1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

AN IMPROVED SPLITTING-BOX ALGORITHM FOR VOLUME VISUALIZATION *

Shi-Nine Yang, Tian-Sheng Wu and Shann-Horng Chang

Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan 300, R.O.C.
Email:snyang@cs.nthu.edu.tw

ABSTRACT

This paper presents an efficiency-enhanced technique for generating isosurfaces in volume data. By incorporating a hierarchical spatial data structure into splitting-box algorithm, many unnecessary density checking operation can be avoided. Empirical tests are given to show that the performance of the proposed algorithm is better than both the conventional marching-cubes algorithm and the original splitting-boxes algorithm in the aspects of computation time and number of triangle patches generated.

1. INTRODUCTION

Scientific visualization has been studied extensively in recent years. One of the most interesting and fast growing techniques in scientific visualization is the volume visualization. The purpose of volume visualization is to project a multi-dimensional data set onto a two-dimensional image plane for gaining an understanding of the structure contained in the volume data. There are two major approaches to visualize a set of multi-dimensional volume data, namely, the ***direct volume rendering*** (DVR) and the ***surface fitting*** (SF) method.

Direct Volume Rendering algorithms include approaches such as ray-casting [1] [2] [3] [4] [5], integration methods, splatting [6], and V-buffer rendering [7]. The latter two methods are sometimes called ***projection methods*** [8] [9]. A DVR algorithm is characterized by mapping volume primitives directly into screen space without using geometric primitives as an intermediate representation. DVR methods are especially appropriate for creating images from data sets containing amorphous feature like clouds, fluids, and gases.

* This research is supported by National Science Council, Republic of China under Grant NSC 85-2221-E-001-030.

A Surface Fitting Algorithm is characterized by approximating contour surface in the volume with geometric primitives such as polygons or patches. It is also called feature-extraction or iso-surface method. The SF approach includes contour-connecting [10], marching-cubes [11], marching-tetrahedra, dividing-cubes [12], splitting-box [13], and others. After all patches have been extracted, rendering hardware can be used to render the surface in real time. Therefore it is suitable for applications involving frequent change of viewing point. However, changing threshold in SF is time-consuming because it requires the revisit of all cells to extract a new set of surface primitives.

The widely adopted method of surface fitting is the marching-cubes method [11] [14] [15]. It examines eight vertices of a voxel. A vertex is assigned the color black (white) if its scalar value is greater (less) than the given threshold. A bit is assigned to record the color of every vertex. Eight bits form a byte-index corresponding to the eight vertices of a voxel. There are as many as 256 configurations for a cube, however they can be classified into only 15 basic cases [11]. A look-up table is pre-calculated for the triangulation of configurations.

The marching-cubes algorithm traverses every cube and calculates its byte index. Then, it looks up the table according to the index for the triangulation of each cube. The isosurface intersects the edges of a cube at the triangle vertices that are obtained from a list of edges in the table via linear interpolation of the densities of edge vertices. The unit normal of each triangle vertex is calculated using central difference and linear interpolation. Conventional graphics-rendering algorithms implemented in either software or hardware can display the resulting representation with triangle vertices and normals.

There are two major problems with marching-cubes method. First, the topological ambiguity may occur when a cube has adjacent vertices with different colors, but diagonal vertices are in the same color [16]. Some solutions are available to resolve this problem [17] [18]. Second, in case of large data volume, the time for

traversing the data set is considerable and the number of generated triangles is numerous. In order to speed up the rendering process, many reports [13] [19] [20] [21] [22] [23] [24] [25] have proposed improved algorithms to reduce the amount of data created.

To speed up iso-surfaces generation, Jane Wilhelms and Allen Van Gelder proposed an octree method [21]. They proposed a variant of octree-like data structure called BONO tree. It can improve the performance of marching-cubes algorithm by reducing the amount of data traversed. In 1993, Heinrich Mueller and Michael Stark presented an algorithm called " Splitting-box" [13], which reduces considerable number of triangles generated. Instead of directly computing the triangles on each cube element, they identify the contour chains of the isosurface boundary in a top down fashion and adapt the size of the polygonal chain according to its shape. It is a good approach to reduce the number of surface primitives, however in their algorithm there are many wasted computations for empty regions. In this study, we incorporate a min-max bisection-tree data structure to the subdivision process of the splitting-box algorithm to avoid these unnecessary checking operations during the generation of contour chains. Empirical tests are given to demonstrate that this simple enhancement results in a significant improvement in computation time.

Section 2 describes briefly the splitting-box algorithm and the notion of min-max bisection-tree. Then an improved algorithm is proposed. In section 3, several empirical tests are presented to demonstrate the performance of the proposed algorithm. It shows that our algorithm outperforms both the pure splitting-boxes algorithm and the marching-cubes algorithm in computation time.

2. ALGORITHMS

First, we briefly introduce the splitting-box algorithm and the min-max bisection tree (MMBT) structure. Then we show how to integrate these two notions to form our improved algorithm.

Let V be a given volume data which consists of a finite regular 3D grids of scalar values and T be the threshold. The splitting-boxes algorithm finds a linear approximation of the isosurface of V with threshold T . Several terms that will be used later are introduced as follows. A box is a rectangular sub-volume of V . The edges and the faces of a box are defined in usual sense, therefore each box has 12 edges and 6 faces. The *length* of an edge is the number of grid points on it. An edge is typed an MC edge if there is at most one transition of colors among its grid points. A face is MC if its four edges are all MC edges. Also, a box is MC if its 12 edges are all MC edges. The splitting-box algorithm first bisects the given box at its longest edge. Then the sub-box is checked for MC. If it is not MC, the bisection process is repeated recursively until the 2x2x2 sub-boxes are generated. For an MC box, contour chains are generated and then triangulated to obtain its surface primitives.

One of the main drawbacks of the splitting-box algorithm is that the "quality-checking" operations must be performed down to the unit box (2x2x2 box) size. For a given threshold value T, a box is found to be *empty* only if the values of its vertices are all greater than or less than T. Figure 2.1 shows an example that a 3x3x3 box must be split further to avoid the loss of possible surface element. There are totally 14 sub-boxes generated by the splitting-box procedure from 3x3x3 box size to 2x2x2 box size. In other words, the "quality-checking" operations for the contour chains in a 3x3x3 box must be processed 14 times. Even for an *empty* 3x3x3 box without any surface element, the 14 times "quality-checking" operations are still needed to guarantee that nothing is in the box. These time wasting procedures slow down the splitting-box algorithm. To cope with this problem, we incorporate a min-max bisection tree into the algorithm to avoid unnecessary subdivisions.

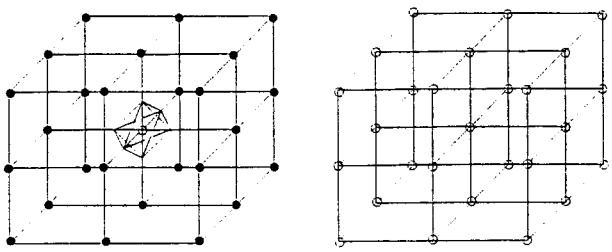

Figure 2.1:Two cases of 3x3x3 boxes

According to the notion of [21], if there is a min-max hierarchical data structure, then those boxes containing no threshold value T require no further subdivision. In order to adapt to the greedy-partition criterion of the splitting-box algorithm, a hierarchical data structure called min-max bisection tree (MMBT) is introduced as follows. A min-max bisection tree is a hierarchical data structure based on decomposition of volume space. A volume space is recursively subdivided according to the MC criterion in the splitting-box algorithm. Each node of the tree corresponds to a box in the volume space. In the min-max bisection tree, two extra fields, namely, the maximum value and minimum value in the box are added to each node. For completeness, the construction algorithm of MMBT is given in the following.

```
PROCEDURE subdivide(box,subbox1,subbox2)
BEGIN
  Subdivide the box at its longest edge into two subboxes;
  Record subdividing direction into box-node;
END

PROCEDURE MMBT(level,box)
BEGIN
   IF( level is 2x2x2 leaf level )
     BEGIN
        Compute the min,max value of box;
        Record the min,max values into box-node;
        Return box-node;
     END
   ELSE
     BEGIN
        subdivide (box , box1, box2);
        MMBT( level+1, box1);
        MMBT( level+1, box2);
        Get summary min,max of box from box1, box2;
        Record the min,max values into box-node;
        Return box-node;
     END
END
```

MMBT construction algorithm

The root of MMBT is the given volume V itself. There are two important fields in each node, namely, the minimum and the maximum, which respectively record the minimum and the maximum density values of the box. The procedure subdivide() is to subdivide the volume data at the edges of the same direction sequences as that of the splitting-box algorithm. Our improved splitting-boxes algorithm first call the procedure MMBT to build the min-max bisection tree (MMBT) which is used to summaries the scope of density values of the grid points in each box. The following process is similar to that of the splitting-boxes algorithm except that a box is subdivided not only because it is not MC but also because the threshold value lies between the minimum and maximum values of the corresponding box-node in the MMBT.

Notice that in the conventional min-max octree [21], the subdivision is performed cyclically along x, y, and z-axis. However, this subdivision rule does not fit the greedy subdivision of splitting-box algorithm. Therefore our subdivision strategy is based on subdividing the longest edge to adapt to the splitting-box algorithm. Accordingly, only a part of the splitting-box algorithm needs change.

The worst case time complexity of MMBT generation is introduced as follows. A box can be bounded by a cube whose edge length is equal to the longest edge length of the original box. For simplicity, the longest edge length of the initial box is $n = 2^m + 1$. Then the longest edge is split into two edges of length $2^{m-1}+1$ if a bisection is applied. Two nodes corresponding to sub-boxes are generated and two comparison operations are performed to compute the minimum and maximum values. The second bisection yields four sub-boxes from the two boxes generated by the first bisection. Then eight sub-boxes whose longest edge length is not greater than $2^{m-1}+1$ are generated by the third bisection.

Let T(n) be the time required for MMBT generation from the initial cube, N(m) be the time needed for an cubic box of egde length 2^m+1 , a leaf node of MMBT is mapped to a box of size 2x2x2, and the computation time for generating a leaf node is N(1). Then we have the following.

$$T(n) = T(2^m + 1) = N(m)$$

$$N(1)=14$$

$$N(m) = (2+4+8) + 8 \cdot N(m-1)$$
$$= 14 + 8 \cdot N(m-1)$$

$$N(m) = 14 + 8 \cdot N(m-1)$$
$$= 14 + 8 \cdot (14 + 8 \cdot N(m-2))$$
$$= \cdots$$
$$= 14 \cdot \sum_{k=0}^{m-1} 8^k + 8^{m-1} \cdot N(1)$$
$$= 14 \cdot (\frac{8^m - 1}{8-1}) + 8^{m-1} \cdot 14$$
$$= O(2^m + 1)^3$$
$$= O(n^3)$$

The running time for generating the MMBT is linear with respect to the size of input data.

3. IMPLEMENTATION AND RESULTS

In order to compare the performance of the following algorithms, namely, the marching-cubes algorithm, the splitting-box algorithm, and our improved algorithm, we implemented all of them on an SGI workstation. By comparing both the numbers of triangles generated and the total computation time, it shows that our improved algorithm is better than the other two algorithms in both aspects.

The first example is a density function defined by the distance function

$$f(x,y,z) = \sqrt{(x-16)^2 + (y-16)^2 + (z-16)^2}$$

The isosurface of threshold T is a sphere with center (16,16,16) and radius T. A hidden line representation of the results of both marching-cubes and improved splitting-box approaches are shown in Figure 3.1 and Figure 3.2

respectively. Table 3.1 shows the numbers of triangles and their running time. Figure 3.3 shows the relative frequencies of the *empty* boxes those do not need further subdivisions.

The second example is a medical data set, the CThead. It consists of the cross-section images of the human head obtained by computer tomography. The data size is 256x256x92, and the range of scalar value is from 0 to 255. We chose the threshold 80 for the density of bone. Figure 3.4 and Figure 3.5 show the results obtained with the marching-cubes algorithm and the proposed algorithm respectively. Table 3.2 shows the numbers of triangles and their running time. Figure 3.6 shows the relative frequencies of empty boxes. The labeles are the sizes of boxes and the numbers of empty boxes.

The data set of the third example is the same as that of the second example. We chose another threshold 50 for the density of skin. Figure 3.7 and Figure 3.8 show the results obtained with the marching-cubes algorithm and the proposed algorithm respectively. Table 3.3 shows the numbers of triangles and their running time. Figure 3.9 shows the relative frequencies of empty boxes.

The fourth example is also a medical data set, an MRbrain. It consists of the cross-section images of a human head obtained by magnetic resonance. The data size is 256x256x109, and the range of scalar value is from 0 to 255. We chose the threshold 50 for presenting the appearance. Figure 3.10 and Figure 3.11 show the results obtained with the marching-cubes algorithm and the proposed algorithm respectively. Table 3.4 shows the numbers of triangles and their running time. Figure 3.12 shows the relative frequencies of empty boxes. Apparently, the proposed algorithm avoids a certain number of unnecessary subdivisions of contour chains.

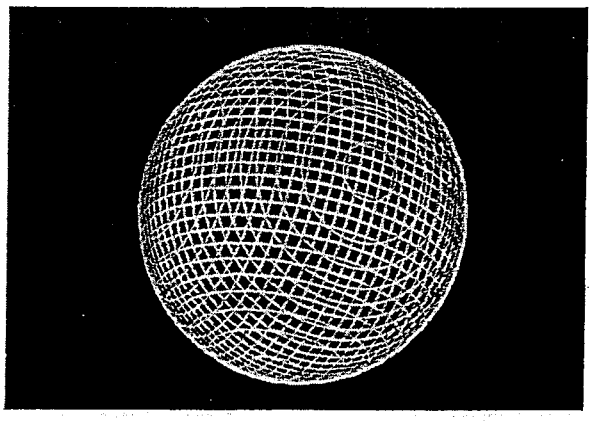

Figure 3.1: Sphere with threshold 80 applying the marching-cubes algorithm

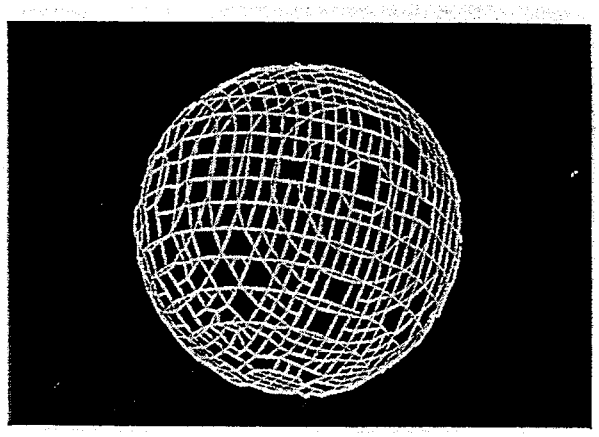

Figure 3.2: Sphere with threshold 80 applying the proposed algorithm

Sphere 33x33x33	Marching-cubes		Splitting-box		MMBT	
	#triangles	CPU time (sec)	#triangles	CPU time (sec)	#triangles	CPU time (sec)
MMBT Creadon	6824	——	3484 (51.05%)	——	3484 (51.05%)	0.21
Surface Finding		2.25		4.41		1.54
Total Extraction		2.25		4.41		1.75

Table 3.1:Experimental results (Sphere)

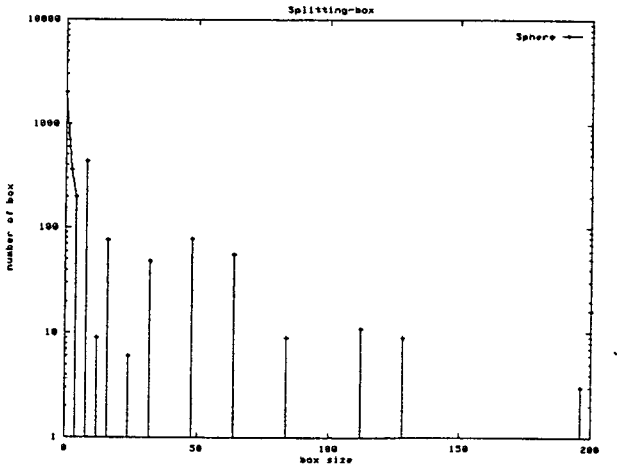

Figure 3.3:Relative frequencies of *empty* boxes in Sphere with threshold 80

Figure 3.4: CThead with threshold 80 applying the marching-cubes algorithm

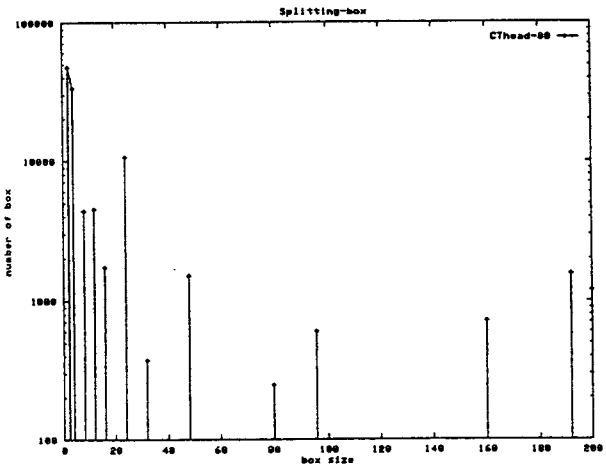

Figure 3.6:Relative frequencies of *empty* boxes in CThead with threshold 80

Figure 3.5: CThead with threshold 80 applying the proposed algorithm

Figure 3.7: CThead with threshold 50 applying the marching-cubes algorithm

CThead 256x256x92 density = 80	Marching-cubes		Splitting-box		MMBT	
	#triangles	CPU time (sec)	#triangles	CPU time (sec)	#triangles	CPU time (sec)
MMBT Creation	634280	———	374131 (58.98 %)	———	374131 (58.98 %)	30.34
Surface Finding		455.68		727.64		128.57
Total Extraction		455.68		727.64		158.91

Table 3.2:Experimental results (CThead1)

Figure 3.8:CThead with threshold 50 applying the proposed algorithm

CThead 256x256x92 density=50	Marching-cubes		Splitting-box		MMBT	
	#triangles	CPU time (sec)	#triangles	CPU time (sec)	#triangles	CPU time (sec)
MMBT Creation	389160	———	218190 (56.06 %)	———	218190 (56.06 %)	30.53
Surface Finding		367.94		702.19		79.33
Total Extraction		367.94		702.19		109.86

Table 3.3:Experimental results (CThead2)

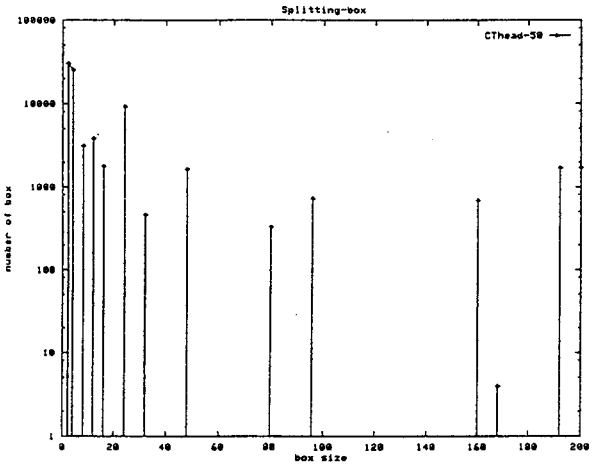

Figure 3.9:Relative frequencies of *empty* boxes in CThead with threshold 50

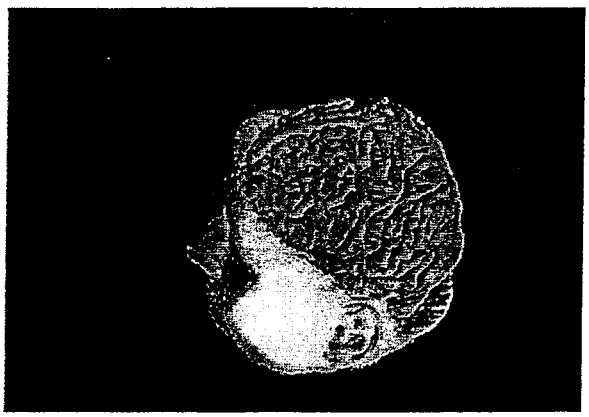

Figure 3.10: MRbrain with threshold 50 applying the marching-cubes algorithm

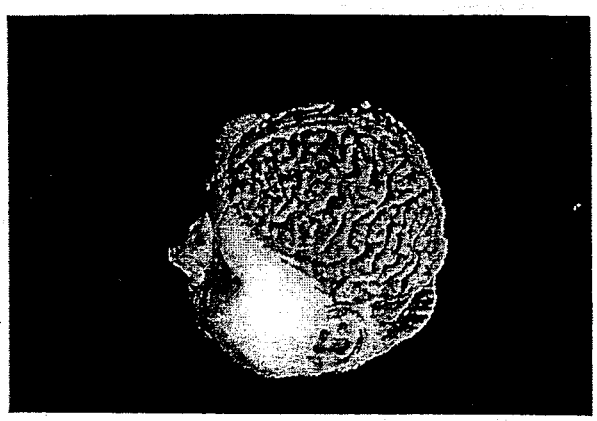

Figure 3.11:MRbrain with threshold 50 applying the proposed algorithm

MRbrain 256x256x109	Marching-cubes		Splitting-box		MMBT	
	#triangles	CPU time (sec)	#triangles	CPU time (sec)	#triangles	CPU time (sec)
MMBT Creation	1371482	———	1038849 (75.74 %)	———	1038849 (75.74 %)	36.09
Surface Finding		553.24		917.96		241.73
Total Extraction		553.24		917.96		277.82

Table 3.4:Experimental results (MRbrain)

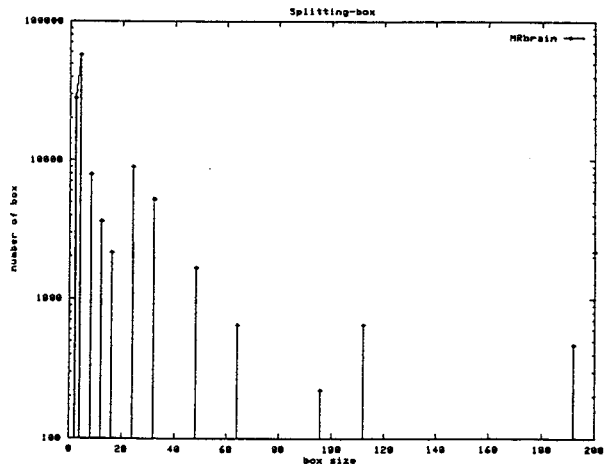

Figure 3.12: Relative frequencies of *empty* boxes in MRbrain with threshold 50

4. DISCUSSIONS AND CONCLUSION

Table 3 shows that the number of triangles generated by the splitting-box algorithm can be reduced from 1/2 to 1/3 of that generated by the marching-cubes algorithm. The running time of the splitting-box algorithm is about 1.5 to 2.0 times of that required by marching-cubes algorithm.

In the proposed (MMBT) algorithm, the number of the contour chains is the same as that in the pure splitting-box

algorithm, however the running time, even including the MMBT construction time, is about 16 % to 40 % of the original splitting-box algorithm. Moreover, the running time, including the time for MMBT creation, is about 30 % to 77 % of the running time of the marching-cube algorithm. If we regard the min-max bisection tree construction time as the preprocessing time, the actual polygon creation time is even less.

The marching-cubes algorithm has the disadvantage that the number of triangles generated is considerably large. The splitting-box algorithm is a good approach to reduce the number of surface primitives but have two problems. First, there still is much computation time wasted for empty region in volume data. Second, from the splitting-box algorithm, we know that it must generate two times contour chains. These time-wasting procedures make the splitting-box algorithm even slower than the original marching-cubes algorithm. It is strongly required for an exact approximation of large contour chains generated from large boxes. By embedding the min-max bisection tree into splitting-box algorithm, we obtain an improved isosurface generation algorithm that is better than conventional marching-cubes algorithm both in number of triangles generated and computation time. Finally we give table 4.1 to show that the BONO tree does not perform better than the min-max bisection tree as long as the splitting-box algorithm is concerned.

DATA	Splitting-box		S-b +BONO		MMBT	
	#triangles	#MC-check	#triangles	#MC-check	#triangles	#MC-check
Sphere	3484	131068	3484	24444 (18. 64 %)	3484	24444 (18. 64 %)
CThead1	374131	23669096	402858	2104488 (8. 89 %)	374131	2063308 (8. 71 %)
Cthead2	218191	23669096	228574	1318488 (5. 57 %)	218191	1307108 (5. 52 %)
MRbain	1038849	186679822	1056273	38835580 (20. 80 %)	1038849	3812904 (20. 42 %)

Table 4.1: The comparison between BONO and our approach

References

[1] Duff, T. and Porter, T., "Compositing digital images," *Computer Graphics(SIGGRAPH'84 Proceddings)*, Volume 18, pp.253-259, July 1984.

[2] Levoy, M. , "Display of Surface from Volume Data," *IEEE Computer Graphics and Applications*, Volume 8, Number 3, pp.29-37, March 1988.

[3] Levoy, M. , "Volume Rendering, A Hybrid Ray Tracer for Rendering Polygon and Volume Data," *IEEE Computer Graphics and Applications*, Volume 10, Number 2, pp.33-40, March 1990.

[4] Levoy, M. , "Efficient Ray Tracing of Volume Data," *ACM Transactions on Graphics*, Volume 9, Number 3, pp.245-261, July 1990.

[5] Elvins, T. T., "A Survey of Algorithm for Volume Visualization," *Computer Graphics* , Volume 26, umber 3, August 1992.

[6] Westtover, L., "Footprint Evaluation for Volume Rendering," *Computer Graphics*, Volume 24, Number 4, pp.367-376, August 1990.

[7] Upson, C. and Keeler, M., "V-Buffer: Visible Volume Rendering," *Computer Graphics*, Volume 24, Number 4, pp.59-64, August 1988.

[8] Wilhelms, J., "Decisions in Volume Rendering," *State of the Art in Volume Visualization*, ACM SIGGRAPH'91 Course Notes, Course Number 8, ACM Press, I.1-I.11, August 1991.

[9] Wilhelms, J. and Van Gelder, A., "A Coherent Projection Approach for Direct Volume Rendering," *Computer Graphics*, Volume 25, Number 4, pp.275-284, August 1991.

[10] Christiansen, H.N., "Conversion of Contour Line," *Siggraph* , Volume 12, Number 3, pp.187-192, 1978.

[11] Lorensen, W.E. , "Marching-cubes: A high resolution 3D surface construction algorithm," *ACM Siggraph Proceedings*, Volume 22, number 4, pp.163-169, July 1987.

[12] Cline, H.E., Lorenson, W.E., Ludke, S., Crawford, C.R. and Teeter, B.C., "Two Algorithms for Three-dimensional Reconstruction of Tomograms," *Medical Physics*, Volume 15, Number 3, pp.320-327, May/June 1988.

[13] Heinrich, M. and Michael, S. "Adaptive generation of surface in volume data," *The Visual Computer*, pp.182-199, September 1993.

[14] Lorenson, W.E., Cline, H.E., Ludke S., Crawford, C.R., and Teeter, B.C., "Two algorithms for the three-dimensional reconstruction of tomograms," *Medical Physics*, Volume 15, Number 3, pp.320-327, May 1988.

[15] Wyvill, G., McPheeters, C. and Wyvill, B., "Data Structure for Soft Objects," *The Visual Computer*, Volume 2, Number 4, pp.227-234, August 1986.

[16] Dust, M.J., "Letter: Additional Reference to Marching-cubes," *Computer Graphics*, Volume22, number 4, pp.65-74, July 1988.

[17] Gallagher, R.S. and Negtegaal, J.C., "An Efficient 3D Visualization Technique for Finite Element Models and Other Coarse Volume," *Computer Graphics*, Volume 23, Number 3, pp.185-192, 1989.

[18] Wilhelms, J. and Van Gelder, A. , "Topological ambiguities in isosurface generation," Volume 24, number 5, Extended abstract in *ACM Computer Graphics*, volume 24, number 5, pp.79-86, December 1990.

[19] Hoppe, H. DeRose T. and Duchamp, T., "Mesh optimization," *Computer Graphics*, pp.19-26, August 1993.

[20] Moore, D. and Warren, J., "Mesh displacement: An improved contouring method for trivariate data," Research Report TR91-166, IBM Research Division, September 1991.

[21] Wilhelms, J. and Van Gelder, A., "Octrees for Faster Isosurface Generation," *ACM Transactions on Graphics*, Volume 11, Number 3, pp.201-227, July 1992.

[22] Schroeder, W. J.,Zarge, J. A., and Lorensen, W. E., "Decimation of triangle meshes," *Computer Graphics*, Volume 26, Number 2, pp.65-70, July 1992.

[23] Turk, G., "Re-tiling polygonal surface," *Computer Graphics*, Volume 26, Number 2, pp.55-64, July 1992.

[24] Jung-Hong Chuang and Woan-Chiaun, "Efficient generation of isosurfaces in volume rendering," *Computer & Graphics*, Vol. 19, No. 6, pp.805-813, 1995.

[25] Jianping Li, Pan Agathoklis, "An efficiency enhanced isosurface generation algorithm for volume visualization," *The Visual Computer*, No. 13, pp.391-400, 1997.