Efficient Volumetric Ray Casting for Isosurface Rendering

Jae Jeong Choi[1] Byeong-Seok Shin[2] Yeong Gil Shin[1] Kevin Cleary[3]

1. Department of Computer Science, Seoul National University, Seoul, Republic of Korea

2. Department of Computer Science and Engineering, Inha University, Inchon, Republic of Korea

3. ISIS Center, Department of Radiology, Georgetown University Medical Center, Washington, D.C., USA

Abstract

While volume rendering is becoming more widely used in medical applications, it is still difficult to generate a good quality image interactively without expensive hardware when the size of the dataset is quite large. For interactive rendering of a large dataset, we present an acceleration method, image-space bounding surface. Using an image-space bounding surface, the isosurface ray casting is accelerated by avoiding unnecessary volume traversals. Image-space bounding surface can be interactively handled by polygon rendering hardware even in a conventional personal computer. Two optimization techniques, LF-minmax map and memory bricking, are also employed to efficiently render isosurfaces. This paper also shows that the algorithm can be extended to multiple isosurfaces rendering. The experimental results show that the algorithm generates a good quality image of a large dataset interactively on a standard PC platform.

Keywords : volume rendering, isosurface ray casting, polygon rendering hardware, image-space bounding surface

1. Introduction

Many medical applications generate scalar fields from imaging devices such as computed tomography (CT) or magnetic resonance imaging (MR). The scalar fields can be visualized by volume rendering algorithms that can be categorized into surface rendering and direct volume rendering techniques. While surface rendering displays isosurfaces in the volume, direct volume rendering visualizes a volume according to an opacity transfer function. Regardless of the volume rendering technique, it is difficult to produce a good quality image without expensive hardware such as multi-processor workstations. In this paper, we present an efficient isosurface ray casting method for a large dataset with inexpensive hardware.

Polygon-based rendering approaches [6, 16] such as Marching Cubes are the most common methods in surface rendering. They generate polygons that approximate isosurfaces from a volume. The extracted polygons are rendered by a polygon rendering algorithm. Even if polygon-based rendering is usually faster than volumetric ray casting, it has some problems. First, it takes a lot of time to change the isosurface of interest in polygon-based rendering, which makes it difficult to do this in real time. Second, for a large dataset, polygon-based rendering generates a huge amount of polygons that cannot be easily managed even on a high-end graphics workstation. Third, it is not straightforward to visualize a cutting plane or the interior of the volume in polygon-based rendering.

Ray casting [2,3,4,13], the most frequently used algorithm in direct volume rendering, has also been used for displaying isosurfaces [5,8,14]. Isosurface ray casting[1] generates rays from a viewpoint through each screen pixel into volume space, and it finds the intersection point of the ray and the isosurface. The intersection point can be found by an analytic method [5] or an interpolation method [14]. The intensity value of the pixel is calculated at the intersection point.

In isosurface ray casting, traversal operations consume the most time. Several techniques have been used

[1] In the remainder of this paper, we define isosurface ray casting as ray casting for isosurfaces.

for efficient traversal. Hierarchical data structures such as octrees [2,8] or K-d trees [12] can be used to skip over empty regions by using a macrocell that contains the minimum and maximum value for its children nodes. Using macrocells, the ray/surface intersection test can be simplified by comparing the isosurface value with the minimum and maximum value.

A distance volume that contains the distance to the nearest possibly contributing cell has also been developed [11,17]. In these methods, a significant amount of additional memory is required to store the distance values, and building the distance volume can make it difficult to interactively modify the isosurface.

The presence-accelerated ray casting [15] estimates a highly accurate object presence by projecting all grid cells associated with the object boundary onto the image plane without using polygon rendering hardware. This technique requires a preprocessing stage to generate a classified volume, which makes it difficult to change the isosurface of interest. Even if an interactive classification is provided, the performance of the algorithm is degraded.

The PARC (Polygon Assisted Ray Casting) algorithm [1,10] estimates the surface location using polygon rendering hardware. An approximation of the isosurfaces is projected into image-space, and the information from the Z-buffer is used to identify segments of the ray that could possibly contribute to the final image. For the approximation of the isosurfaces in the PARC algorithm, a bounding surface is created in volume space. Even if the rendering time of the PARC algorithm is reduced greatly, too many polygons are generated when the size of the dataset is quite large such as the Visible Human dataset [7]. A hierarchical volume can be employed to reduce the number of polygons to be interactively handled by polygon rendering hardware, but ray casting performance is degraded because the number of cells to be processed increases.

This paper presents an efficient isosurface ray casting method for a large dataset. We propose an acceleration technique using an image-space bounding surface for interactive rendering. Using an image-space bounding surface, the performance of isosurface ray casting can be greatly enhanced. Since the number of polygons in an image-space bounding surface is not affected by the size of dataset, it is possible to render a large dataset interactively. For fast searching of isosurfaces, two optimization techniques, LF-minmax map

and memory bricking, are also employed. The experimental results show that the new algorithm provides interactive performance for a large dataset on a standard PC platform as well as on a high-end graphics workstation.

In Section 2, the basic algorithm of isosurface ray casting is explained. In Section 3, an image-space bounding surface is presented. Section 4 describes two optimization techniques, LF-minmax map and memory bricking. In Section 5, the isosurface ray casting algorithm is extended to render multiple isosurfaces. In Section 6, some rendering results and computation times are shown. Finally, conclusions and suggestions for future work are given.

2. Ray Casting for Isosurface Display

In this section, we explain the ray casting method for isosurface display. Assume that each voxel represents the density of material at a point and an isosurface value is a certain density value. From the intersection between the ray and the bounding box of the volume, voxels are sampled at regular intervals. The condition that an isosurface lies in the interval $[t_i, t_{i+s}]$ is :

$$D(t_i) \le d_{iso} \le D(t_{i+s}) \quad \text{or}$$

$$D(t_{i+s}) \le d_{iso} \le D(t_i)$$

where

t_i : distance from the viewpoint to the point i

$D(t_i)$: density value at t_i

d_{iso} : isosurface density value

s : sampling distance

When the above condition is satisfied, there exists t_{iso} in $[t_i, t_{i+s}]$, which meets the following condition since the trilinear interpolation function is continuous.

Condition : $D(t_{iso}) = d_{iso}, t_i \le t_{iso} \le t_{i+s}$

Several methods have been devised to find t_{iso}, one of which is the analytic method. The analytic method [5] models the isosurface as an analytic function, and the intersection between a ray and the isosurface is computed directly from the function. In the interpolation method, the isosurface can be found by interpolation of the surface location between successive sample points. The interpolation method generates a better image than the analytic method (Figure 1) because the interpolation method continuously searches the exact isosurface in the cell instead of estimating the surface location from the analytic function.

3. Accelerated Ray Casting Using Image-Space Bounding Surface

In isosurface ray casting, traversal operations consume the most time. The performance can be enhanced if we reduce the number of traversals. A set of surfaces, called a bounding surface, which contain all the possibly contributing cells is used to reduce the number of traversal operations. For example, in Figure 2, only the thick lines in Figure 2(b) need to be traversed instead of all the thick lines in Figure 2(a).

The PARC algorithm [10] generates a bounding surface in volume space. While the PARC algorithm generates a bounding surface in volume space, our method constructs a bounding surface in image space. It is called an image-space bounding surface, and it has the following advantages. First, its rendering time is proportional to the image size not the volume size. Second, all the possibly contributing pixels can be anticipated, which avoids generating unnecessary rays.

A flowchart of our method is shown in Figure 3. At the initialization step, the isosurface ray casting algorithm without a bounding surface must be run to generate depth values which are necessary for constructing an image-space bounding surface. During the isosurface ray casting, the depth value, the distance from the viewpoint to the isosurface, is saved for each pixel. If a ray does not intersect with the isosurface, the distance from the viewpoint to the far plane of the volume is saved for the pixel.

Once a depth buffer is generated from the isosurface ray casting, an image-space bounding surface can be constructed. Figure 4(a) and (b) show how to construct an image-space bounding surface. The rectangle representing a pixel is added into the image-space bounding surface as depicted in Figure 4(a). To fill the gap between pixels from the depth difference, rectangles connecting two pixels are added into the image-space bounding surface as shown in Figure 4(b). Figure 4(c) and (d) show example images of an initial image-space bounding surface and a rotated image-space bounding surface.

After constructing a bounding surface, it is sent to the polygon rendering hardware. In the rendering process, depth values are saved for each pixel in the Z-buffer. With the depth values, isosurface intersection calculations are performed on a cell-by-cell basis along the ray as described in Section 2. The image-space bounding surface must be reconstructed whenever the viewpoint or the isosurface of interest is changed.

When the number of polygons in the image-space bounding surface becomes large, the bounding surface can be constructed in lower resolution. However, there are some trade-offs between the polygon rendering time and isosurface ray casting time according to the resolution of the bounding surface. As the bounding surface is generated in higher resolution, the isosurface ray casting becomes more efficient because more accurate skipping tests are possible. However, a higher resolution bounding surface requires more polygon rendering time.

Current personal computers as well as high-end graphics workstations have the ability to render a number of polygons. Polygon rendering hardware can accelerate the estimation of isosurface locations, which can enhance the rendering performance greatly.

4. Further Optimization for Interactive Isosurface Rendering

Even if an image-space bounding surface approximates surface locations well, the algorithm has to traverse many empty cells. In order to traverse empty cells efficiently, two other optimizations are employed in our implementation.

Some algorithms used a hierarchical data structure such as an octree to traverse the volume efficiently [2].

However, an octree requires a complex operation to advance along the ray. We use an LF (Loose-Fitting) minmax map[2] to simplify the ray sampling.

In a traditional octree, the intersection points **a, b, c, d** in Figure 5(a) have to be checked to find the nodes I, II, III that may contribute to the ray. In the LF-minmax map, the nodes are checked with equal spacing as shown by points **a, b, c** in Figure 5(b). In this case, it is possible to miss the part of the line labeled **i** in Figure 5(b) and corresponding to the line between points **b** and **c** in Figure 5(a). This can result in an inaccurate skipping test. To solve this problem, we expand each node in the LF-minmax map to cover a larger area than the original area. The function to compute an intermediate level node is $f(x,y,z) = n_{\lfloor x/m \rfloor \lfloor y/m \rfloor \lfloor z/m \rfloor}$ where the sampling point is (x, y, z) and the size of the intermediate level is $m \times m \times m$. The node $n_{i,j,k}$ saves the minimum and maximum value in the region of [$mi-\beta, mi+m+\beta$], [$mj-\beta, mj+m+\beta$], [$mk-\beta, mk+m+\beta$]. Using a larger β value prevents the ray from missing the interval **bc** in Figure 5(a). The shaded part in Figure 5(d) represents the part that can cause a problem when the sampling distance and node size are the same.

The larger the value of β, the less chance the sample will be missed, but the more chance the skipping test will fail. Therefore, the smallest β that does not miss the sample should be chosen. When the node size is 1 and the sampling distance is 1, the best value of β is $\frac{1}{2\sqrt{2}}$ as shown in Figure 5(d).

When an LF-minmax map is used as an intermediate level of the hierarchical structure for the skipping test, the skipping test is performed using the sampling distance of the intermediate level. If the skipping test fails, the level of the LF-minmax map is decreased. Assume that the size of a node is $m \times m \times m$ and the sampling distance is m. If the skipping test fails at the sampling point t, the interval that may include the isosurface is [$t-m$, $t+m$]. Therefore, the algorithm checks the minmax map in the lower level and resumes at the $t+m$ point.

Another optimization technique used in our implementation is memory bricking [8]. Current CPUs employ high-speed cache memory between the main memory and processor to improve performance. In the isosurface ray casting as well as ray casting, the voxel data is referenced often. By repositioning the voxel data within the

[2] A minmax map is an octree that has the minimum and the maximum value of the included cells.

memory, the performance of the algorithm is increased because of the spatial coherence.

5. Multiple Isosurfaces Rendering

In this section, we extend the isosurface ray casting algorithm to render multiple isosurfaces. In the case of *m* isosurfaces, a list of isosurfaces can be defined as follows :

ρ_0 : minimum density value

$\rho_1, \ldots, \rho_m$: sorted list of density values of isosurfaces in increasing order

ρ_{m+1} : maximum density value

A level of the ray is defined *l* if the density value of the current sample point is between ρ_l and ρ_{l+1}. Along the ray, for a sampled position of the ray t_i and at a level *l*, $D(t_i)$ satisfies :

$$\rho_l \le D(t_i) \le \rho_{l+1}$$

At the next position t_{i+s}, the following two conditions are checked to determine whether the interval $[t_i, t_{i+s}]$ includes isosurfaces :

Condition (a): $D(t_{i+s}) \ge \rho_{l+1}$

Condition (b): $D(t_{i+s}) < \rho_l$

When condition (a) is satisfied, the interval includes the ρ_{l+1} isosurface. The rendering algorithm thus calculates a shading value on the ρ_{l+1} isosurface and increases the level *l* by 1. When condition (b) is satisifed, the interval includes the ρ_l isosurface, so the rendering algorithm calculates a shading value on the isosurface

ρ_l and decreases the level l by 1. This process is repeated until the sample position exits the volume or the accumulated opacity exceeds an opacity threshold.

An example is shown in Figure 6 for two isosurfaces, denoted by the density values ρ_1 and ρ_2. A sample ray is shown, and the initial level of the ray is zero. When the sampled value becomes greater than ρ_1 (here, at the point t_{i-s}), the algorithm finds the exact isosurface for ρ_1 (point **a**), increases the level of the ray by one and continues to find the next isosurface. At the point t_{i+s}, the sampled value is larger than ρ_2. The algorithm finds the exact isosurface for ρ_2 (point **b**), sets the level of the ray to two, and continues. The opposite case occurs at the point t_{i+2s}, where the sampled value becomes smaller than ρ_2. In this case, the algorithm finds the isosurface for ρ_1 (point **c**), and decreases the level to one.

6. Experimental Results

Our algorithm was implemented on a Silicon Graphics Onyx 10000 (a 196 Mhz R10000 processor, Infinite Reality Graphics Board, 512 Mbytes main memory) and on a Pentium II personal computer (a 450Mhz processor, a Matrox Graphics Millenium G200 graphics board, 320 Mbytes main memory).

The algorithm was applied to the Visible Man dataset to show its ability to handle a large dataset. The Visible Man dataset is available through the National Library of Medicine [7]. We used the first 512 slices from the CT dataset (512x512, 8 bits[3]). The size of total dataset is 128 Mbytes. The sample images rendered using our algorithm are shown in Figure 7. Figure 7(a) shows the rendered image for two surfaces, skin and bone. Figure 7 (b) and (c) show rendered images for one isosurface, skin and bone, respectively.

Table 1 shows the performance in generating the images shown in Figure 7. For comparison purposes, we also implemented the PARC algorithm for multiple isosurfaces. We measured the rendering time for four

[3] Originally, the pixel size of the Visible Man CT dataset is 12 bits. To fit the whole dataset into the main memory, each pixel was converted to 8 bits. The quality of the images might be affected by this conversion.

cases : 1) standard isosurface ray casting with no optimization, 2) PARC algorithm, 3) image-space bounding surface only and 4) all optimization techniques. As described in Section 3, the rendering times of case 3 and case 4 are divided into 3 components, a) polygon rendering, b) ray casting and c) bounding surface generation.

In the PARC algorithm, the polygon rendering time depends on the macrocell size. We tested the PARC algorithm using “2x2x2” to “16x16x16” macrocells. The rendering times of the PARC algorithm reported in Table 1 are the best results found. In our algorithm, the rendering time is related to the rotation angle between successive renderings. The rendering times of case 3 and case 4 in Table 1 were acquired when the images were rotated 5 degrees around x and y axes from the previous images. If an image is rotated 20 degrees, it requires about 20 percent more rendering time.

Table 1 shows that our algorithm is about 1.8-3.8 times faster than PARC algorithm. For example, in rendering Figure 7(c) on the PC, case 1 took 21.24 seconds, case 2 took 1.63 seconds, case 3 took 0.59 seconds and case 4 took 0.51 seconds. Using all optimizations, our algorithm(case 4) is 3.2 times faster than the PARC algorithm(case 2) in this example.

To show the efficiency of our algorithm, we measured surface extraction times and polygon rendering times using the marching cubes algorithm in the Visualization Toolkit [9] and the same dataset. To extract the skin and bone isosurfaces from the dataset, it took 206 seconds and 220 seconds, respectively, in the SGI Onyx. The surface rendering time for extracted isosurfaces was more than 2 minutes in each case. Compared with our rendering time, the result is almost 100 times longer.

Table 1 also shows that the other optimization techniques, LF-minmax map and memory bricking, become more effective when an image for multiple isosurfaces is generated because traversal operations can be simplified by the techniques. For example, in rendering Figure 7(a) on the PC, case 4 (1.53 seconds) is 1.91 times faster than case 3 (2.92 seconds). However, in rendering Figure 7(b) on the PC, case 4 (0.66 seconds) is only 1.15 times faster than case 3 (0.76 seconds).

Figure 8 and Figure 9 show example images rendered by our algorithm. Figure 8 shows some images with a cutting plane using two datasets, CT head (256x256x225, 8 bits) and MR brain (128x128x84, 8 bits). These images show the ability of our algorithm to render multiple isosurfaces of good quality on a PC platform.

Figure 9 shows the images as the isosurface of interest changes with the CT head dataset.

7. Conclusion and Future Work

This paper presented an interactive ray casting algorithm for a large dataset. To accelerate isosurface ray casting, we proposed an image-space bounding surface. An image-space bounding surface is used to prune out unnecessary regions of volume. Regardless of the size of the dataset, an image-space bounding surface can be interactively rendered by inexpensive polygon rendering hardware. The algorithm also employs LF-minmax map and memory bricking for fast searching of isosurfaces. The experimental results show that the new algorithm generates a high-quality image, 1.8-3.8 times faster than PARC algorithm.

In our experience, the rendering performance of the optimization techniques is affected by the characteristics of volume data. Therefore, in future work, how the structural characteristics of volume data affect the performance of optimization techniques should be studied.

As hardware is being developed rapidly, new rendering techniques that take advantage of hardware are required. This paper discussed how to use standard graphics hardware in isosurface ray casting. In the future, other hardware, such as parallel processors, might be used with this algorithm to improve the rendering time.

Acknowledgements

This work was funded in part by U.S. Army grant DAMD17-99-1-9022. The content of this manuscript does not necessarily reflect the position or policy of the U.S. government.

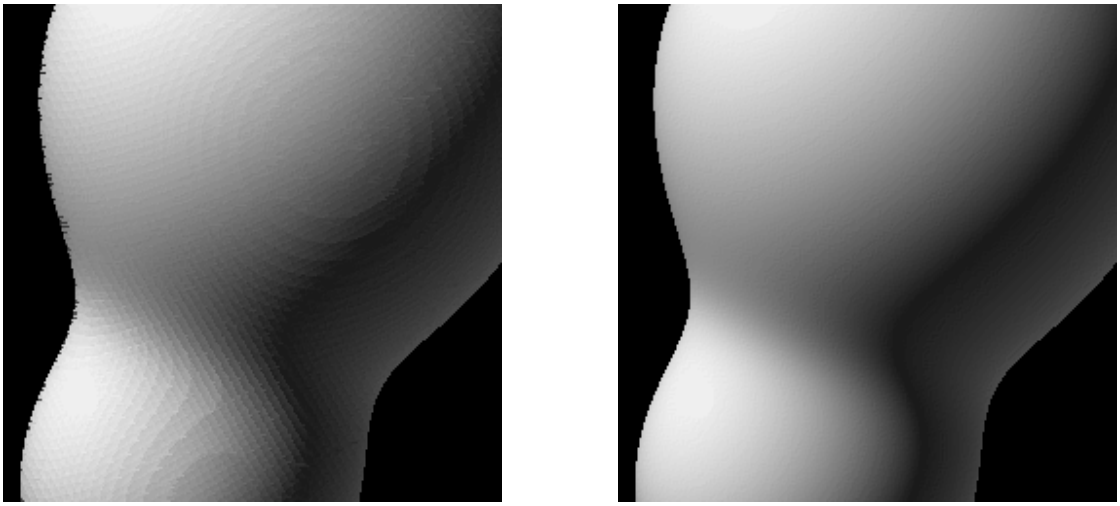
References

1. Avila, R., Sobierajsk, L. and Kaufman, A., Towards a Comprehensive Volume Visualization System, IEEE Visualization 92, 1992 : 13-20.

2. Levoy, M., Efficient Ray Tracing of Volume Rendering, ACM Transactions on Graphics 1990 ; 9(3) : 245-261, 1990.
3. Levoy, M., Volume Rendering : Displays of Surfaces from Volume Data, IEEE Computer Graphics and Applications 1988; 8(3) : 29-37.
4. Levoy, M., Volume Rendering by Adaptive Refinement, The Visual Computer 1990; 6(1) : 2-7.
5. Lin, C., Ching, Y., An Efficient Volume-Rendering Algorithm with an Analytic Approach, The Visual Computer 1996; 12(10) : 515-526.
6. Lorensen, W., Cline, H., Marching Cubes : A High Resolution 3D Surface Construction Algorithm, Computer Graphics 1987; 21(4) : 163-169.
7. National Library of Medicine (U.S.) Board of Regents, Electronic Imaging : Report of the Board of Regents, U.S. Dept. of Health and Human Services, Public Health Service, National Insti. Of Health, NIH Publication 1990; 90-2197.
8. Parker, S., Shirley, S., Livnat, Y., Hansen, C., Sloan, P., Interactive Ray Tracing for Isosurface ray casting, IEEE Visualization 98, 1998 : 233-238.
9. Schroeder, W., Martin, K., Lorensen, B., The Visualization Toolkit : An Object-Oriented Approach To 3D Graphics, 2nd Edition, Prentice Hall, 1997.
10. Sobierajski, L., and Avila, R., A Hardware Acceleration Method for Volumetric Ray Tracing, IEEE Visualization 95, 1995 : 27-34.
11. Sramek, M., Fast Surface Rendering from Raster Data by Voxel Traversal Using Chessboard Distance, IEEE Visualization 94, 1994 : 188-195.
12. Subramanian, K., and Fussell, D., Applying Space Subdivision Techniques to Volume Rendering, IEEE Visualization 90, 1990 : 150-158.
13. Tiede, U., Hohne, K., Bomans, M., Pommert, A., Riemer, M., Wiebecke, G., Investigation of Medical 3D-Rendering Algorithms, IEEE Computer Graphics and Applications, 1990; 10(2), 41-53.
14. Tiede, U., Schiemann, T., Hohne, K., High Quality Rendering of Attributed Volume Data, IEEE

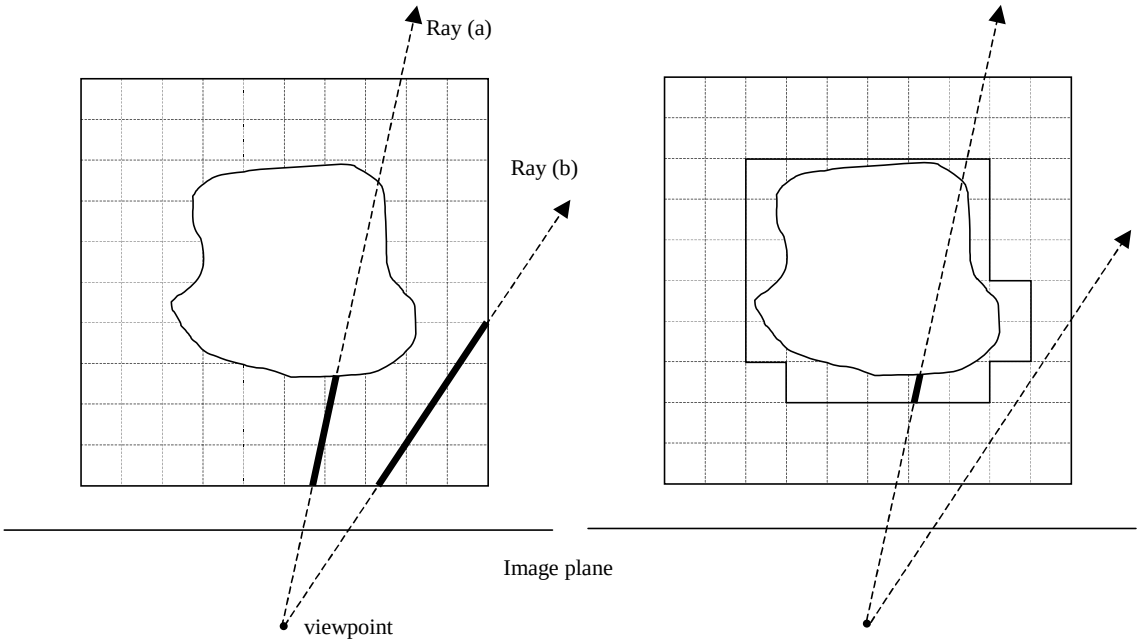
Visualization 98, 1998 : 255-262.

15. Wan, M., Kaufman, A. and Bryson, S., High Performance Presence-Accelerated Ray Casting, IEEE Visualization 99, 1999 : 379-386.
16. Wilhelms J., Van Gelder, A., Octrees for Faster Isosurface Generation, ACM Transactions on Graphics 1992; 11(3) : 201-227
17. Zuiderveld, K., Koning, A. and Viergever, M., Acceleration of Ray-Casting Using 3D Distance Transforms, Visualization and Biomedical Computing, 1992 : 324-335.



(a) Analytic method (b) Interpolation method

Figure 1. Images rendered by isosurface ray casting algorithm. An analytic method is used to generate image (a), and an interpolation method is used to generate image (b).



(a) brute-force **(b) bounding surface**

Figure 2. A 2D example of isosurface ray casting. When a bounding surface is applied to the ray casting, unnecessary volume traversal can be avoided.

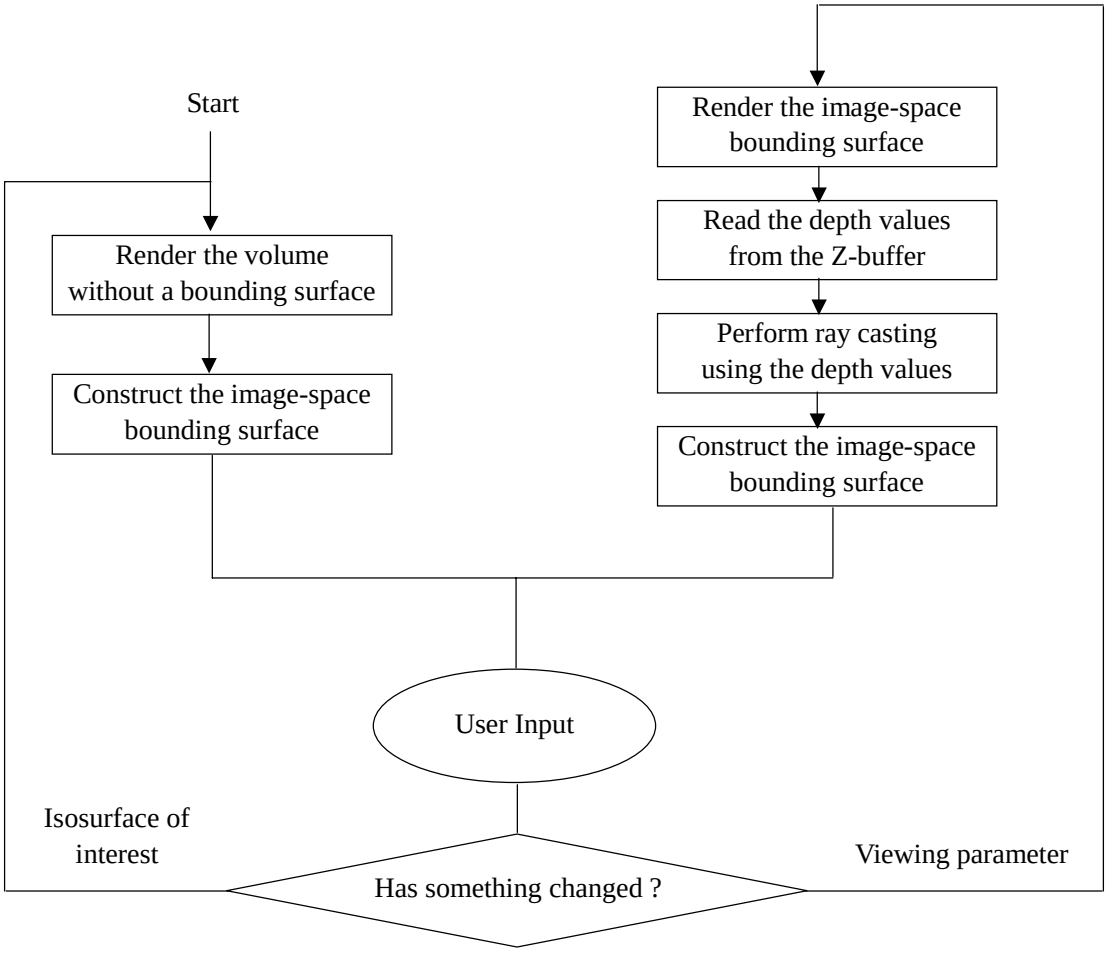


Figure 3. Flowchart of the image-space bounding surface algorithm.

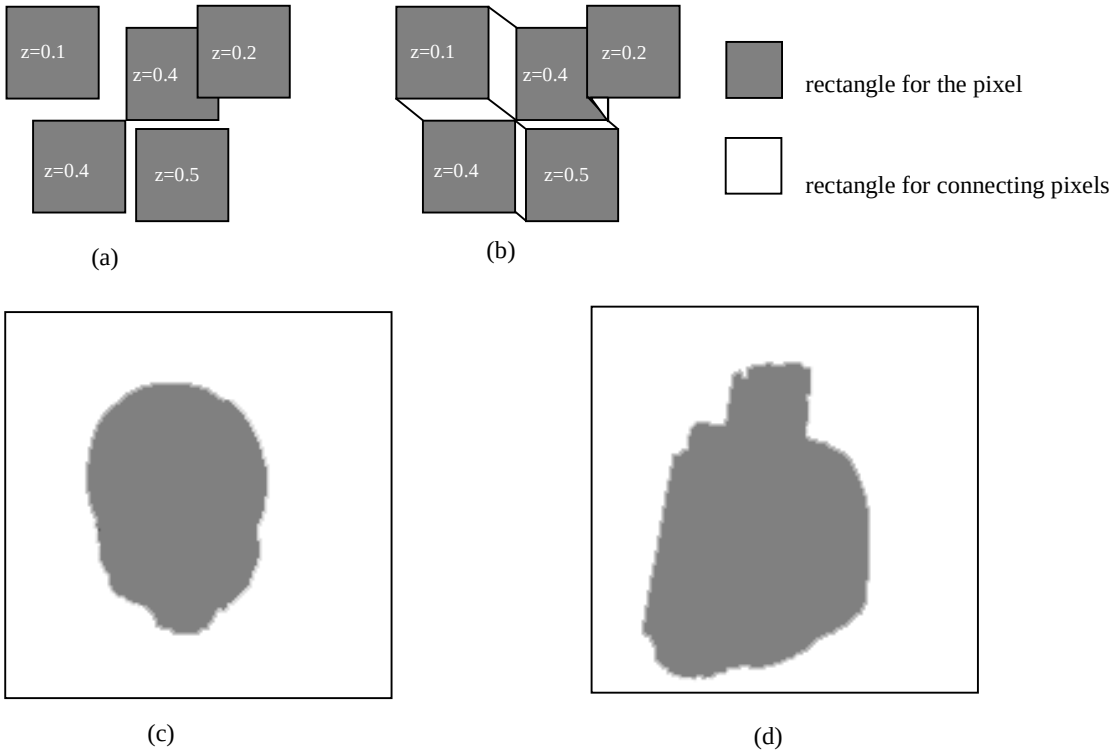


Figure 4. An image-space bounding surface is used to anticipate the pixels to be processed in the next image.

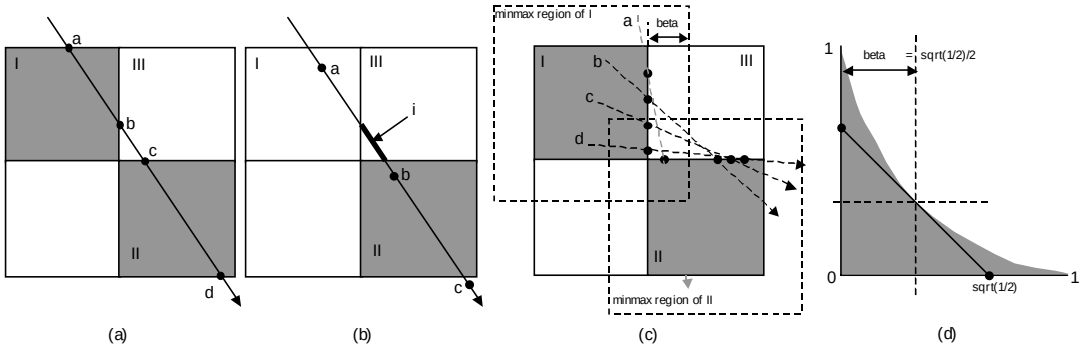


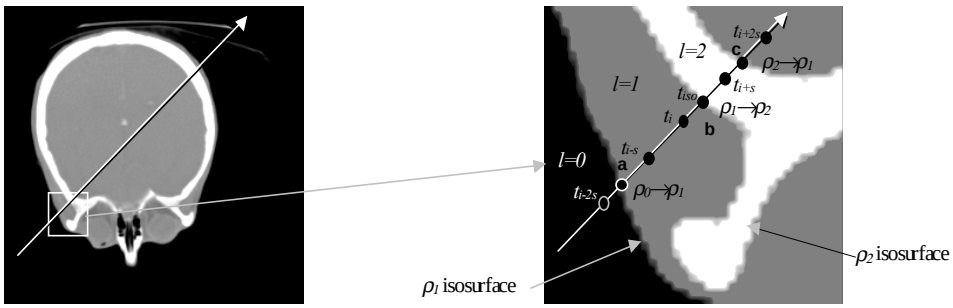
Figure 5. The construction process of LF-minmax map.

(a) Node a, b, c, d are checked when using an octree, which makes the advance operation complex

(b) Nodes are checked in regular step when using LF-minmax map

(c) The range of minimum and maximum value of each node in the LF-minmax map

(d) The appropriate β value when the size of node is 1x1x1

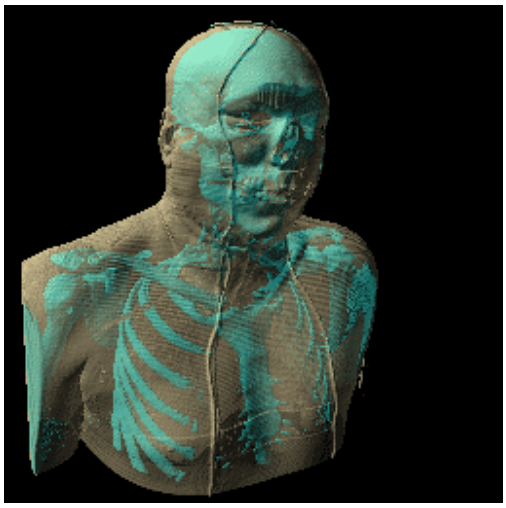


(a) **(b)**

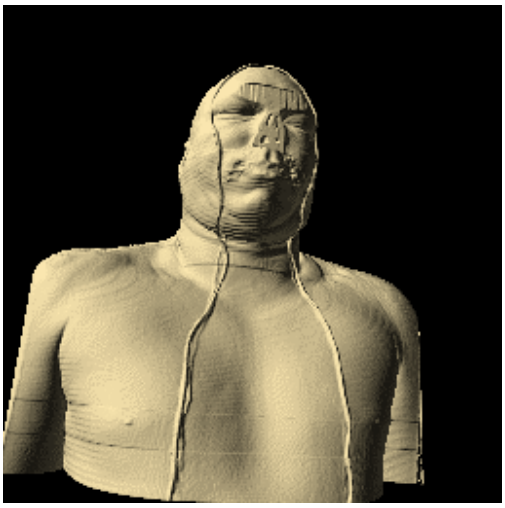
Figure 6. The multi-level isosurface ray casting algorithm

(a) A 2D slice image

(b) A magnified view of the rectangle in (a). The image has two isosurfaces, ρ_1 and ρ_2. l is the level of the ray.



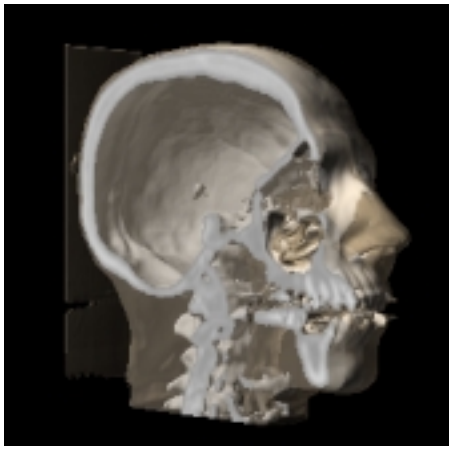
(a) two isosurfaces(skin+bone)



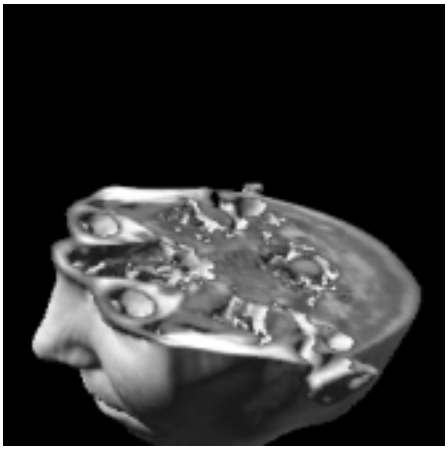
(b) one isosurface(skin)

(c) one isosurface(bone)

Figure 7. Rendered Images from Visible Man Dataset (512x512x512, 8 bits per pixel). The resolution of the images are 256x256.



(a) Bighead – 2 isosurfaces



(b) Brain – 1 isosurface

Figure 8. The sample rendered images. Bighead CT (256x256x225) and Brain MR (128x128x84) datasets are used.

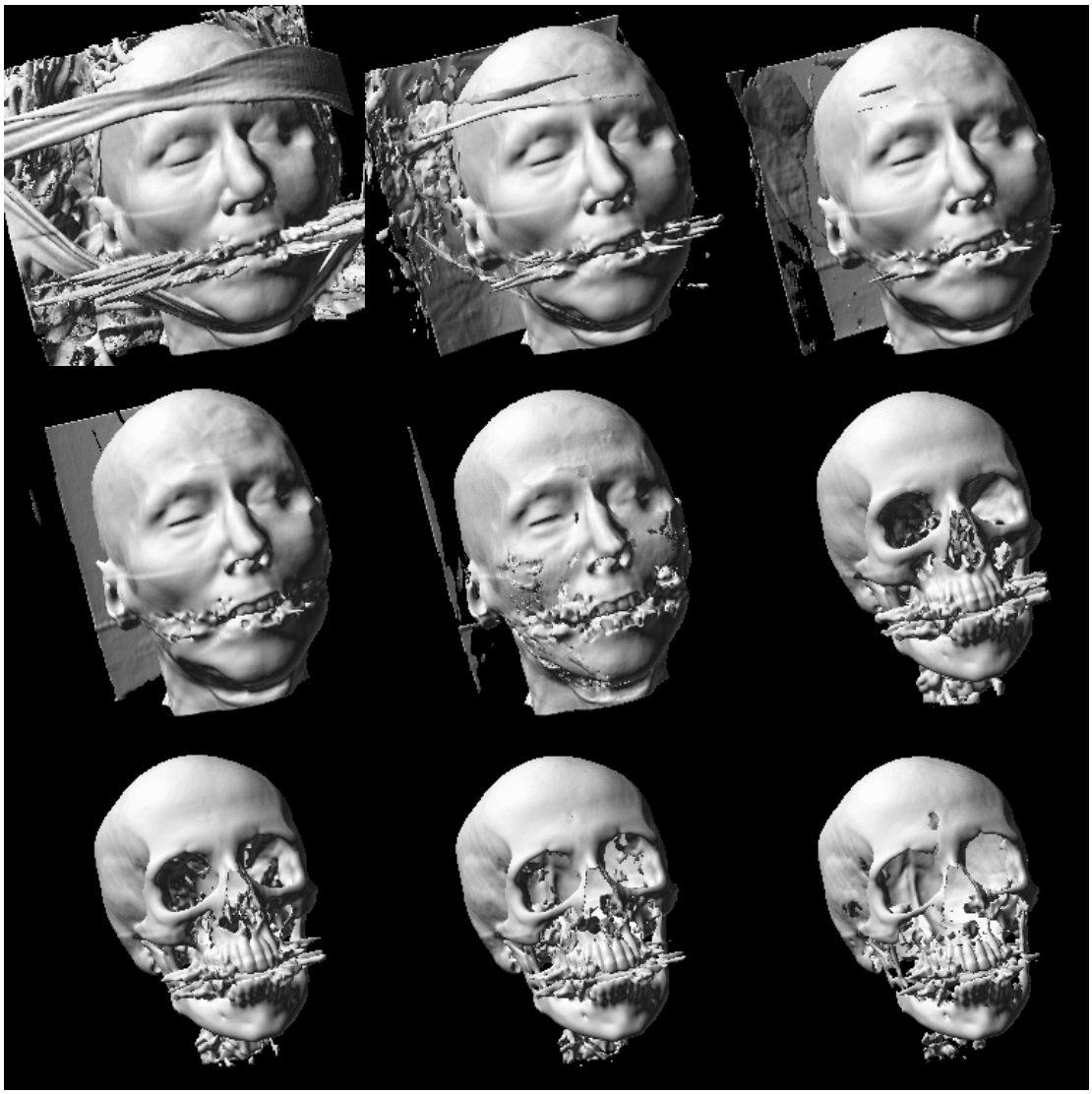


Figure 9. The images from the different isosurfaces (CT Head, 256x256x225, 8 bits).

Table 1. Rendering Time for Figure 7.

The rendering time of our algorithm is 1.8-3.8 times faster than PARC algorithm

(unit : seconds)

		Figure 7 (a)		Figure 7 (b)		Figure 7 (c)	
		SGI	PC	SGI	PC	SGI	PC
Standard isosurface ray casting with no optimization		26.6	21.19	23.9	18.40	27.3	21.24
PARC(Polygon Assisted Ray Casting)		5.25	4.15	1.65	1.20	2.2	1.63
Image-space Bounding Surface Only	Total Rendering Time	2.82	2.92	0.93	0.76	0.90	0.59
	Polygon Rendering	0.04	0.13	0.04	0.15	0.02	0.10
	Ray Casting	2.58	2.67	0.69	0.49	0.74	0.41
	Bounding Surface Generation	0.20	0.12	0.20	0.12	0.14	0.08
All Optimization Techniques	**Total Rendering Time**	**1.60**	**1.53**	**0.74**	**0.66**	**0.58**	**0.51**
	Polygon Rendering	0.05	0.15	0.04	0.15	0.02	0.10
	Ray Casting	1.28	1.26	0.50	0.39	0.42	0.33
	Bounding Surface Generation	0.20	0.12	0.20	0.12	0.14	0.08
Speed-up Factor	No Optimization/ All Optimization Techniques	**16.6**	**13.8**	**32.3**	**27.9**	**47.1**	**41.6**
	PARC/ All Optimization Techniques	**3.3**	**2.7**	**2.2**	**1.8**	**3.8**	**3.2**