Interactive Surface Rendering for Medical Visualization

Fang Chee Hung Anthony*
Kelvin Sung*
Heng Pheng Ann†

Abstract

A data structure and a visibility preprocessing technique are introduced to assist in the manipulation and to accelerate the rendering of the primitives generated by the Marching Cubes Algorithm. A view oriented traversal algorithm of the data structure is described. The traversal scheme allows the ordered display of semi-transparent surfaces. The rendering time of the isosurfaces is reduced by approximating the visibility of the intersected volume cells from a predefined set of viewpoints. Experiments showed that rendering speedup of 2 to 5 times are achieved on MRI and CT datasets.

Keywords: Computer Graphics; Medical Imaging; Scientific Visualization; Geometric Modeling; Surface Rendering

Subject Descriptors: I.3.7 Three Dimensional Graphics and Realism
I.4.10 Image Representation (Volumetric)

1 Introduction

The field of Scientific Visualization is broad and encompasses different applications and techniques. One of the fast-growing areas in Scientific Visualization is Volume Visualization. Volume Visualization can be broadly described as the visual interpretation of scalar or vector datasets defined on multidimensional grids for the purpose of gaining insight into a scientific problem. Currently, one major application area in this field is medical imaging, where volume data is generated from the X-ray Computer Tomography (CT) scanners, the Positron Emission Tomography (PET) scanners, and the Magnetic Resonance Imaging (MRI) devices.

Computerized methods of visualizing volume data can be classified into direct and indirect approaches [9]. The direct approaches, also known as volume rendering, refer to the methods that create images from the three-dimensional volume without generating intermediate geometric representations. These are usually done by casting rays from the view point onto each pixel and extrapolating the rays into the volume space (e.g. [4, 13, 26, 8, 14]). On the other hand, the indirect approaches construct the required isosurfaces,

*Department of Information Systems and Computer Science, National University of Singapore, Email: {fangchee|ksung}@iscs.nus.sg

†Institute of Systems Science, National University of Singapore, Email: pheng@iss.nus.sg

expressed in the form of an intermediate geometric representation, and generate images from the resulting geometric data. These include contour rendering (e.g. [18, 20]) and surface rendering (e.g. [29, 15, 7, 22]) methods.

As discussed in [9], the major characteristics of the volume rendering approaches are the high quality resulting images and the relatively slower rendering time. This is in contrast to the major advantage of the surface rendering techniques where once the surfaces have been constructed, a high performance graphics workstation can be used to interactively control the visualization process. However, surface rendering algorithms often generate too many polygons for interactive manipulations. For example, typically, the Marching Cubes Algorithm [15] generates hundreds of thousands of triangles for isosurfaces in medical datasets. This paper presents our work done in designing and implementing software techniques to accelerate the rendering process while providing both interactive control and high quality images of surface rendering for 3D medical data. More specifically, we describe:

1. A comprehensive set of data structures for efficient storage, traversal, and manipulation of isosurface primitives generated by the Marching Cubes Algorithm.

2. A simple but effective scheme for rendering multiple semi-transparent surfaces.

3. A visibility preprocessing algorithm that reduces the rendering time of opaque surfaces with minimum sacrifice in image quality.

Although the discussions are based on the medical volume dataset, the algorithms introduced in this paper can be applied to any scalar and rectilinear volume datasets in general.

This paper is organized as follows. The next section discusses some of the issues involved in implementing the Marching Cubes Algorithm (MCA) [15]. Section 3 introduces a data structure to store the primitives generated by the MCA. An efficient view oriented traversal of the data structure is then described. The traversal scheme enables the rendering of single or multiple semi-transparent isosurfaces. Section 4 begins by discussing the previous approaches in visibility preprocessing and follows by describing our preprocessing technique. The results of applying our preprocessing technique are presented at the end of Section 4.

2 The Marching Cubes Algorithm (MCA)

One of the best volume data surface extraction algorithms is the Marching Cubes Algorithm (MCA). This algorithm was independently reported by Wyvill, Mcpheeters, and Wyvill in 1986 [29]; and by Lorensen and Cline in 1987 [15]. The main difference in the two approaches was a result of their respective application areas. Wyvill et. al. developed the algorithm to extract an isosurface from an analytical field defined by a set of key points. The algorithm was designed to model and animate soft objects. Lorensen and Cline designed their algorithm to process three-dimensional medical data.

The MCA can be considered as a divide-and-conquer approach that operates in two phases. In the first phase, a constant surface value (hence the term isosurface) is defined by the user and all the cubes (or voxels) that are intersected by the surface are found. We shall call this set of intersected cubes the boundary set. In the second phase, each cube in the boundary set is processed to produce a set of connected polygons. Polygon vertices

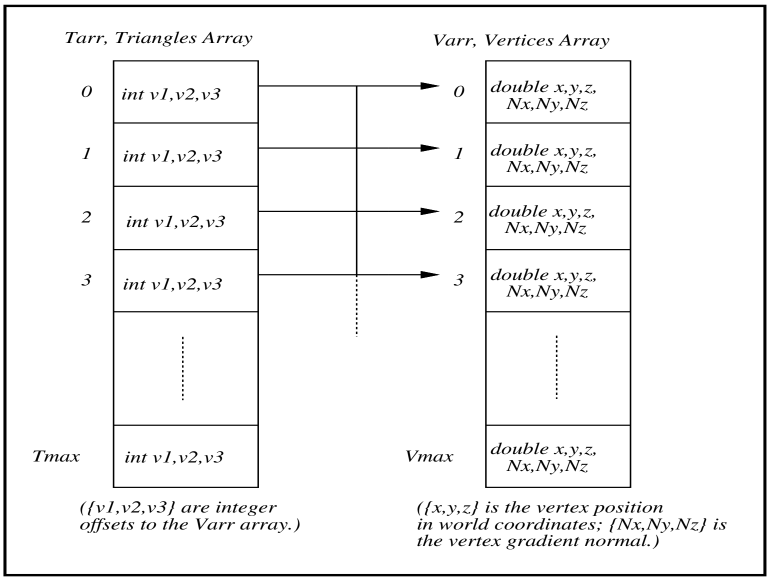

Figure 1: A Simple Vertex Scheme.

are computed through linear interpolation along cube edges. Although it is common to describe the MCA as a two-phase process, in practice, it is more efficient to implement the two phases successively for each cube in consideration. That is, to immediately triangulate the isosurface within a cube once it is determined to be a boundary cube. The advantage in performing the two phases one after the other is that the size of the triangle and vertex array can be determined exactly once the boundary set is obtained.

An edge lookup table is usually used to assist the triangulation process [15]. Each boundary cube is indexed into the table to obtain (1) the number of triangles within the cube and (2) the cube edges involved in each triangle. We have adopted Wyvill and Jevans's lookup table generation technique [28], where the triangulations are performed in a consistent clockwise manner. This consistent vertex orientation is crucial for dealing with backfacing polygons, which can usually be omitted when rendering topologically closed and opaque isosurfaces[1].

3 Data Structures

Conventionally, marching cubes produces a long chain of triangles. This chain may be easily stored as indexed polygons. The idea is illustrated in Figure 1.

Varr is a pool of floating point coordinates of vertices and their normals for interpolative shading purposes. Tarr is a simple chain of triangles produced during surface extraction. Each triangle has three vertices which are stored as integer offsets into Varr.

This simple scheme of storing the triangles has its advantages. Its simplicity allows rapid traversal of the geometric database. This is especially important when working with high-end graphics hardware architecture with efficient rendering pipelines. The storage

[1]However the back faces have to be properly illuminated when (1) a surface is semi-transparent, (2) part of the surface has been intentionally removed or shifted, or (3) a surface is partially clipped. In such cases, we can visually differentiate between back and front faces by defining different material properties for them.

scheme is also efficient because there is no redundancy in the vertices pool.

However, in many cases, this simple structure is inadequate. One example is when depth-wise traversal of the database is required (e.g. in rendering semi-transparent surfaces). In such cases, some intensive preprocessing and sorting must usually be performed. For example, in [17] the A-buffer algorithm [5] was adopted to simulate semi-transparency. The computationally intensive pixel level clipping and depth sorting are the major drawbacks of this approach. Furthermore, a faithful implementation of the subpixel-level sampling required by the A-buffer algorithm is difficult to integrate into an existing rendering hardware (e.g. the SGI architecture).

One approach to avoid depth sorting is to recognize that the 3d grid space[2] presents a natural ordering of the triangles and to take advantage of this structural organization. With the 3d grid space, depth ordering can be easily determined from any given view point.

3.1 Extended Vertex Scheme

Figure 2 depicts a data structure for the storage of the geometric database. This structure embeds the information about the grid-wise structural organization of the volume space. It is essentially a 2-level structure where one level stores the spatial distribution of the boundary cubes set and the other stores the geometric description of the isosurface. At the higher-level, a cell-based spatial hierarchy representing the boundary set is constructed over the indexed polygon structure described in Figure 1. We retain the indexed polygon structure at the lower level to maintain the support for fast sequential traversal of the full set of geometric primitives.

The extended structure is constructed during the MCA process. The size of the slice array, Sarr, is equal to the number of slices of the volume data. Each element of Sarr has a pointer to an 1D array (Rarr) of rows that are intersected by the isosurface. Each Rarr element has a row# to identify the row number within the slice, as well as an integer to store the number of cells in the row that are intersected. Each Rarr element also contains a pointer to an 1D array (Carr) of cells which the isosurface intersects. Each Carr element contains a cell# identifier, the number of triangles generated in the cell, and an array of triangles, Tarr. The contents of Tarr and Varr is the same as in the simple vertex scheme where Varr still is a common pool of vertices that are shared among all the triangles. The arrays in the data structure (except Sarr) are dynamically allocated during run-time, after their sizes have been determined.

The extended vertex scheme supports the storage of multiple isosurfaces where each surface is represented independently at a lower level by an indexed polygon structure. A single higher level spatial hierarchy encapsulates and references to all the lower level geometric structures that describes each surface (see Figure 3). To distinguish the different surfaces at the higher level, each surface is assigned an unique identifier which is used in each element of Carr to index into the appropriate surface structure.

The disadvantage of this structure is that the addition levels of indirection slows down the traversal process. However, from the following explanations, it would be clear that the flexibility derived outweighs the shortcomings.

[2]Where the triangles were created and interpolated from.

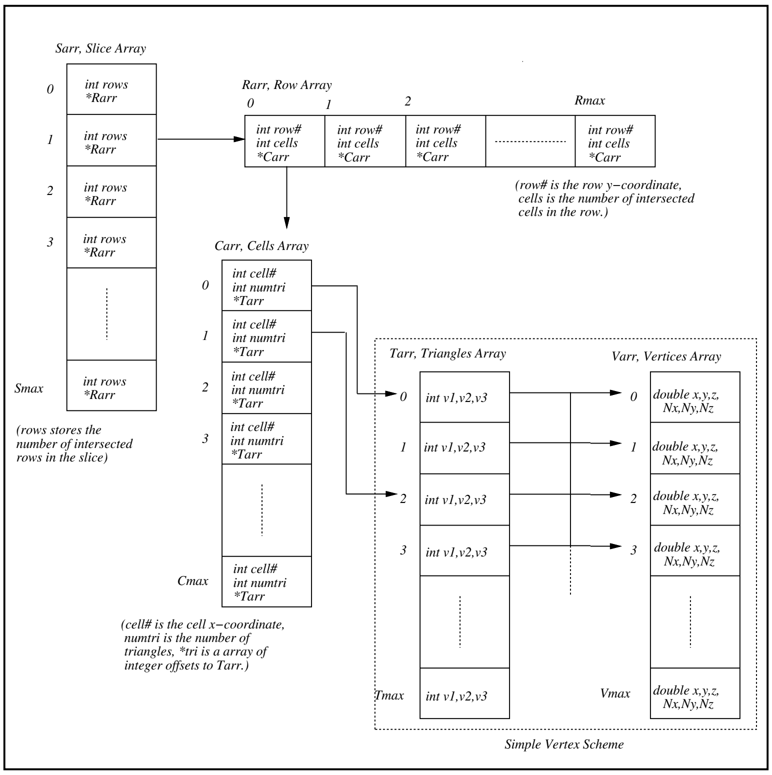

Figure 2: Extended Vertex Scheme.

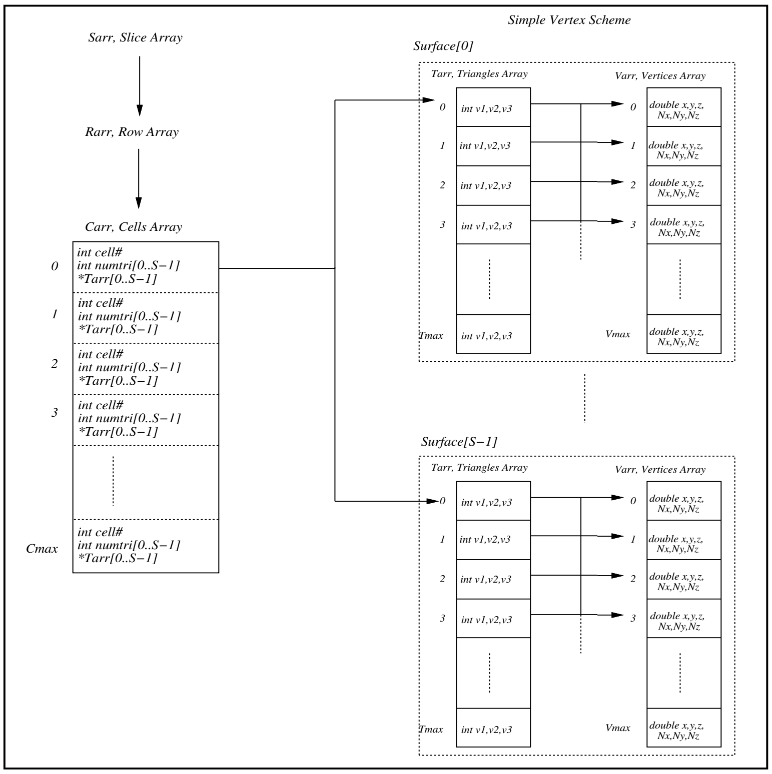

Figure 3: Extended Vertex Scheme to support multiple isosurfaces.

3.2 Depthwise Structure Traversal

One major advantage of maintaining this more complex data structure lies in its ability to support directional traversal. When using the simpler scheme (Figure 1), the geometric data can only be traversed either forward or backward. If the surface extraction scanned the volume space from the lowest cell coordinate $(0, 0, 0)$ to the largest cell coordinate (I, J, K), then traversing Tarr forward will be equivalent to traversing the volume data from the cell coordinate $(0, 0, 0)$ to the cell coordinate (I, J, K). Traversing Tarr backward would be equivalent to traversing the volume data from (I, J, K) to $(0, 0, 0)$. Here, the traversal is always along the x direction first, followed by the y and z directions.

The extended structure supports a traversal starting from any of the 8 end points of the volume data. By choosing a viewpoint nearest/farthest from a given viewpoint to begin the traversal, the method is equivalent to a front-to-back/back-to-front display of the geometric primitives contained in each cell.

Our method of structure traversal is similar to the method of volume traverse described in [2]. The difference is that we are traversing a geometric database as oppose to scanning a volume space. To traverse the structure with increasing depth, we first determine the volume end point that is nearest to the current eye position (for decreasing depth, the farthest end point is determined). This can be easily obtained by examining the line of sight vector with respect to the center of the volume space.

More precisely, if point E is the eye point, C is the volume center and the line of sight vector $V = E - C$, then with reference to Figure 2, we will traverse the extended vertex structure in the following manner:

if V.z > 0 then traverse Sarr from Smax to 0, else from 0 to Smax
if V.y > 0 then traverse Rarr from Rmax to 0, else from 0 to Rmax
if V.x > 0 then traverse Carr from Cmax to 0, else from 0 to Cmax

This operation is only required to be performed once for each view. Beginning a traversal from the nearest end point corresponds to a front-to-back traversal, and traversal from the farthest end point corresponds to a back-to-front traversal. In the case when there are two near endpoints at an equal distance from the view point (e.g. when viewing along an orthogonal axis of the volume space), one is chosen arbitrarily.

A depthwise display of primitives has many advantages. Firstly, a front-to-back display method will definitely result in less pixels and z-buffer updates. However, this advantage may not obvious when specialized graphics pipelined architecture such as the SGI is used. A front-to-back traversal would certainly improve the rendering time if no specialized graphics hardware is used. Secondly, a back-to-front depth-wise traversal allows the user to observe the image as it is being generated. This enables the user to examine structures that are not visible in the final image. A third and more important advantage is in the displaying of semi-transparent isosurfaces.

3.3 Semi-Transparency

The transparent attribute of surfaces is commonly referred to as its alpha component [5]. In general, to correctly render non-opaque surfaces, the primitives must be presented in a back-to-front order (e.g. the SGI architecture [23, pp. 15-17]). Intensities and alpha accumulations are performed through an appropriate blending function. With our data structures and the depthwise traversal scheme, transparency rendering is supported naturally. We demonstrate the capabilities of our method to render single and multiple isosurfaces extracted from a CT and an MRI volume dataset (Plates A and B). Notice that the back and internal structures are clearly visible.

One other simple way of simulating semi-transparency of multiple surfaces is to render each surface individually onto different images and then combine the images pixel-by-pixel based on their corresponding depth and alpha values (e.g. [19, 16]). However this would result in incorrect images if a line of sight intersects multiple points on the same surface. Examples of the incorrect results are shown in Plates C and D. Although the images appeared to be less "cluttered", these images are incorrect because they do not faithfully represent the material properties of the geometric surfaces [3].

It is important to note that the traversal scheme only ensures a back-to-front ordered display of spatial boundary cells. If a cell contains more than 1 disjoint patches, the order in which these surface patches are presented may not necessarily be in a back-to-front order. In our implementation, if a cube contains more than one patch, we will simply render them in the order in which the triangles are stored. This defect may result in erroneous images. However, the error is not significant due to the small size of the triangles and that the number of disjoint patches within a cube is usually one. From our experience, this has not caused any visually significant erroneous output.

4 Visibility Preprocessing

"Brute-force" surface construction algorithms such as the MCA produces a large number of geometric primitives. When rendering as opaque surfaces, many of these primitives lies within the outer surfaces and hence will not contribute to the final image. From experience, it is found that, on the average, only about half of the outer surfaces are visible with any given viewing position. Our preprocessing approach eliminates the need to render most of these hidden polygons. A simple backface polygon culling will not remove the internal front facing polygons that are hidden from the view point. Moreover, in cases where the surface topology is intentionally made open (e.g. in cutting operations), backface polygon culling techniques will result in holes in the images.

4.1 Background

The idea of visibility preprocessing to reduce rendering time is not new. Recently, Airey et. al [1], and Teller and Sequin [24] discussed approaches to building visibility structures to assist the pruning of the non-visible portions of a geometric database to support an interactive building walk through. Although we are also reducing the number of primitives to be rendered for any given viewing position, our problem is simpler because the degree of freedom associated with our viewing position is more restricted. In order to support a realistic interactive building walk through, a system must allow the viewing position to be translated and rotated freely within the geometric database. However, in volume visualization, the database is typically examined from outside of the volume space and thus only the rotation of the viewing position is of primary importance. Both of the above building walk through systems assumed a human designed structure, where a building is divided into *floors* by *ceilings*, and there are *walls* dividing *rooms* on each floor, etc.

[3] Such views can be useful if, for instance, we wish to visualize an internal structure in relation to the *outermost* surface of an external structure (e.g. a brain within an outer skin surface in Plate D).

These visibility occluding dividers (i.e. the *ceilings* and *walls*) are conveniently chosen to partition the database into visibly-disjoint units. In this way, a large portion of the non-visible database could be pruned during run time, and thus significantly reduce the rendering time. Our problem is more difficult because the MCA generated primitives are randomly distributed in the volume space, as a result there are no natural visibility occluding dividers in our database.

Foley et. al. [10], and Chen and Williams [6] proposed the orthogonal approaches to solving the problem. Instead of pruning the databases during run time, they generated images from a set of pre-defined viewing positions in the preprocessing stage. These pre-generated images are interpolated to approximate the images associated with any other viewing position. These approaches can be considered to be similar to that of the 2D image metamorphosis work (e.g. [3]), where the source and the destination images are given and the algorithms approximate an in-between image. Chen and Williams's [6] work limited the movement of the viewing position and concentrated on the visual realism (e.g. motion blurring, shadowing, etc.).

Foley et. al. [10] implemented the idea to animate and simulate interactive volume rendering where 38 viewing positions were selected at different locations on a sphere surrounding the volume space. The positions were selected by uniformly triangulating the surface of the sphere. This was done such that the area of each triangle is roughly equal. In their approach, the preprocessing is very costly because a full ray-casting volume rendition must be performed for each of the pre-defined viewpoint. The preprocessing stage took several hours on a Cray supercomputer and more than a day on a workstation [10]. Besides the large disk space that are required to store the pre-generated images, one other major shortcoming of their approach is that it is not possible to incrementally improve the quality of an interpolated image over time.

4.2 Basic Idea

We have combined the above two approaches and constructed visibility data structures from a set of pre-defined viewing positions. In this way, the number of primitives to be rendered for any given viewpoint is reduced to approximate the actual visible set. Our work is very much inspired by Foley et. al. [10], where we first determine the visible cells from a number of pre-defined viewpoints. Then, given any arbitrary viewpoint, the corresponding image can be approximated by rendering only the sum (set Union) of the surfaces visible from the pre-processed viewpoints that bound the given viewpoint. Unlike [10], we have selected 26 predefined viewpoints distributed around the center of the volume on a unit sphere. These viewpoints correspond to unit vector directing towards the origin from: (1) the 8 corners of the volume, (2) the center positions of the 6 faces, and (3) the mid-points of the 12 volume edges.

The visibility of the cells are determined with the splatting approach [27] where each cell is classified as visible if any of its vertex is visible. The visibility can be easily determined for the 26 pre-defined viewing positions. This is because when the volume space is viewed with a parallel projection, parallel lines of the cell vertices that are perpendicular to the image plane are formed. Since each line of the cell vertices projects onto a single position, a cell-vertex buffer can be created to determine the visibility of the cells easily.

4.3 Cell Visibility Preprocessing

In order to determine which are the visible triangles from a predefined viewpoint, the easiest way is to render the entire scene from that viewpoint and record the triangles that made a contribution to the final image. This would imply that the entire scene must be rendered 26 times during the preprocessing stage. To reduce the preprocessing time, we have implemented a splatting procedure (similar to that of [27]) to approximate the visibility:

```
void preprocess()
    For each predefined viewpoint
        Clear the frame-vertex buffer;
        Traverse data structure in a front-to-back manner and
        for each non-empty cell
            Splat cell onto frame-vertex buffer;
            If all +ve vertices splat to non-empty frame-vertex positions
                Clear Bit: Cell is NOT visible from the viewpoint;
            else
                Mark Bit: Cell is visible;
```

The frame-vertex buffer is similar to a frame buffer but instead of a 2D array of pixels, we have a 2D boolean array representing the visibility of the cell vertices. The buffer is initially cleared. As cells are splatted (or projected) onto the buffer in a front-to-back manner, elements in the buffer are marked to indicate that they have been covered by the cells. If all of the frame-vertex positions that a cell splats onto have been previously marked, then the cell is considered to be not visible from that particular viewpoint. It is important that the cells are splatted in a front-to-back manner. This is to ensure that the closer cells are updated on the frame-vertex buffer before the further ones.

The splatting of each cell is efficiently achieved through bit-operations using the 8-bit cell index produced (as a by-product) by the MCA. This index is originally used as an offset into the edge look-up table during the triangulation process. We have found other uses for it here. Thus no overhead is incurred during the preprocessing to obtain this index. Recall that the 8-bit index is generated in the MCA by comparing each cell vertex's density value with a user-defined threshold value. A '"1" (or +ve) is set at the corresponding bit position if the density is higher than the threshold, a "0" (or -ve) otherwise. When a cell is splatted onto the buffer, we examine if all of the +ve vertices have been covered. If they are, then we consider the cell (or the triangles in the cell) to be hidden by other cells nearer to the viewpoint.

To maintain the visibility information of each cell, we store a 26-bit code (in a 4 byte integer) in each cell (i.e. in each array element of Carr) within the data structure. Each bit represents the cell visibility in the corresponding predefined viewpoint. We shall call this code a vcode (visibility code). The vcode is used during rendering to quickly determine the cell's visibility from the current viewpoint.

4.4 Faster Rendering

When rendering a scene from a given viewpoint, the 4 bounding predefined viewpoints and their positions in the vcode are determined. A 26-bit mask code (mcode) is formed by setting ones in all the 4 bit positions and zeros elsewhere. During rendering, every cell in the higher level of the extended vertex structure is examined by performing a bitwise "AND" operation between the vcode and the mcode to determine the cell's visibility. A cell is visible if the result of the "AND" operation is non-zero. Intuitively this would mean that the cell is visible from one or more of the bounding viewpoints. If a cell is determined to be visible, all the primitives contained within the cell are rendered in the order which they are stored.

4.5 Results

We tested our algorithm on an MRI dataset with 109 slices and on a CT dataset with 113 slices. The dimensions of the slices of both datasets are 128 by 128. The rendering results before and after display preprocessing are tabulated in Table 1. Timings indicated are in real-time (seconds) and are benchmarked on a SGI Indigo Elan with 32 MBytes of main memory.

Unlike [10] where preprocessing takes up more than a day on a workstation (38 views of 500x500 images), our preprocessing requires only 1 to 2 seconds per viewpoint, a total of less than a minute for typical isosurfaces, and is independent of the image size.

Obviously, the actual achievable speedup depends on the surface complexity. In some cases, a speedup of almost 6 times were observed. This is commonly the case in the high-resolution MRI images when a threshold value of the extracted surfaces corresponds to many structures within the volume. For example the threshold value for skin also corresponds to many other internal tissues.

As illustrated in Table 1, for the MRI skin surface, on the average, only 12% of the cells are visible from each of the predefined view point. We use a semi-transparent view of the skin surface to explain the situation: The correct semi-transparent view in Plate E shows many internal structures having the same surface density as the skin surface. Although constructed in the surface extraction stage these surfaces are never visible in an opaque view (Plate F). By omitting these surfaces and the backfacing surfaces, only 17% of the cells' primitives are rendered (Plate G). This results in an average rendering speedup of 5.7 times. On CT datasets, we have experience an average speedups of 2-4 times (Plate H).

We note that the rendering speedup is slightly less than the reduction of cells rendered. For example, for the MRI skin surface, there is an average of 7.4 times reduction in the cells rendered whereas the rendering speedup factor is 5.7 times. This is due to the overheads involved in (1) run-time visibility determination of cells, and (2) the additional indirections in structure traversal (the full rendering traverses only the lower level indexed polygon structure).

We caution that this technique is only relevant in the displaying of opaque surfaces. When displaying semi-transparent surfaces, almost all triangles make contributions to the final image. Hence visibility determination would be incorrect and irrelevant. We have intentionally rendered such incorrect images to illustrate the bulk of polygons that would not have been rendered during display (Plate G and H). Notice that the "far" side of the surface and the internal structures were not rendered.

5 Conclusion

We have introduced (1) a data structure to organize the geometric results of an isosurface extraction from volume data, (2) a simple and effective depthwise traversal scheme of the

Volume Data	CT (128x128x113)		MRI (128x128x109)	
	Bone	Skin	Skin	Brain
Threshold Value	0.58	0.19	0.14	0.31
Surface Extraction Time (Secs)	66.32	68.01	95.85	62.70
Number of Triangles	211,114	234,490	434,724	202,802
Number of Intersected Cells	104,273	116,168	207,993	97,737
Avg. Visible Cells	25.35%	32.97%	13.49%	20.29%
Min. Visible Cells	30%	37%	12%	33%
Max. Visible Cells	50%	64%	23%	39%
Pre-Processing Time (Secs)	28.39	33.04	67.31	27.12
Full Rendering Time (Secs)	9.42	10.2	19.53	9.19
New Rendering Time (Secs)	3.77	5.1	3.41	3.31
Speedup	2.5	1.98	5.7	2.7

Table 1: Performance and Analysis of Visibility Preprocessing

data structure, (3) a scheme to render multiple semi-transparent isosurfaces, and (4) a visibility preprocessing approach to accelerate the rendering of opaque and topologically closed isosurfaces. We have experienced speedups of 2 to 5 times for high resolution medical datasets.

The techniques described in this paper have been implemented and incorporated into a medical visualization software developed at the Institute of System Science of the National University of Singapore [11]. Currently, the system is being used by the medical staffs in various hospitals in Singapore, as well as the Johns Hopkins Hospital in the United States for medical research and educational purposes.

We are investigating in the approaches to incorporate a mesh simplification or polygon merging algorithm (e.g. [21, 25, 12]) with our visibility preprocessing technique. If, for example, the larger polygon patches can be bounded by a set of boundary cubes, the visibility preprocessing techniques described in this paper would still be applicable. If the polygon merging scheme adopted is effective, we would expect a further speedup.

Acknowledgments

With thanks to Wieslaw Nowinski, Lek Ngee, and Bonnie Nguyen for their input and kind assistance; To Ying Huey for her encouragement and patience; And to the University of North Carolina at Chapel Hill for making the MRI and CT datasets available through public domain.

References

[1] John M. Airey, John H. Rohlf, and Frederick P. Brooks Jr. Towards image realism with interactive update rates in complex virtual building environment. *Computer Graphics*, 24(2):41–50, March 1990. ACM Workshop on Interactive Graphics Proceedings.

[2] Doi Akio and Kiode Akio. A cell-traverse display algorithm for regular and rectilinear 3d grid data. *The journal of Visualization and Computer Animation*, 3:129–145, 1992.

[3] Thaddeus Beier and Shawn Neely. Feature based image metamorphosis. *Computer Graphics*, 26(2):35–42, 1992.

[4] J. F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics*, 16(3):21–9, 1982.

[5] Loren Carpenter. The a-buffer, an antialiased hidden surface method. *Computer Graphics*, 18(3):103–108, July 1984. ACM Siggraph '84 Conference Proceedings.

[6] ShenChang Eric Chen and Lance Williams. View interpolation for image synthesis. *Computer Graphics Proceedings*, pages 279–288, 1993. SIGGRAPH'93 Annual Conference Series.

[7] H. E. Cline, W. E. Lorensen, , and S. Ludke. Two algorithms for the three-dimensional reconstruction of tomograms. *Medical Physics*, 15(3), May/Jun 1988.

[8] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. *Computer Graphics*, 22(4):65–74, 1988.

[9] T. Todd Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.

[10] Thomas A. Foley, David A. Lane, and Gregory M. Nielson. Towards animating ray-traced volume visualization. *The Journal of Visualization and Computer Animation*, 1:2–8, 1990.

[11] P.A. Heng, N. Lek, S. Meiyappan, B.T. Nguyen, H.T. Nguyen, W.L. Norwinski, P.K. Pillay, P. Phavichitr, R. Raghavan, and C.P Yu. Mira medical imaging system. Technical Report TR93-117-0, Institute of Systems Science, National University of Singapore, May 1993.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *Computer Graphics Proceedings*, pages 19–26, 1993.

[13] J.T. Kajiya and B.P. von Herzen. Ray tracing volume densities. *Computer Graphics*, 18(3):165–74, 1984.

[14] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[15] W.E. Lorensen and H.E. Cline. Marching cubes: A high-resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), July 1987.

[16] Willaim E. Lorensen and Harvey E. Cline. Volume modeling. In Marc Levoy, editor, *Tutorial on Volume Visulization Algorithms*. Visualization '90, San Francisco, California, October 1990.

[17] Bradley A. Payne and Arthur W. Toga. Surface mapping brain function on 3d models. *IEEE Computer Graphics and Applications*, pages 33–41, September 1990.

[18] S.M. Pizer, H. Fuchs, and C. Mosher. Three-dimensional shaded graphics in radiotheraphy and diagnostic imaging. *NCGA '86 Conference Proceedings*, pages 10–13, 1986.

[19] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, July 1984. ACM Siggraph '84 Conference Proceedings.

[20] M.L. Rhodes and Yu-Ming Kuo. Simple three-dimensional image synthesis techniques for serial planes. *SPIE Medical Imaging II*, 914:1286–1289, 1988.

[21] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26:65–70, July 1992.

[22] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, Novermber 1990.

[23] Silicon Graphics Computer Systems. *Graphics Library Programming Guide*. Silicon Graphics, 1991.

[24] Seth J. Teller and Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. *Computer Graphics*, 25(4):61–69, July 1991. ACM Siggraph '91 Conference Proceedings.

[25] G. Turk. Re-tiling polygonal surfaces. *Computer Graphics*, 26(2):55–64, July 1992.

[26] Heang K. Tuy and Lee Tan Tuy. Direct 2-d display of 3-d objcets. *IEEE Computer Graphics and Applications*, pages 29–33, October 1984.

[27] Lee Westover. Footprint evalutaion for volume rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[28] B. Wyvill and D. Jevans. Table driven polygonisation. *Modeling and Animating with implicit Surfaces*, August 1990. SIGGRAPH '90 Course Notes 23.

[29] G. Wyvill, C. McPheeters, , and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, August 1986.

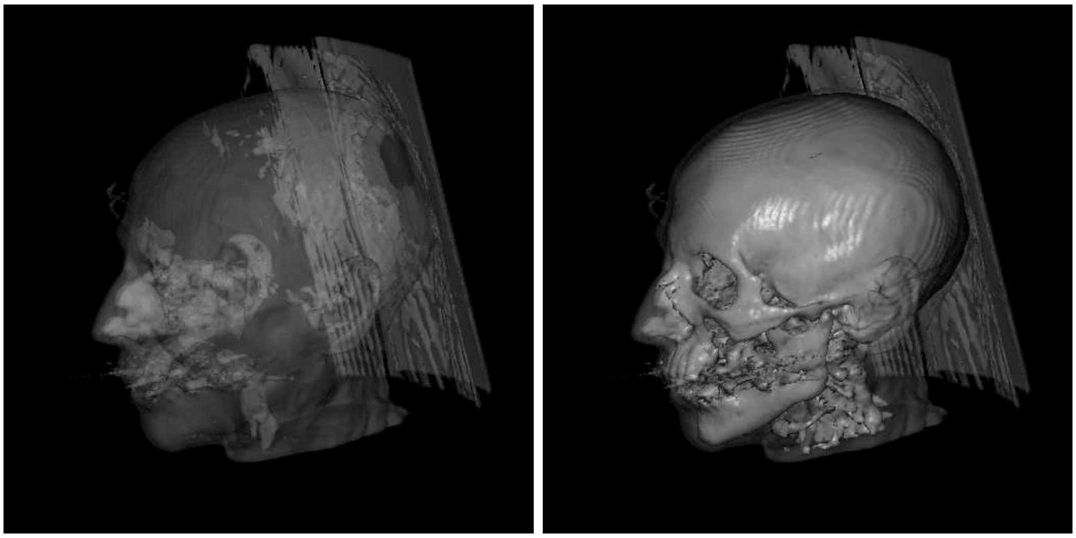

Plate A: Semi-transparency: (Left) Single semi-transparent CT skin surface;(Right) CT bone surface within semi-transparent skin.

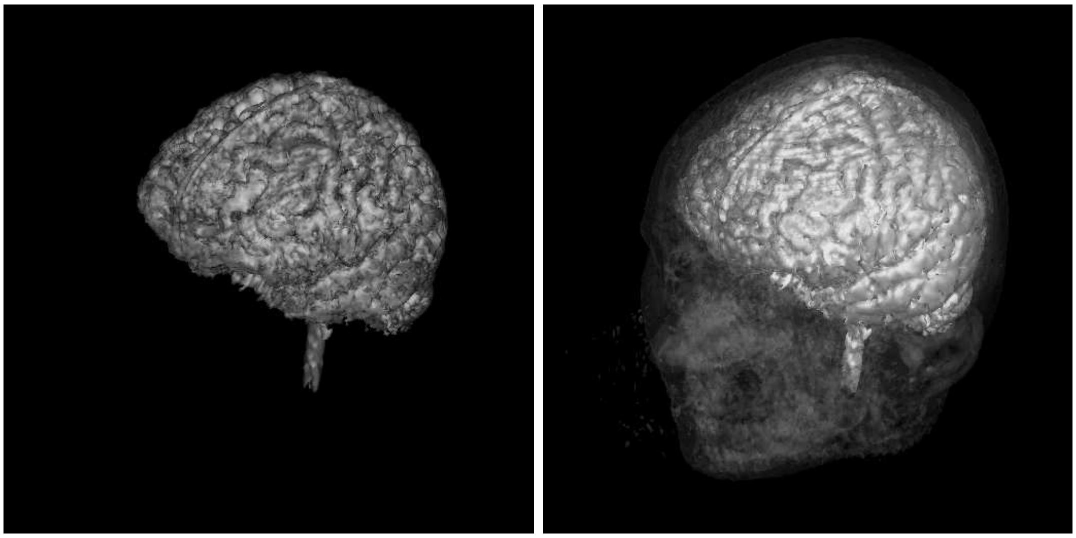

Plate B: Semi-transparency: (Left) MRI cortical surface; (Right) MRI cortical surface within semi-transparent skin.

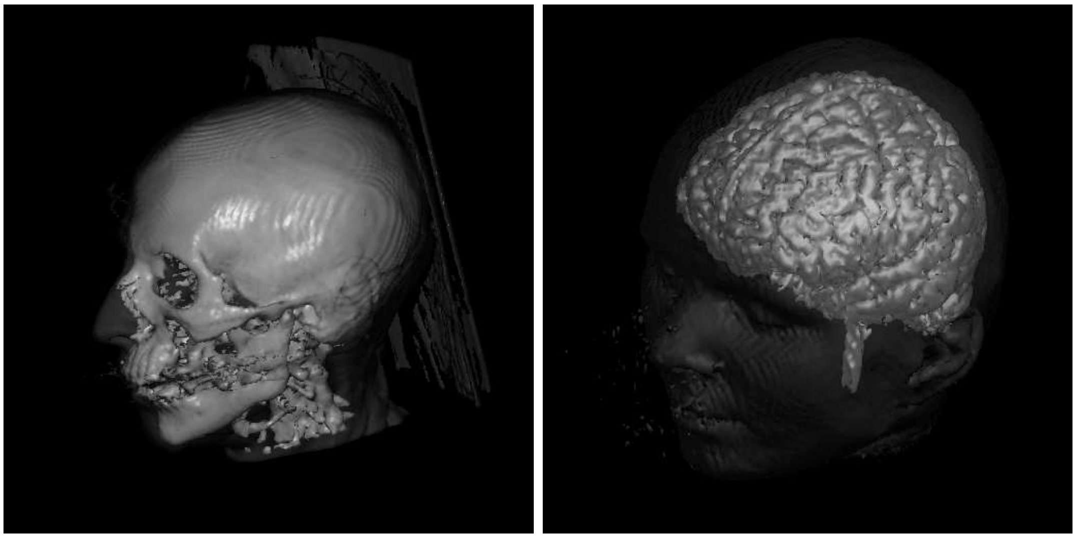

Plate C and D: Incorrect semi-transparency results rendered by compositing two separately rendered images. (C) CT skin and bone surfaces; (D) MRI cortical and skin surfaces.

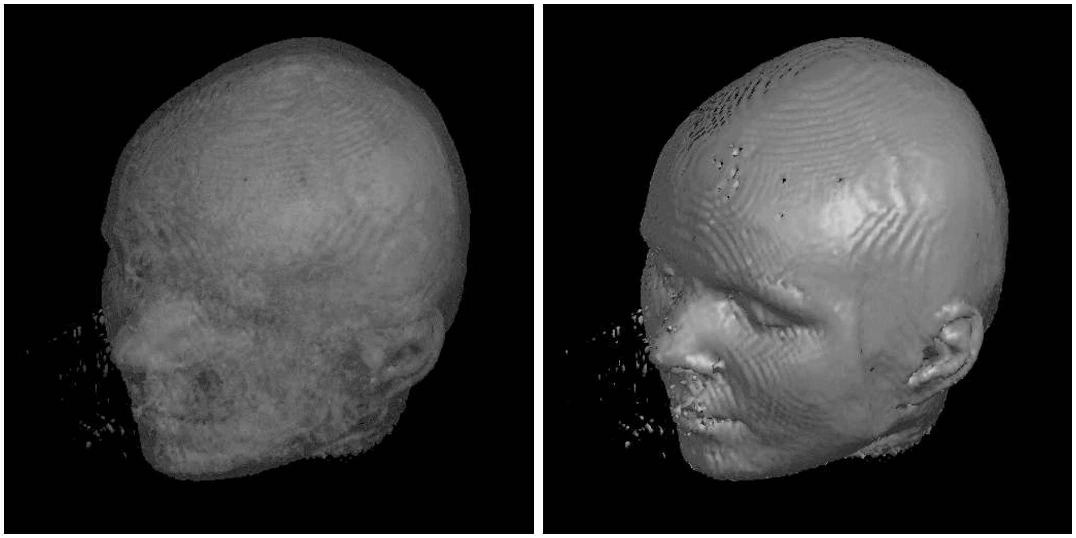

Plate E and F: Visibility preprocessing: (E) MRI semi-transparent skin surface, rendered with full set of primitives; To illustrate the internal structures extracted in MCA that are not normally visible in an opaque closed surface (F).

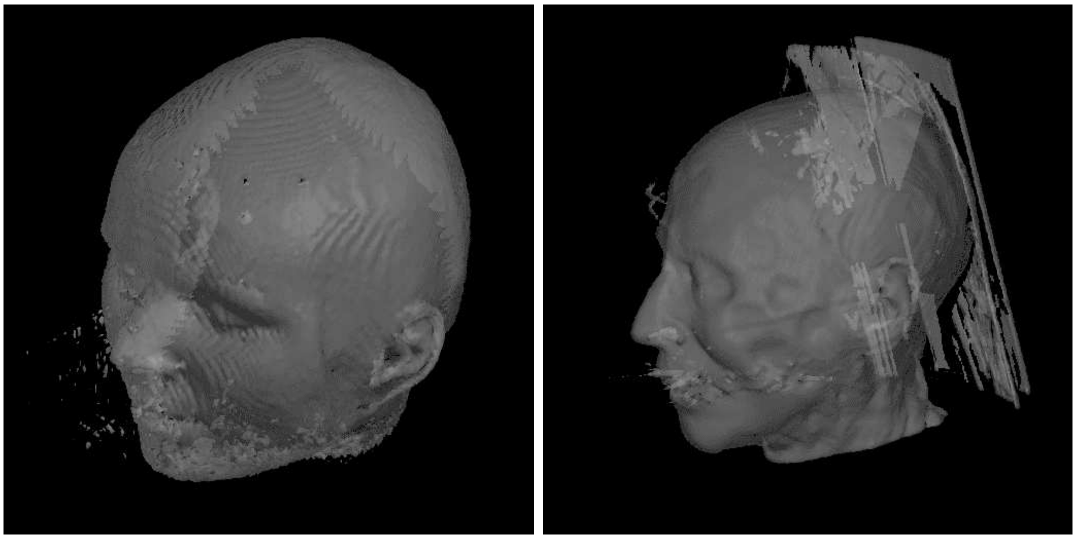

Plate G and H: Incorrect semi-transparent surfaces to illustrate the bulk of primitives that are not rendered after visibility preprocessing. (G) MRI skin surface, compare this with Plate E which shows the correct semi-transparent view; (H) CT skin surface, compare with Plate A which shows the correct semi-transparent view.