors", Perception & Psychophysics, 43, pp179-188, 1988.

8. Denali Technical Overview, Version 1.0, March 1993, Kubota Pacific Computer Inc.

9. Kubota Volume Extension, EFT Release, Part Number 340-0273-01, Kubota Pacific Computer Company.

10. Volume Rendering with Denali, Version 1.0, June 1993, Kubota Pacific Computer Company.

11. Guan, SY, Lipes R, "Innovative Volume Rendering Using 3D Texture Mapping", Medical Imaging 1994: Image Capture, Formatting, and Display, SPIE 2164, 1994.

12. Symmetric Multiprocessing Systems Technical Report, 1993, Silicon Graphics Inc.

13. Cabral B, Cam N, Foran F, "Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware", submitted to Volume Visualization, Baltimore, Oct 1994.

14. Molnar 92: Steven Molnar, John Eyles, John Poulton, "PixelFlow: High-Speed Rendering Using Image Composition." *Computer Graphics* 26(2):231-240, July 1992. Proceedings of SIGGRAPH'92.

15. Cullip TJ, Ulrich Neuman, "Accelerating Volume Reconstruction with 3D Texture Memory," Technical Report TR93-027, Department of Computer Science, UNC at Chapel Hill, June 1993.

16. Westover L, "Splatting - A Parallel, Feed-Forward Volume Rendering Algorithm." Dept. of Computer Science, UNC at Chapel Hill, Tech Report TR91-029, July 1991. Ph.D. Dissertation.

17. Fishman EK, Drebin RA, Magid D, et.al., ""Volumetric Rendering Techniques: Applications for 3-Dimensional Imaging of the Hip", Radiology, 163, pp737-738, 1987.

18. Ney DR, Fishman EK, Kawakshima A, Robertson DD, Scott WW, "Comparison of helical and serial CT with regard to three-dimensional imaging of musculoskeletal anatomy", Radiology, 185(3), pp865-869, Dec 1992.

19. Rubin GD, Dake MD, Napel SA, McDonnell CH, Jeffrey RB, "Three-Dimensional Spiral CT Angiography of the Abdomen: Initial Clinical Experience", Radiology, 186, pp147-152, 1993.

20. Schwartz RB, Jones KM, Chernoff DM, Mukherji SK, Khorasni R, Tice HM, Kikinis R, Hooton SM, Steig PE, Polak JF, "Common Cartoid Artery Bifurcation: Evalution with Spiral CT", Radiology, 185, pp513-519, 1992.

21. Molina P, Hemminger BM, Detterbeck F, "Realtime Three-Dimensional Visualization of Thorax: Initial Clinical Experience", in preparation for Radiology.

22. Hinckley K, Pausch R, Goble JC, Kassell NF, "A Three-Dimensional User Interface for Neurosurgical Visualization", Medical Imaging 1994: Image Capture, Formatting, and Display, SPIE 2164, 1994.

23. Chen M, Mountford SJ, Sellen A, "A Study in Interactive 3D Rotation Using 2D Control Devices", Computer Graphics, Vol 22, No 4, Siggraph, August 1988.

24. Kaba J, Matey J, Stoll G, Taylor H, Hanrahan P, "Interactive Terrain Rendering and Volume Visualization on the Princeton Engine', Proceedings of Visualization 1992, IEEE Computer Society Press, Oct 19-23, 1992.

25. Schroeder, Stoll G, "Data Parallel Volume Rendering Through Line Drawing", 1992 Workshop on Volume Visualization, IEEE Computer Society Press, Oct 1992.