Parallel Methods for Isosurfacc Visualization

Tushar Udeshi Steven Parker Charles Hansen Peter Shirley
Department of Computer Science
University of Utah
50 S. Campus Center Drive, MEB 3190
Salt Lake City, Utah
84112-9205
{tudeshi | sparker | hansen | shirley}@cs.utah.edu

Abstract

isosurface extraction and vis utilization is crucial for explorative scientific visualization of extremely large scientific data. The shear number of polygons extracted and the subsequent rendering time limit interactivity. We explore two solutions to this problem: exploiting parallel graphics hardware and parallel isosurface extraction/rendering via ray-tracing. We experimentally compare multi-pipe rendering that uses parallel binary-swap compositing/rendering and parallel isosurface extraction/rendering through ray-tracing,

1 Introduction

Many applications generate scalar fields *p(x,y,z)* which can be viewed by displaying *isosurjaces* where *p(x,y,z)* — *p[so*- Ideally, the value for p|so is interactively controlled by the user. When the scalar field is stored as a structured set of point samples, the most common technique for generating a given isosurface is to create an explicit polygonal representation for the surface using a technique such as *Marching Cubes* [9], This surface is subsequently rendered with attached graphics hardware accelerators such as the SGI Infinite Reality. Marching Cubes can generate an extraordinary number of polygons, which take time to construct and to render. For very large (i.e., greater than several million polygons) surfaces the isosurface extraction and rendering times limit the interactivity. In this paper, we address this problem by two distinct methods. One method addresses the rendering problem by using multiple graphics adapters in parallel[1]. The second method generates images of isosurfaces directly with no intermediate surfa.ee representation through the use of ray tracing.

2 Hardware Based Parallel Rendering

The use of parallelism in computer graphics hardware is widely known. Most current generation graphics adaptors utilize parallelism in their design and implementationfl, 2, 11], While these systems are extremely proficient at rendering geometry, the bottleneck for rendering large polygon sets is the speed at which polygons can be sent through the graphics pipeline[2]. Since there is a single thread which can send polygons to the graphics adaptor, the

[1]Thc so called *SGI 0nyx2 Reality Monster.*

"For 3D texture mapping based volume rendering, the bottleneck is typically pixel nil-rate. We limit onr application to explicit polygon rendering.

majority of rendering applications are serial and implicitly exploit the parallelism inherent in the graphics adaptor.

For scientific visualization of very large data sets, 512^3 and higher, parallel isosurface extraction and rendering techniques have been studied[3, 5, 6, 14]. These techniques exploit the large memory and parallelism of massively parallel computers to deal with the data explosion caused by scientific visualization of the large simulations running on the same machines. These techniques mimic, in software, the parallelism in the graphics hardware and achieve speedup although not at interactive rates. One of the problems with these approaches is the lack of an attached framebufier or graphics adaptor for displaying and interacting with the image.

Clearly what is desired is a combination of these approaches which takes advantage of the large memory and parallelism provided by large-scale parallel computers and the interactive rendering capabilities provided by graphics hardware. Fortunately, we have recently seen the convergence of these two with the SGI ONYX2 which is an SGI Origin 2000 with attached InfiniteReality graphics adaptors[12]. SGI ONYX2 with multiple InfiniteReality graphics adaptors are called Reality Monsters.[12] While these promise acceleration based on parallelism on both the macro scale (multiple graphic adaptors) and the micro scale (internal to each graphics adaptor), these systems are new and methods for exploiting them have not been studied. This section addresses an approach to exploiting the multiple graphics adaptors for polygon rendering.

2.1 Parallel Graphics Hardware Approach

The basic idea behind our algorithm is to divide the renderable data among the available graphics adapters, render each subset separately and *locally,* and combine the resulting partial images in an incremental fashion. This technique is strongly related to composition based volume renderers in that each graphics adapter renders a portion of the final image and these are combined[4]. It is similar to the composition network approach of Pixel Flow[1l] although Pixel Flow utilized custom designed hardware while our method exploits existing commercial hardware..

2.1.1 Binary Swap Assume *n* is the total number of polygons representing an isosurface and *g* is the number of graphics adapters. Typically *p* is a power of two although this is can be relaxed through simple extensions. We assume the isosurface, ri, exists in shared memory. Each graphics adapter loads and locally renders *njg* polygons. At this point, each graphics adapter has a partially complete image, these images with the corresponding z-bufi*ers are then composited onto the graphics adapter used for the final display. We use the *binary-swap*method for image composition which composites the image in *log2{g)* steps. At each step, the graphics pipes send the top half of their active image to their partner and receive the bottom half from their partner. The partners are determined as being 2* away where *i* is the compositing step. Thus when utilizing 8 graphics adapters, for the first composite the partners are [(0,1), (2,3), (4,5), (6,7)] and for the second composite the partners are [(0,2), (1,3), (4,6), (5,7)]. Composition of the incoming image with the current image is done in hardware making use of the stencil buffer. This has the effect of maximizing hardware usage and scales better with image size than performing the composite with the CPUs. The active image is reduced by half at each step. At the end of *log2{g)* levels, the active image on each graphics adapter is composited into the display graphics adapter. Figure 1 shows the compositing steps. The bottom row shows the results of rendering the initial polygon distribution on each graphics pipe. The gray areas in the images are back

-jj. $_0$£ graphics pipes	Time in Sees	speedup
1	9.23	1.0
2	6.33	1.45
4	4.06	2.27
8	2.93	3.15

TABLE 1

Scalability results for 10M polygon data set using a 1024x1024 Image

facing polygons. The three compositing levels are the next three rows up. The compositing partners are shown with lines between the compositing levels.

2.2 Results

We have tested our technique with a variety of large polygon data sets. In this section, we present the results. In all tests, we compare our technique with the best (non-compositing) single pipe version. The rendering code is all written in OpenGL.

For the first example, we wanted to stress the multipipe rendering system to understand where the tradeoiTs were for this technique. Recall, our goal is to render extremely large polygon data sets such as those one would see generated from 1GByte data sets. If we lower the number of polygons, we would expect the overhead from the multiple graphics adapters to limit the speed up. In fact, for sufficiently small polygon sets, we would expect the single graphics adapter to out-perform multiple graphics adapters. To test this, we take a 131,000 triangle isosurface generated from a CT scan. To test the scaling of polygon count, we instantiated multiple copies of this data, from 2 up to 8 copies. This provided a series of polygon data sets which varied from 131K to 1M polygons. We rendered these with 1 to 8 graphics adapters. To mitigate instantaneous timing anomalies, we rendered each frame 100 times. The results are shown in Figure 2 and Figure 3 for 512 x 512 and 1024 x 1024 images respectively. The X-axis is the size, in triangles, of the isosurface being rendered. The Y-axis is the time in seconds for rendering 100 images. As is shown in the plots, for the 1024x1024 image the single graphics pipe out performs the multiple pipes for the isosurface containing 131,000 triangles. This is to be expected since the overhead of reading back the frame buffers and compositing in the multiple graphics adapters case adds overhead and the rendering speed of the Infinite Reality graphics adapters can easily handle that modest number of polygons. However, once the polygon count increases to 262K triangles, the multiple graphics adapters out perform the single graphics pipe for all cases. The improvement for multiple graphics pipes steadily increases as the polygon count increases. One can notice the difference between the 512x512 and 1024x1024 images. In the 512x512 case, the multiple graphics adapters are always faster than a single graphics adapter, even on this small number of polygons! The overhead of reading back and re-writing to the graphics adapters (the compositing step) for the larger images in the 1024x1024 case results in slower rendering times for low numbers of polygons. The overhead predominates for the 131K triangle case. However, as the polygon count increases to 1M triangles, the overhead becomes less of the overall time and the cost for rendering a 1024x1024 image reduces to near the cost of a 512x512 image.

Table 1 shows the scaling of this technique using one through eight graphics adaptors.

For the next example, we extracted the isosurface representing the skin for the head portion of the visible woman data set (see Figure 1). The isosurface is composed of 1.4

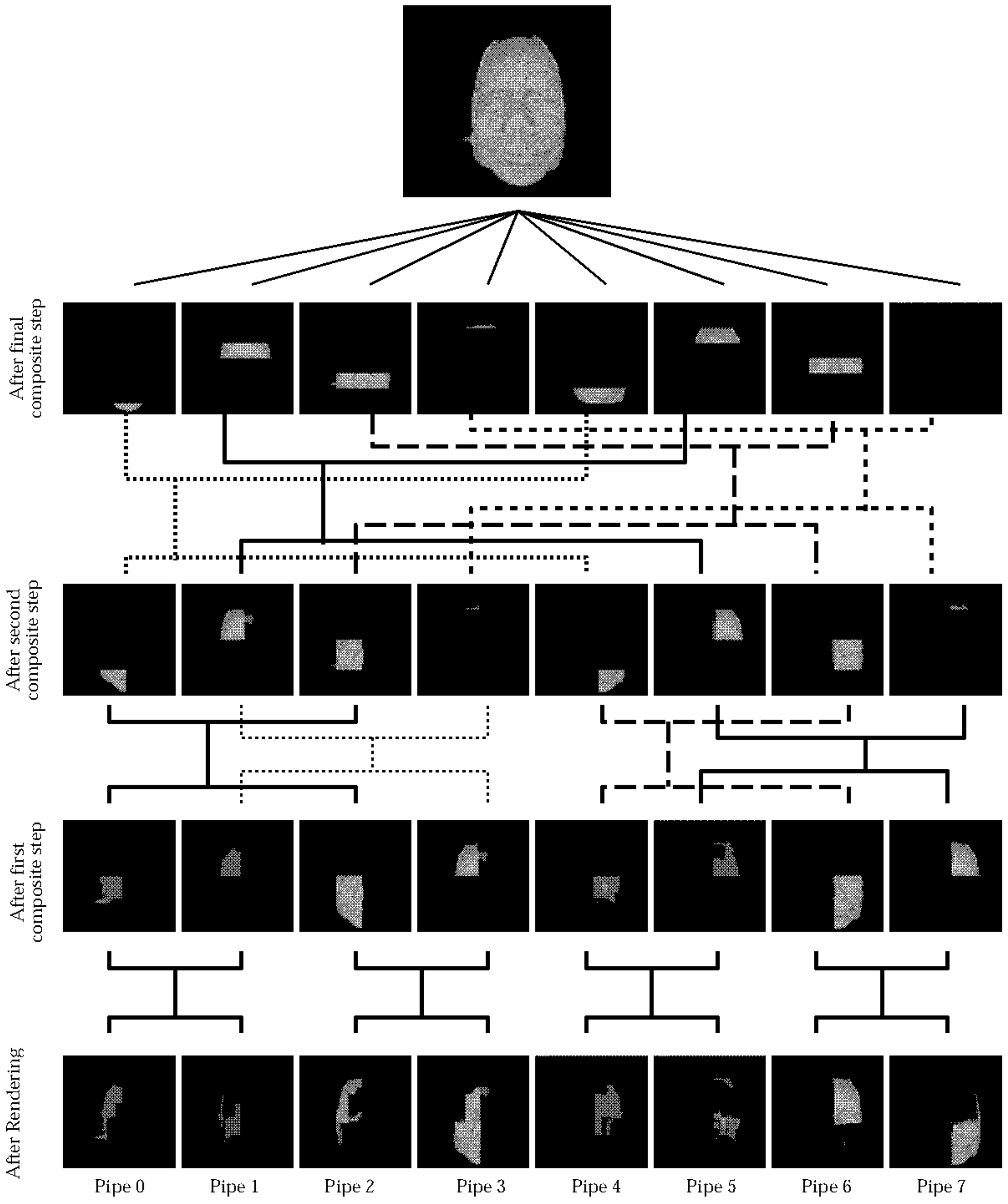
After final composite step
After second composite step
After first composite step
After Rendering
Pipe 0
Pipe 1
Pipe 2
Pipe 3
Pipe 4
Pipe 5
Pipe 6
Pipe 7

FIG. 1. *The compositing levels are along the vertical axis and the graphics adapters are along the horizontal axis. After the final step (the top most level), the final image is composed from each of the partial images.*

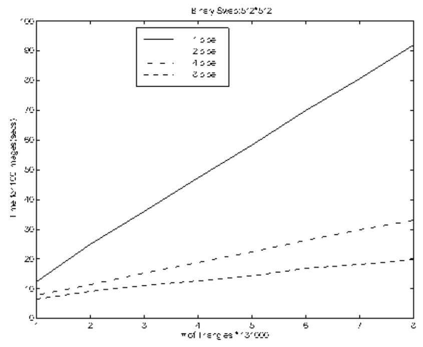

FIG. 2. *Times for rendering a 512 x 512 image*

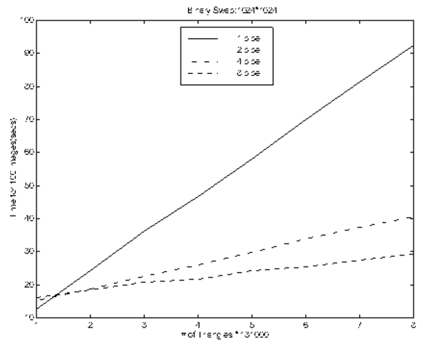

FIG. 3. *Times for rendering a 1024 x 1024 ifff^ge*

million triangles. We again instantiated multiple copies of this data with 2, 3, and 4 copies resulting in 2.8M, 4.2M, and 5.6M polygon isosurfa.ee data sets. Figure 4 and Figure 5 show the results for both 512x512 and 1024x1024 images. Notice that the rendering times are very close with the 512x512 image being slightly faster for rendering with larger numbers of graphics pipes. This is due to the increased number of steps in the compositing operations with an increased number of graphics pipes. However, with even a modest 1.4M polygons, 8 graphics adapters outperforms a single graphics adapter by a factor of 2. For 5.6M polygon data set, 8 graphics pipes outperforms a single graphics pipe by 6 times.

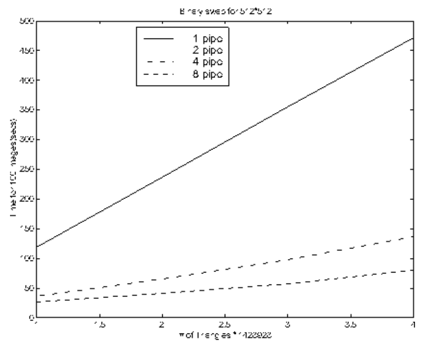

FIG. 4. *Times for rendering a 512 x 512 image for 1J, to 5.0 million polygons*

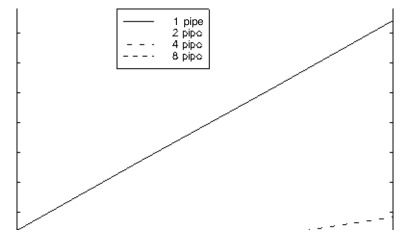

FIG. 5. *Times for rendering a 1024 ^{x} 1024 Wfrnge for 1J$_{(}$ to 5.0 million polygons*

3 Ray-tracing Isosurface Extraction

An alternative to extracting geometry and employing parallel graphics hardware to interactively render very large isosurfaces is to leverage the parallel CPUs to generate an image *without* extracting intermediate geometry. Our next algorithm generates images of isosurfaces directly with no intermediate surface representation through the use of ray tracing. Ray tracing for isosurfaces has been used in the past (e.g. [8, 10, 16]), but we apply it to very large datasets in an interactive setting. Details necessary to implement this technique can be found in [15].

[15]

The basic ray-isosurface intersection method used in this technique is shown in Figure 6. Conventional wisdom holds that ray tracing is too slow to be competitive with hardware z-bufTers. However, when rendering a surface from a sufficiently large dataset, ray tracing should become competitive as its low time complexity overcomes its large time constant [7]. The same arguments apply to the isosurfacing problem. Suppose we have an n X n X n rectilinear volume which for a given isosurface value has $O(n^2)$ polygons generated using Marching Cubes. Given intelligent preprocessing, the rendering time on g graphics adaptors will be $O(n^2/g)$. If a ray tracing algorithm is used to traverse the volume until a surface is reached, we would expect each ray to do $O(n)$ work. If the rays are traced on p processors, then we expect the runtime for an isosurface image to be 0(ri/p), albeit with a very large time constant and a limit that p is significantly lower than the number of pixels. For sufficiently large ri, ray tracing will be faster than a z-bufi*er algorithm for generating and rendering isosurfaces. The question is whether it can occur on an n that occurs in practice (e.g., n — 500 to n — 1000) with a p that exists on a real machine (e.g., p — 8 to p — 128). This paper demonstrates that with a few optimizations, ray tracing is *already* attractive for at least some isosurface applications.

3.1 The Algorithm

Our algorithm has three phases: traversing a ray through cells which do not contain an isosurface, analytically computing the isosurface when intersecting a voxel containing the isosurface, shading the resulting intersection point. This process is repeated for each pixel on the screen. Since each ray is independent, parallelization is straightforward. An additional benefit is that adding incremental features to the rendering has only incremental cost. For example, if one is visualizing multiple isosurfaces with some of them rendered transparently, the correct compositing order is guaranteed since we traverse the volume in a front-to-back order along the rays. Additional shading techniques, such as shadows and specular reflection, can easily be incorporated for enhanced visual cues. Another benefit is r

FIG. 0. A ray Is Intersected directly with the Isosurface. No explicit surface Is computet"

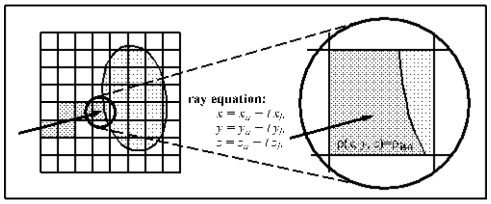

FIG. 7. The ray traverses each cell (left), and when a cell is encountered that has an Isosurface in It (light), an analytic ray-Isosurface intersection computation Is performed,

the ability to exploit texture maps which are much larger than texture memory.

3.2 Ray-Isosurface Intersection

If we assume a regular volume with even grid point spacing arranged in a rectilinear array, then the ray-isosurface intersection is straightforward. Analogous simple schemes exist for intersection of tetrahedral cells, but the traversal of such grids is left for future work. This work will focus on rectilinear data.

To find an intersection (Figure 7), the ray *a-j-th* traverses cells in the volume checking each cell to see if its data range bounds an isovalue. If it does, an analytic computation is performed to solve for the ray parameter *I* at the intersection with the isosurface:

$$p\{x_u + tXbjVa + LVb,Za + tZb) - p[_{so} = 0.$$

When approximating *p* with a trilinear interpolation between discrete grid points, this equation will expand to a cubic polynomial in *I*. This cubic can then be solved in closed form to find the intersections of the ray with the isosurface in that cell. Only the roots of the polynomial which are contained in the cell are examined. There may be multiple roots, corresponding to multiple intersection points. In this case, the smallest *I* (closest to the eye) is used. There may also be no roots of the polynomial, in which case the ray misses the isosurface in the cell. The details of this intersection computation and the associated optimizations needed for an interactive system can be found in [15].

3.3 Results

We applied the ray tracing isosurface extraction to interactively visualize the Visible Woman dataset. The Visible Woman dataset is available through the National Library of Medicine as part of its Visible Human Project [13]. We used the computed tomography (OT) data which was acquired in 1mm slices with varying in-slice resolution. This data is composed

FIG. 8. Ray tracings of the skin and bone Isosurfaces of the Visible Woman (see color page).

Isosurface	Traversal	Intersec.	Shading	FPS
Skin (p - 600.5)	55%	22%	23%	7-15
Bone {p= 1224.5)	66%	21%	13%	6-15

TABLE 2

Data From Ray Tracing the Visible Woman. The frames-per-second (FPS) gives the observec range for the Interactively generated viewpoints on 64 CPUs.

of 1734 slices of 512x512 images at 16 bits. The complete dataset is 910MBytes. Rather than down-sample the data with a loss of resolution, we utilize the full resolution data in our experiments. As previously described, our algorithm has three phases: traversing a ray through cells which do not contain an isosurface, analytically computing the isosurface when intersecting a voxel containing the isosurface, and shading the resulting intersection point.

Figure 8 shows a ray tracing for two isosurface values. Figure 9 illustrates how shadows can improve our the accuracy of our geometric perception. Table 2 shows the percentages of time spent in each of these phases, as obtained through the cycle hardware counter in SGFs speedshop. As can be seen, we achieve about 10 frames per second (FPS) interactive rates while rendering the full, nearly 1GByte, dataset.

Table 3 shows the scalability of the algorithm from 1 to 128 processors. View 2 is simpler than view 1, and thus achieves higher frame rates. Of course, maximum interaction is obtained with 128 processors, but reasonable interaction can be achieved with fewer processors. If a smaller number of processors were available, one could reduce the image size in order to restore the interactive rates. Efficiencies are 91% and 80% for view 1 and 2 respectively on 128 processors. The reduced efficiency with larger numbers of processors (> 64) can be explained by load imbalances and the time required to synchronize processors

FIG. 9. A ray tracing with and without shadows (see color page).

at the required frame rate. These efficiencies would be higher for a larger image.

4 Conclusions

We have presented two parallel methods for the visualization of isosurfaces which utilize different mechanisms for acceleration. Both are effective although for different applications. The interactiveness of the ray-tracing technique provides visualization of an isosurface at a higher frame rate than is possible with the multipipe hardware solution. This is due to the geometry extraction and surface construction time needed for the multipipe hardware solution.

However, it is often very useful to perform quantitative analysis on the isosurface. For example, one frequently would like to know the volume bounded by an isosurface of a particular value. Similarly, with the explicit surface, it is possible to examine quantitatively the difference between surfaces. With the ray-tracing solution, one only has the final image and such evaluations are limited. On the other hand, spatial relationships are better understood through interacting with the isosurface and the ray-tracing solution is well suited for this style of data exploration.

References

[1] Kurt Akeley. **Reality** Engine **graphics.** 27:109 116, August 1993.

[2] Kurt Akeley and Tom Jermoluk, High-performance polygon rendering. *Computer Graphics,* 22(4):239 246, August 1988. ACM Siggrapli '88 Conference Proceedings.

[3] David A. Ellsworth, A new algorithm for interactive graphics on multicomptiters, *IEEE Computer Graphics and Applications,* 14(4), July 1994,

	View 1		View 2	
# of CPUs	FPS	speedup	FPS	speedup
1	0.18	1.0	0.39	1.0
2	0.36	2.0	0.79	2.0
4	0.72	4.0	1.58	4.1
8	1.44	8.0	3.16	8.1
12	2.17	12.1	4.73	12.1
16	2.89	16.1	6.31	16.2
24	4.33	24.1	9.47	24.3
32	5.55	30.8	11.34	29.1
48	8.50	47.2	16.96	43.5
64	10.40	57.8	22.14	56.8
96	16.10	89.4	33.34	85.5
128	20.49	113.8	39.98	102.5

TABLE 3

Scalability results for ray tracing the bone Isosurface In the visible human. A 512x512 Image was generated using a single view of the bone Isosurface,

[4] Kwan-Lul Ma et al. Parallel volume renderer using binary-swap Image composition, *IEEE Computer Graphics and Applications,* 14(4), July 1994,

[5] C, Hansen and P, Hhiker, Massively parallel Isosurface extraction. In *Proceedings of Visualization "92,* pages 77 83, October 1992,

[G] C, Hansen, M, Krogh, and W, White, Massively parallel visualization: Parallel rendering. In *Proceedings of SI AM Parallel Computation Conference,* February 1995,

[7] James T, Kajiya, An overview and comparison of rendering methods, *A Consumer's and Developer's Guide to Image Synthesis,* pages 259 2G3, 1988, ACM SIggraph '88 Course 12 Notes,

[8] Chyi-Cheng Lin and Yu-Tai Clung, An efficient volume-rendering algorithm with an analytic approach. *The Visual Computer,* 12(10):515 52G, 199G.

[9] William E, Lorensen and Harvey E, Cllne. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics,* 21(4):1G3 1G9, July 1987, ACM SIggraph '87 Conference Proceedings,

10] Stephen Marschner and Richard Lobb, An evaluation of reconstruction filters for volume rendering, hi *Proceedings of Visualization "94-* pages 100 107, October 1994,

11] Steven Mohiar, John Eyles, and John Poult on, Pixelflow: High-speed rendering using Image composition. *Computer Graphics,* 2G(2):231 240, July 1992, ACM SIggraph '92 Conference Proceedings,

12] John Montrym, Daniel Baum, David Dlgnam, and Christopher Migdal, Infinltereahty: A real-time graphics system, 2G:293 302, August 1997,

13] National Library of Medicine (U.S.) Board of Regents, Electronic Imaging: Report of the board of regents, u,s, department of health and human services, public health service, national **institutes** of health. NIH **Publication** 90-2197, 1990.

14] F, Ortega, C, Hansen, and J, Ahrens, Fast data parallel polygon rendering. In *Proceedings of Supercomputing '93,* pages 709 718, November 1993,

15] Steven Parker, Peter Shirley, Yarden Llvnat, Charles Hansen, and Peter-Pike Sloan, Interactive ray tracing for Isosurface rendering. In *Proceedings Visualization '98,* 1998,

1G] Milos Sramek, Fast surface rendering from raster data by voxel traversal using chessboard distance. In *Proceedings of Visualization "94,* pages 188 195, October 1994.