Adaptive Marching Cubes

Renben Shu, Chen Zhou and Mohan S. Kankanhalli
Institute of Systems Science,
National University of Singapore
Kent Ridge, Singapore 0511

Abstract

The Marching Cubes algorithm (MC) is a powerful surface rendering technique which can produce very high quality images. However, it is not suitable for interactive manipulation of the 3D surfaces constructed from high resolution volume data sets in terms of both space and time. In this paper, we present an adaptive version of MC called *Adaptive Marching Cubes* (AMC). It significantly reduces the number of triangles representing the surface by adapting the size of the triangles to the shape of the surface. This improves the performance of the manipulation of the 3D surfaces. A typical example with the volume data set of size $256 \times 256 \times 113$ shows that the number of triangles is reduced by 55%. The quality of images produced by AMC is similar to that of MC. One of the fundamental problems encountered with adaptive algorithms is the *crack problem*. Cracks may be created between two neighboring cubes processed with different levels of subdivision. We solve the crack problem by patching the cracks using polygons of the same shape as those of the cracks. We propose a simple but complete method by first abstracting 22 basic configurations of arbitrary sized cracks and then reducing the handling of these configurations to a simple rule. It requires only $\boldsymbol{O}(n^2)$ working memory for a $n \times n \times n$ volume data set.

Key words: Surface rendering, Surface construction, Interactive manipulation of 3D surface

1.0 Introduction

Surface rendering is a means of extracting meaningful and intuitive information from 3D data sets. This is achieved by converting the volume data into a surface representation using an extraction step and then using conventional computer graphics techniques to render the surface. The surface extraction process is very important and several techniques have been developed (Kaufman 1990):

- A method using a cuberille model, in which the rectilinear faces of all nontransparent voxels are used to form a polygon (square) mesh (Chen et al. 1985, Herman and Liu 1979).

- A surface tracking algorithm that creates a surface from exterior voxel faces by starting from a seed on that surface and using a connectivity rule to form the rest of the surface (Artzy et al. 1981, Gordon and Udupa 1989, Sobierajski et al. 1993, Trivedi et al. 1986, Udupa and Hung 1990).

- The Dividing Cubes algorithm that generates a cloud of points (Cline et al.1988).
- The Marching Cubes algorithm that creates a fine triangle mesh (Lorensen and Cline 1987).

Among them, the Marching Cubes algorithm (MC) is able to generate very high quality images by generating a set of triangles which closely approximates a surface of interest. A lot of effort has been devoted to improve the marching cubes algorithm in one way or another. The original MC proposed in (Lorensen and Cline 1987) suffers from the hole problem, which is caused by ambiguities in the method used to approximate the surface. This has been dealt with in Baker 1989, Cline and Lorensen 1988, Durst 1988, Fuchs et al. 1990, Nielson and Hamann 1991, Wilhelms and Van Gelder 1990. Another problem with the original MC is speed, which has been improved by using the octree to reduce the number of cubes traversed (Wilhelms and Van Gelder 1992). However, the result is still not adequate for interactive manipulation of 3D surfaces constructed from high resolution data sets.

One solution is to store all the triangles representing the surface so that when the surface is manipulated e.g. rotated, there is no need to regenerate the surface. But this approach faces both space and time problems because of the large number of triangles generated by MC. For example, a 3D surface constructed from a typical high resolution volume data set of $256 \times 256 \times 113$ requires 718 964 triangles, and these triangles need 25 MB memory for storage (each triangle requires 36 bytes). In addition, the large number of triangles implies that more time is needed for the rendering. Because of these problems, it is difficult to use MC for interactive applications in 3D visualization. This is a serious problem in an application area such as medical visualization.

Recently, Turk (1992) uses a re-triangulation technique that introduces new points onto a polygonal mesh, and then discards the old points to create a new mesh to reduce the number of triangles representing the given surface. Schroeder et al. (1992) use an approach of vertex removal and local re-triangulation for simplifying polygonal models. They remove vertices that are within a prespecified tolerance of a plane that approximates the surface near vertex. Their method also identifies sharp edges and sharp corners and makes sure such features are retained in order to better represent the original data. Both methods post-process the triangulated surface generated by MC.

More recently, Muller and Stark (1993) propose a method which adaptively generates the surface, i.e. adapts the size of triangles to the shape of surface. The problem with their method is that certain big objects could be ignored. This is because the method does not check the value of any sample point within a box or subbox. Their method also suffers from the crack problem. Cracks may

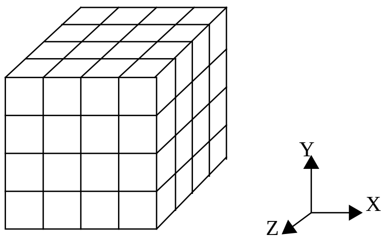


Figure 1. Cuberille grid data.

be created between two neighboring processing units with different levels of subdivision. The crack problem has been recognized as a very difficult problem for all adaptive algorithms such as (Clay and Moreton 1988). Although Muller and Stark solve the crack problem, their method is at the expense of losing certain features of objects, sometimes even big features. Their method deals with the crack problem by processing the common faces of neighboring boxes in the same way, but independently without storing additional context information. It does this by stretching a curved polyline on a common MC face to a straight polyline. This method never checks under what circumstance cracks may occur. This would lead to unnecessary modification of intersection points on the common face which could result in loss of big features of the objects. In this paper, we also propose an adaptive approach, but our approach is quite different from theirs in the way of dividing up the volume data set and in the patching of cracks. More importantly, our method is free from the problem underlying the Muller and Stark algorithm.

The remainder of this paper will be organized as follows: Section 2 briefly describes MC.and presents the basic idea of AMC. Then section 3 discusses the crack problem caused by adaptive surface representation and its solution. After that section 4 gives the data structures for crack patching and the results of our experiments on AMC, followed by section 5 which presents the conclusions.

2.0 Adaptive Marching Cubes

2.1 Background

We briefly summarize the MC algorithm here in order to simplify the presentation of our algorithm. Marching cubes is a surface rendering algorithm that converts a volumetric data set into a polygonal isovalued (user-specified) surface consisting of triangles whose vertices are on the edges of the voxels (unit cubes) of the cuberille grid (see Figure 1). The method processes one voxel at a time. The values of grid points and linear interpolation are used to determine where the isovalued surface intersects an edge of a voxel. How the intersection points are assembled into triangles depends on the number and configuration of the grid points with values above or below the threshold used to compute the isovalued surface. The various configuration are shown in Figure 2,

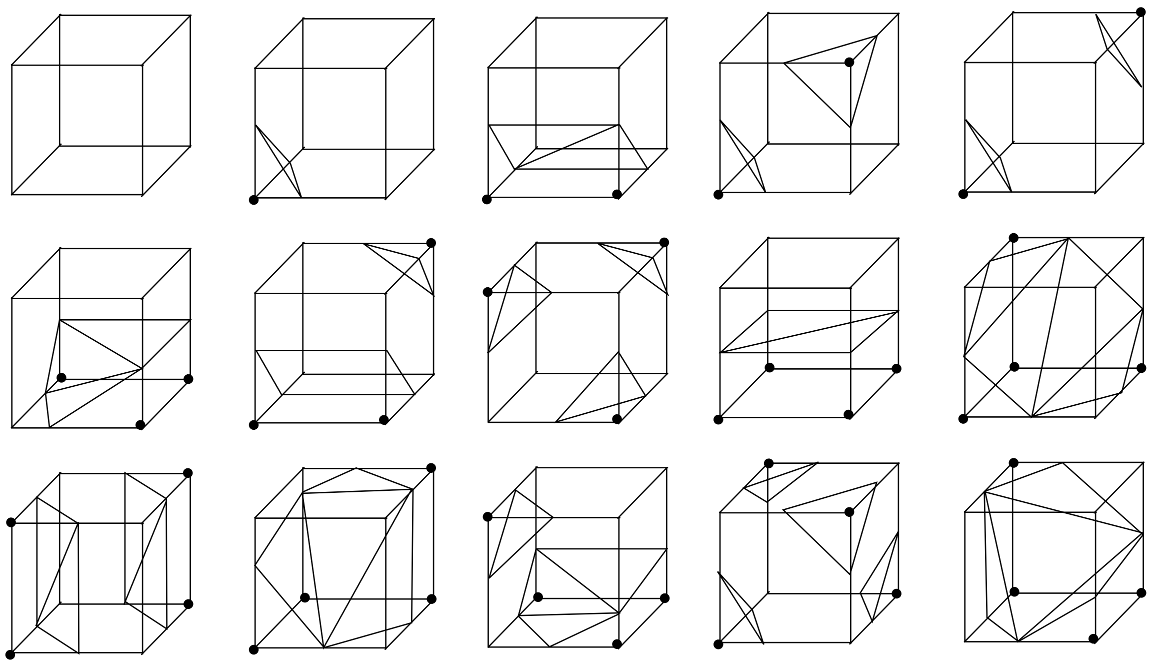


Figure 2. Configurations of triangulated cubes.

where a grid point that is marked indicates its value being above the threshold. While there are $2^8 = 256$ possible configurations, there are only 15 shown in Figure 2. This is because some configurations are equivalent with respect to certain operations. First, the number can be reduced to 128 by assuming the two configurations are equivalent if marked grid points and unmarked grid points are switched.This means that we only have to consider cases where there are four or fewer marked grid points.Further reduction to 15 cases shown is possible by equivalence due to rotations.

2.2 Basic AMC Strategy

Assume that the volumetric data set size is $N_x \times N_y \times N_z$,where $N_x = 2^i, N_y = 2^j, N_z = 2^k$ (it is easy to generalize AMC for an arbitrary size data set). As we mentioned in section 1.0, the basic strategy in AMC is to adjust the shape of the approximating surface based on the curvature of the actual surface within a cube. Initially, we partition the volumetric data set into cubes with equal size of 2^m, which we call initial cubes, where $m \le min\,(i, j, k)$, and then use the MC surface configuration approach described in (Lorensen and Cline 1987) to triangulate the cubes, considering only the values of the 8 vertices of cubes. Then we recursively partition these cubes into

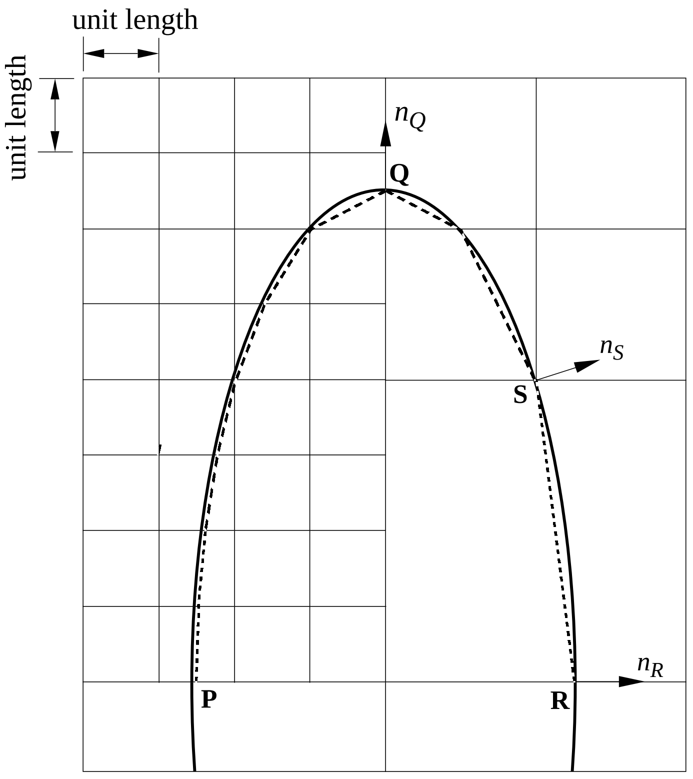


Figure 3. How the AMC algorithm works in 2D.

smaller and smaller cubes based on the smoothness of the surface within these cubes. For each cube, if the actual surface inside of it is flat enough, then the triangles of the MC surface configurations are used to approximate it. Conversely, if the actual surface has high curvature, the cube will be partitioned into 8 subcubes and the process will be repeated until all surfaces in the subcubes are flat enough to be approximated with the MC configurations or the length of the subcube sides is one. Figure 3 illustrates the basic adaptive idea for a 2D curve, where part of an ellipse *PQR* is to be approximated to a set of straight lines. If the smallest squares containing the ellipse are of unit length, then the 2D analog of MC would require 7 straight lines to approximate it. This is shown by the dashed lines along *PQ*. However if AMC is used, only 3 straight lines are needed as shown along *QR*.

Figure 3 shows that the curve segment *RS* has normals n_R and n_S that are not too different from each other. This means that the curvature of *RS* is small enough that we can approximate it with a straight line. However, for the segment *QS*, its normals n_Q and n_S differ greatly which means that the segment has a curvature that is too large to be approximated by a single straight line. Hence the big square with the side length of 4 containing ellipse *QS* is partitioned into 4 subsquares and two lines are used to approximate it.

This idea can be extended easily to the 3D case which we propose in this paper for AMC. Unfortunately, the problem is not so simple since AMC suffers from the crack problem encountered by all adaptive algorithms.

We now present the pseudocode of the AMC algorithm.

```
1  divide up volume data set into initial cubes of equal size
2  for each initial cube
3      call process_cube(initial cube)
4      if cracks exist then
5          patch cracks /* explained in Section 3.3 */
6      end if
7  end for


8  procedure process_cube(cube)
9     if (cube contains a surface) then
10       find intersections of the surface and cube edges ;
11       calculate intersection normals ;
12       if ( ( cube is of unit size ) or
             ( any triangles with normals $n_0$, $n_1$ and $n_2$, ARCCOS($\vec{n}_i \bullet \vec{n}_{(i+1) mod 3}$)$<\delta$ for any $i \in \{0, 1, 2\}$ ) )
13          then
14             output triangles ;
15             store information for crack patching ;
```

```
16      else
17         divide cube into 8 subcubes ;
18         for each subcube
19            call process_cube(subcube) ;
20         end for
21       end if
22      else
23         store information for crack patching ; /* described in Section 3.3 */
24     end if
25   end process_cube
```

Note: Statement 12 means given a small constant δ, which is used to measure the angle of two normals, for any two normals of the triangle in question, their angle is less than δ. In our implementation, the value of δ was set to $30°$.

3.0 The Crack Problem

3.1 Definition

Consider an example to see how a crack is formed (see Figure 4). This shows two neighboring cubes, $C_1(V_1V_2V_3V_4)$ and $C_2(V_1'V_2'V_3'V_4')$, that have been "pulled apart" in order to show their neighboring faces, F_1 and F_2, more clearly (F_1 and F_2 actually represent the common face F of C_1 and C_2). For clarity, only part of the approximated surface within them is shown. A 1-vertex (black) of cube is one that is "inside" the actual surface, while a 0-vertex (white)of cube is one that is "outside" of the surface. The figure only shows the cube vertices for F_1 and F_2. The shaded regions are partial approximated surfaces in cubes C_1 and C_2. The lines AB and BC represent the intersection edges of the surface triangles in C_1 on face F_1 while PQ represents the intersection edge of the surface triangles in C_2 on face F_2. If C_1 is joined to C_2, then the thick line joining A and C is where PQ meets F_1 while the thick polyline joining P and Q is where ABC meets F_2. The

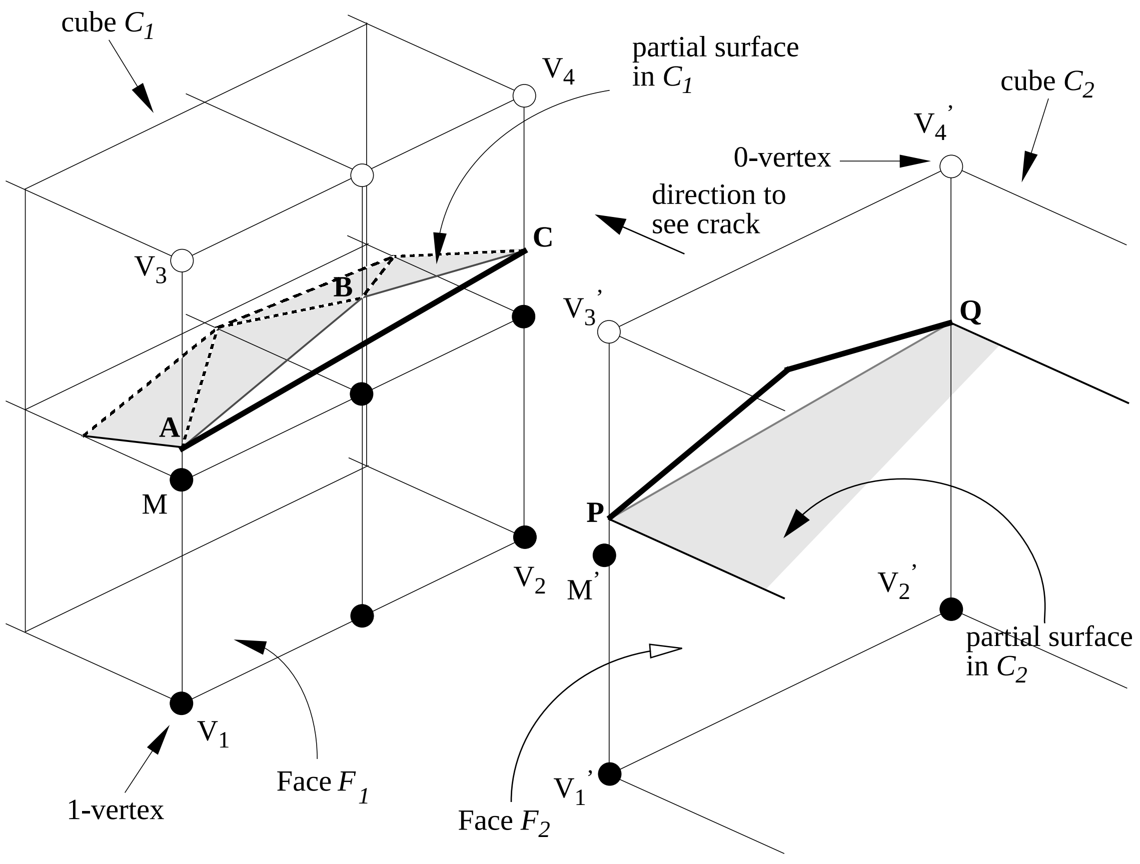


Figure 4. How a crack can arise.

crack would then be the triangular region on face F. The crack can be seen along the direction of the arrow.

As can be seen, C_1 has been sub-divided while C_2 has not. This is because the surface in C_1 has a higher curvature than that in C_2. As a result of this, the polyline approximations of the curve from the two neighboring cubes, C_1 and C_2, on face F are different, resulting in the triangular crack here.

Figures 5 and 6 show how cracks appear in a real data set. It can be clearly seen that cracks appear around the upper left region of the human mouth in Figure 5. Whereas no cracks appear at the same region in Figure 6. Other cracks can also be seen in Figure 5. Before we discuss the solution to the crack problem in depth, we define the following terms:

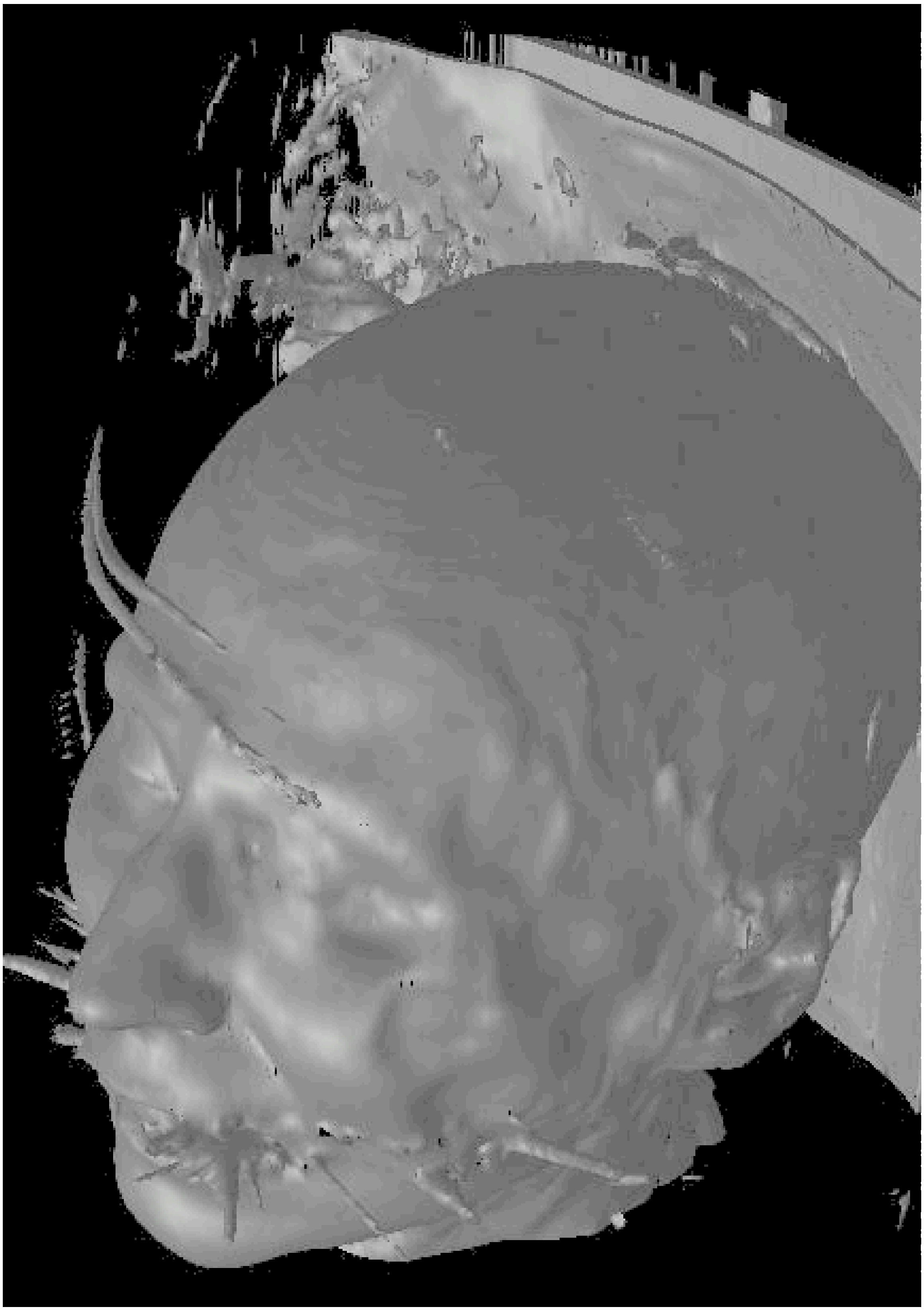


Figure 5. The occurrence of cracks caused by straightforward adaptive approach (skin surface).

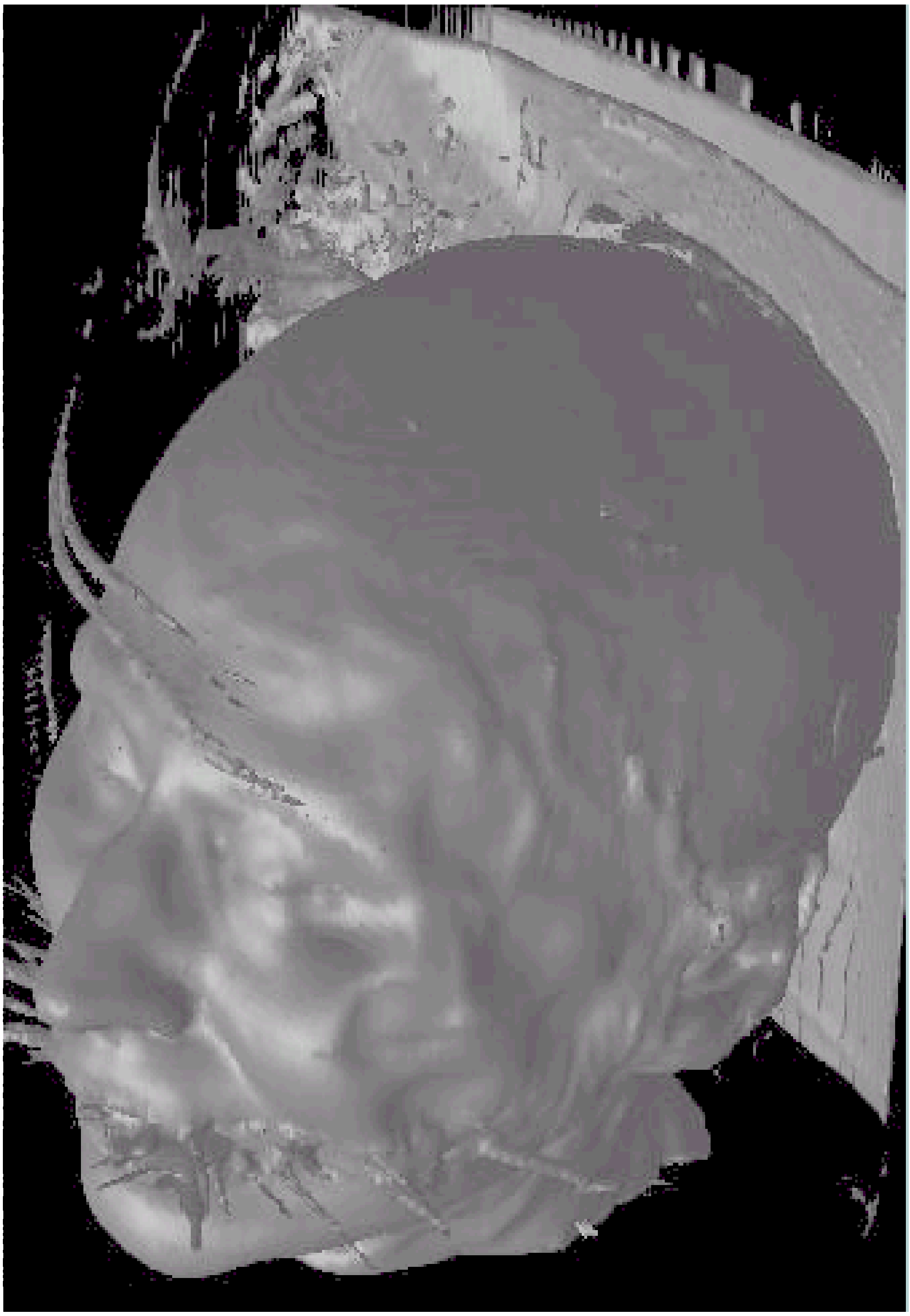


Figure 6. Image generated by MC (skin surface).

- *patch face*: The smallest face for crack patching. It is the common face of two neighboring cubes of equal size, one divided and the other undivided. In Figure 4, face F is a patch face, a common face of two neighboring cubes, C_1 (divided) and C_2(undivided).

- *intersection point*: The approximate intersection point between the actual surface and a cube edge with different colors at the two ends. e.g. in Figure *4*, *A*, B and ***C*** are intersection points.

- *intersection edge*: The linking line of two intersection points. e.g. in Figure *4*, *AB*, *BC* and *PQ* are intersection edges.

The different sizes of neighboring cubes lead to different approximations of the curves, which are formed by a 3D object intersecting the common face between the two neighboring cubes, on the common face. For each curve, one approximation is a line, and the other is a polyline. Thus, these two polylines (a line can be considered as a special polyline) of approximation form a closed polyline or polygon, which is the crack of our concern.

It should be noted that the polyline can consist of patch face edge segments as well as intersection edges. In addition, there is another reason for the occurrence of the crack problem. When a 3D object in the undivided cube along the patch face is small enough, it would be neglected. While the polyline contributed solely by the divided cubes alone the patch face is exactly a closed polyline, which is the crack.

3.2 Solution

To solve the crack problem, we generate polygons with the same shape as those of the cracks and then patch them. The shape of a crack depends on the number and configuration of all cube vertices on the patch face with values above (1-vertex) or below (0-vertex) the threshold value for the surface. Our idea is to reduce all possible cases of shapes to some basic configurations and then design the crack-patching algorithm to cover all cases based on these relatively smaller configurations. Because the size of a patch face may be arbitrary, we can not use the method in MC which reduces the definite 256 cases to 15 basic configurations by making good use of certain symmetries. The key issue here is how to deal with the arbitrary size of a patch face.

The starting point is initially to focus only on the 4-vertex configurations (1-vertex or 0-vertex) of a patch face. This makes the ‘arbitrary’ issue to a ‘definite’ one.

Case 1: Both 1-vertex and 0-vertex present.

In terms of topology there are only two cases of patch face shown in Figure 7 if we consider intersection edges contributed only by the undivided cube. Then for both cases we can add the intersec-

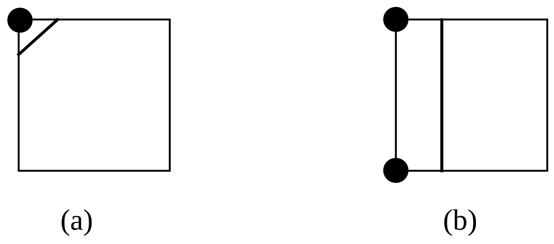


Figure 7. Patch faces.

tion edges contributed by the divided cubes to the patch face to form the basic configurations of cracks, classifying the subcases in terms of the number, N, of patch face edge segments which are needed to form the polygon to patch cracks. Note that the case of two diagonally opposite 1-vertices can be treated on the same basis as (a).

For case (a), we can reduce the number of cases of crack shapes by taking advantage of the symmetry along the diagonal of a patch face which passes through the 1-vertex, and the symmetry along the intersection edge itself. These cases are shown in Figure 8.Note that a solid line represents an intersection edge contributed by the undivided cube along the patch face and, a dotted arc represents a polyline which is made up of the consecutive intersection edges contributed by the divided cubes. Any face edge segment between two intersection points can be substituted with the polyline shown in Figure 9.

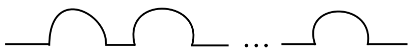


Figure 9. Polyline.

For example, the two cases in Figure 10 are treated equivalently:

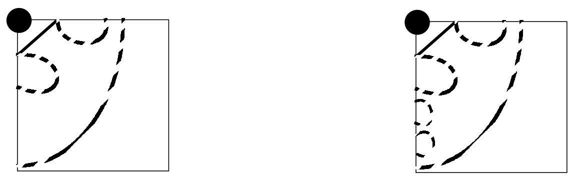


Figure 10. Two equivalent cases.

The letter 'C' in each square in Figure 8 shows the region of a crack. All the subsequent figures in this section also follow the above conventions.

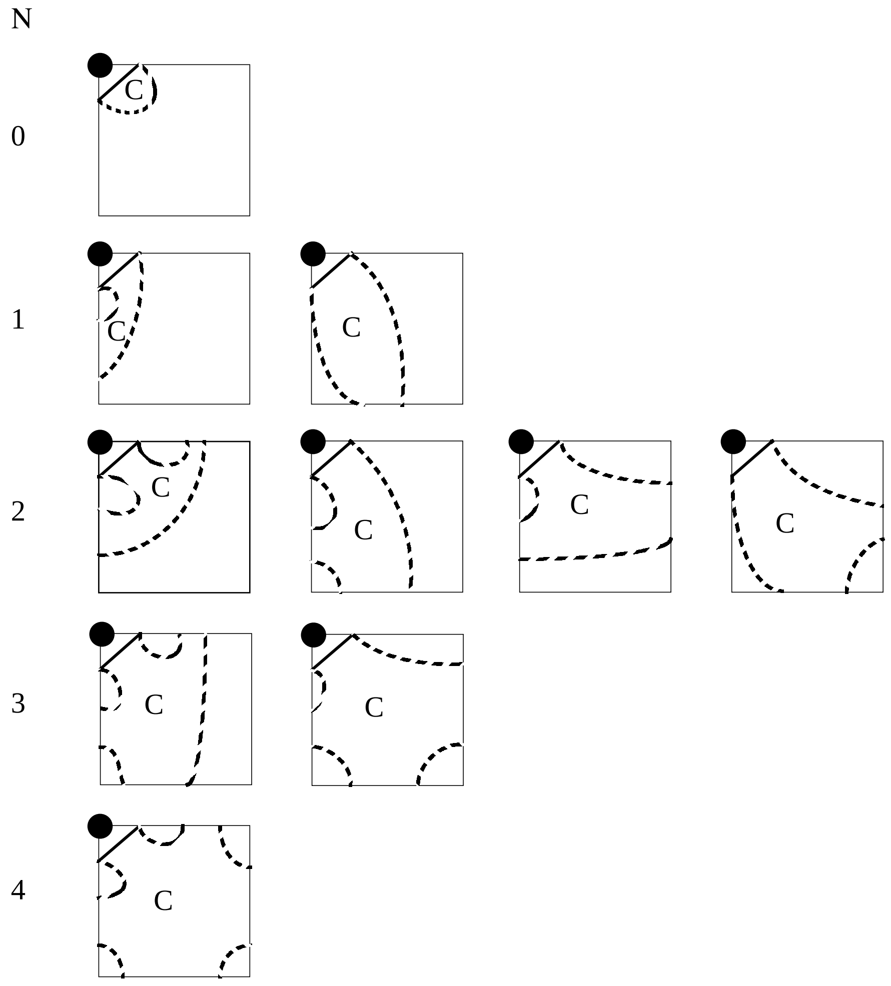


Figure 8. Cases of one 1-vertex on a patch face square.

Each case of the crack obeys the following rules. These rules are the basis for each case in Figure 8.

(1) With each end of a solid line, there is one dotted arc ending.

The above statement can be expressed in another way, i.e. each intersection point contributed by the undivided cube along the patch face must also be the one contributed by the divided cube. For example, in Figure 3, the positions of A and P are same. This is because to calculate an intersection point for $V_1'V_3'$, finally we reach to $V_3'M'$ and calculate the intersection point just based on this unit edge, which is the same as V_3M. The further reason is that first we apply the way of bisection to find the unit edge with different colors at two ends along the patch face edge. And then apply the method of intersection point location in MC to calculate the intersection point within that unit edge. This approach ensures that the above statement is always true.

(2) A patch face edge with different colored vertices at both ends has an odd number of intersection points on it. While a patch face edge with same colored vertices at both ends has an even number of intersection points on it.

To prove this, we make use of the fact that any patch face edge has $2^n + 1$ cube vertices on it for some $n \geq 0$. Our proof is by induction on *n*.

***PROOF*:**

When $n = 0$, the above is true.

Assume that it is true for some $n = k$.

Suppose $n = k + 1$.

Let P_{mid} represent the middle point of an edge *E* having $M = 2^{k+1} + 1$ points with the two ends P_0 and P_M. Then let E_1 represent the edge with the two ends, P_0 and P_{mid}, and E_2 be the edge with the two ends, P_{mid} and P_M, each of which has $2^k + 1$ points.

We need to consider two cases:

(a) *E* has vertices of the same color at both ends.

If P_{mid} has the same color with the two ends, then both E_1 and E_2 have same colored vertices at both ends. Therefore by induction, they both have an even number of intersection points so giving *E* an even number of points.

If P_{mid} has a different color from the two ends, then E_1 and E_2 have different colored vertices at both ends. Therefore by induction, they both have an odd number of vertices and so *E* has an even number of points.

(b) *E* has vertices of different colors at both ends.

No matter which color P_{mid} has, one of E_1 and E_2 has same colored vertices at both ends, and the other has different colored vertices.

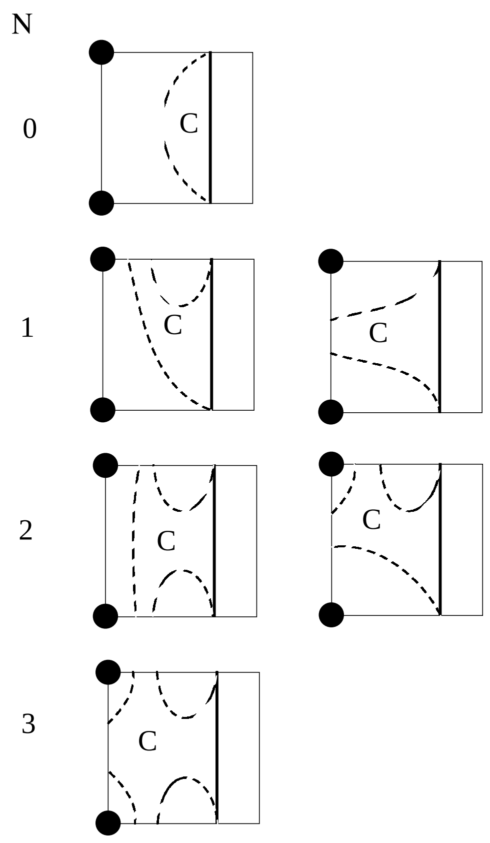


Figure 11. Cases of two 1-vertices on a patch face square.

Then by induction, one edge has an even number of vertices and the other has an odd number, and so *E* has an odd number of vertices.

For case (b) of the patch face, after intersection edges contributed by the divided cubes have been added we can reduce the number of cases by making use of the symmetry along the axis across the patch face edge with two 1-vertices, and the symmetry along the intersection edge itself. These cases are shown in Figure 11. The same considerations with respect to the polyline substitution apply here as well.

Case 2: Only 1-vertex or 0-vertex present.

The configurations are shown in Figure 12 with the same considerations as before on polyline substitution.

Multiple cracks on one patch face is possible. See Figure 13. For the case of patch face with two diagonally opposed 1-vertices, there are at least two cracks on it. It is also possible that any case in

Figure 12 can be combined to form multiple cracks with any case in Figures 10, 11 and case of patch face with two diagonally opposed 1-vertices.

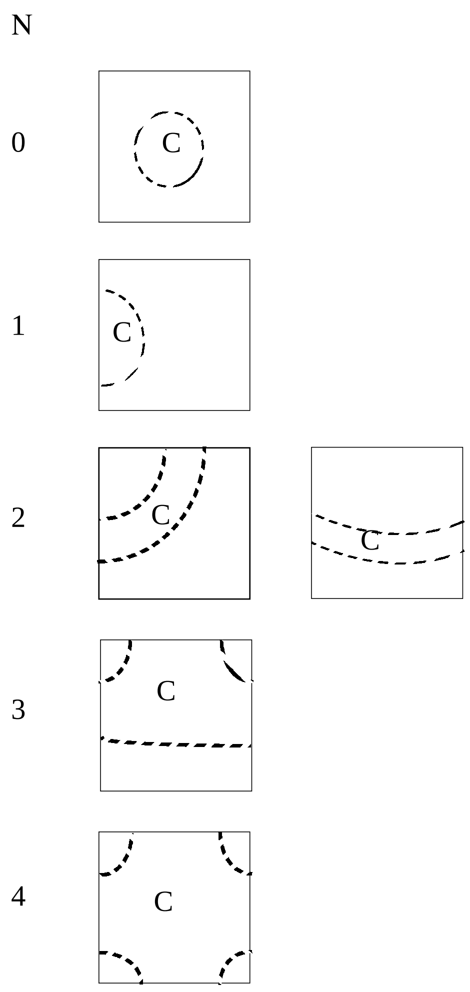


Figure 12. Cases of all 0-vertices or all 1-vertices on a patch face square.

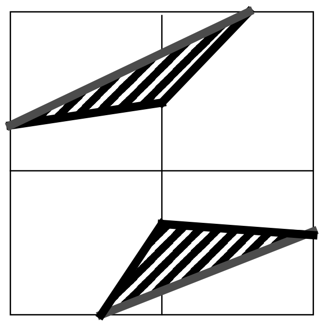


Figure 13. Multiple cracks on one patch face.

Based on the 22 basic configurations of arbitrarily sized cracks in Figure 8, 11, and 12, we can extract the common features to form a simple rule for crack patching. The rule is that there is only one polygon on each patch face, and it is formed by either patch face edge segments or intersection edges. W e shall present this crack-patching algorithm in the following section.

It should be noted that we do some special treatment for the case when the isosurface intersects exactly at a cube vertex. In such a case, the intersection point at a cube vertex is moved by a very small distance along the 6 cube edges respectively. This results in 6 new intersection points instead of the initial one. Then the newly formed cases can be treated in the usual way.

3.3 Crack-patching Algorithm

The information needed to be kept for crack patching includes:

- Position of a cube face which is used to identify multiple patch faces on the face of an initial cube.
- Size of a cube face which is for finding a patch face and is coupled with the position information to identify multiple patch faces on the face of an initial cube.
- Intersection edges on a cube face.
- Normals of each intersection points of those edges.

It should be noted that AMC travels cubes in a scan-line order, i.e. X-->Y-->Z, refer to Figure 14. When AMC processes each cube, besides keeping record of the information of intersection edges and normals on any face, for each of the left, lower and back faces of the cube it checks with the size information of its neighboring cube to see if any crack occurs. If it does, both size and position of the larger cube of the two neighboring cubes will be stored, which define a patch face. While for each of the other 3 faces both size and position of current cube face will be stored.

The action of crack patching is taken after an initial cube has been processed. Cracks on the left, lower and back faces of the initial cube and on the faces between smaller cubes within the initial cube are patched. Because the intersection edges forming cracks within a certain patch face are not necessarily stored consecutively due to the traversal sequence, it is necessary to keep the information which defines the patch face so that we can correctly pick the intersection edges. For each patch face on an initial cube face, polygons are formed to exactly cover all the cracks one by one. A crack is sometimes formed by patch face edge segments as well as intersection edges.

The details of how cracks are patched on a patch face is in the following pseudocode:

```
1.  select an intersection edge within the patch face ;
2.  assign this intersection edge as preceding edge ;
3.  specify one end of preceding edge as starting end and the other as terminal end ;
4.  first push starting end into a polyline stack, and then push terminal end into the stack ;
5.  assign polyline-closed-flag as non-closed ;
6.  while (polyline-closed-flag == non-closed)
7.      if (successfully find another intersection edge connected with preceding edge at terminal end)
8.      then
9.      specify the connected end of preceding edge and this new intersection edge as starting end,
          and the other end of this new intersection edge as terminal end ;
10.     assign this intersection edge as preceding edge ;
11.     else /* successfully find another intersection edge, one end of which is on the same patch
              face edge as terminal end */
12.      specify the end of this new intersection edge starting end, which is on the same patch face
            edge as terminal end, and the other end of this new intersection edge as terminal end ;
13.       assign this new intersection edge as preceding edge ;
14.       push starting end into the stack ;
15.     end if
      /* check if the polyline is closed */
16.     if (terminal end == the first point of the polyline) then
```

```
17.         assign polyline-closed-flag as closed ;
18.      **else if** (terminal end is on the same patch face edge as the first point of the polyline) **then**
19.            push terminal end into the stack ;
20.            assign polyline-closed-flag as closed ;
21.         **else** /* polyline-closed-flag retained as non-closed */
22.            push terminal end into the stack ;
23.         **end if**
24.      **end if**
25.   **end while**
```

It can be seen that a closed polyline is in the stack. This method is correct only if multiple cracks on a patch face are not interconnected. But this is always true, and we can prove it by reductio ad absurdum.

Theorem: Multiple cracks on a patch face are not interconnected.

PROOF:

Assume that multiple cracks on a patch face are interconnected. Then there exists at least one intersection point which belongs to two cracks simultaneously. There are three kinds of such a point. Let's consider them one by one.

(1) The intersection point is an internal one, i.e. it is not on a patch face edge.

Since this kind of intersection point is contributed only by the divided cubes along the patch face, and the edge which the intersection point is on belongs to only two cubes, it means only two intersection edges pass through that intersection point. Thus there is no extra intersection edge passing through that point to be contributed to cover another crack.

(2) The intersection point is on the patch face edge and also the one contributed by both divided and undivided cubes alone the patch face.

In this case, the two intersection edges passing through that intersection point contributed by two cubes can only belong to one crack.

(3) The intersection point is on the patch face edge and is not the one contributed by the undivided cube.

One intersection edge passing through that intersection point contributed by one cube can only belong to one crack.

It follows that the statement of the theorem is true.

The time complexity to locate all the participants in a single patch is $O\left(n^2\right)$, where *n* is the number of intersection edges on a patch face, which depends on the difference between division level of the two sides of the patch face.

4.0 Implementation and Results

4.1 Space Required for Crack Patching

As mentioned in the pseudocode of AMC algorithm in section 2.2, crack patching occurs when AMC just finishes processing one initial cube. This is essential for an efficient data structure for crack patching. The space for crack-patching information can be shared in the following way. All the information on the faces of initial cubes for crack patching on X-Y plane share the common space, whose size is just enough to store information for cracks on faces in one slice.All the information on the faces of initial cubes on X-Z plane share the common space, whose size is just enough for cracks on faces in one strip.All the information on the faces of initial cubes on Y-Z plane share the common space, whose size is just enough for cracks on face in one initial cube.And finally all the information on the faces within initial cubes can share the common space, whose size is just big enough for cracks on faces within one initial cube.crack patching. As a result, Only $O\left(n^2\right)$ working memory is needed for crack patching. Figure 14 shows the traversed initial cubes, and Figure 15 shows the common space for crack patching.

4.2 Results

We implemented both original MC and our adaptive AMC on an IBM RS-6000/320 workstation. Table 1, 2 and 3 show the performance comparisons between them using different volumetric data sets, and Figure 16, 17 and 18 are the corresponding images. Note that the time item includes both the running time for surface extraction and rendering phases. It can be seen that the average time

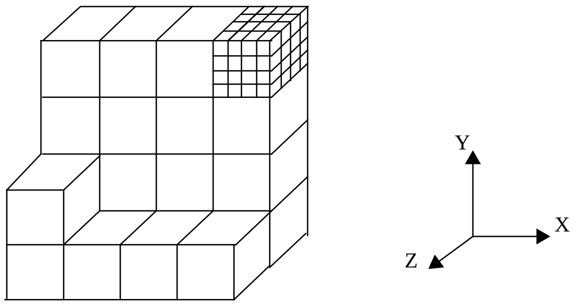


Figure 14. An example of traveled cubes with initial cube size of 4 and volumetric data set size of 16 x 16 x 16.

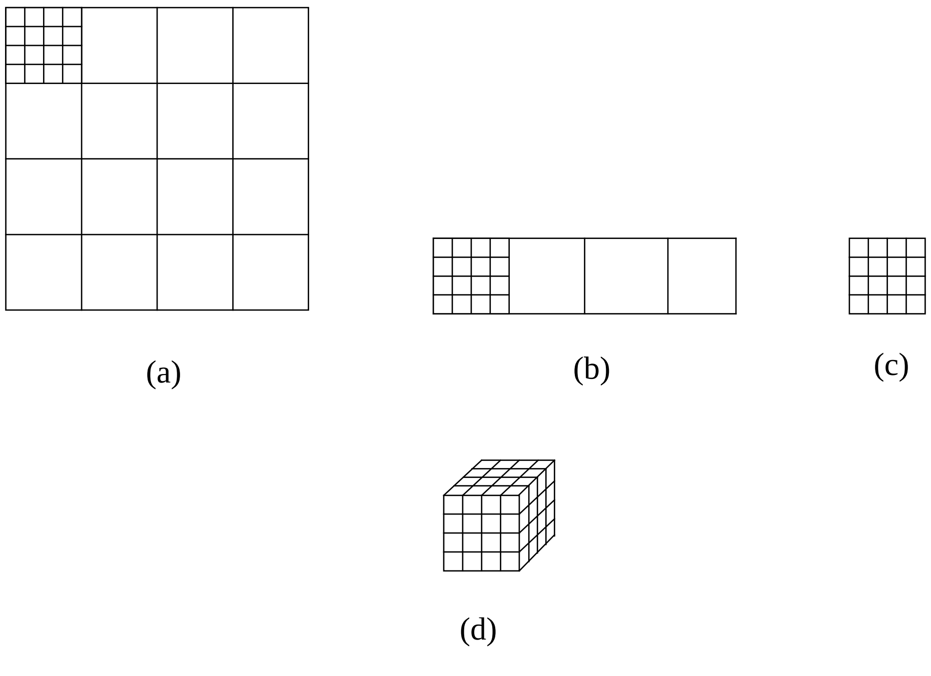


Figure 15. (a), (b) and (c) represent the common space for cracks on plane X-Y,X-Z and Y-Z respectively, and (d) represents the common space for cracks on faces within initial cube of size 4. The size of volumetric data set is 16 x 16 x 16.

as well as the number of triangles are reduced substantially, and the image quality is similar to that of MC. Certain sharp features are however lost if too large a starting size is used.

We found an interesting effect when running the adaptive algorithm with initial cube size of two using the volumetric data set of a $256 \times 256 \times 113$ CT scan of a human head. It was found that

Algorithm	Time(seconds)	% of AMC_X/MC	Number of Triangles	% of AMC_X/MC	Image
MC	331		718,964		Fig. 17.a
AMC-2	230	69%	299,292	42%	Fig. 17b
AMC-4	123	37%	181,230	25%	Fig. 17.c
AMC-8	79	24%	110,602	15%	Fig. 17.d

Table 1. Performance comparison (skin surface) using the data set of a 256 x 256 x 113 CT scan of the human head.

Note: AMC-X denotes the AMC algorithm with the initial cube size of X.

Algorithm	Time(seconds)	% of AMC_X/MC	Number of Triangles	% of AMC_X/MC	Image
MC	278		594,686		Fig. 18.a
AMC-2	213	77%	269,470	45%	Fig. 18.b
AMC-4	124	45%	184,122	31%	Fig. 18.c
AMC-8	96	35%	145,938	25%	Fig. 18.d

Table2. Performance comparison (bone surface) using the data set of a 256 x 256 x 113 CT scan of the human head.

Algorithm	Time(seconds)	% of AMC-X/MC	Number of Triangles	% of AMC-X/MC	Image
MC	164		393,606		Fig. 19.a
AMC-2	81	49%	102,868	26%	Fig. 19.b
AMC-4	39	24%	52,832	13%	Fig. 19.c

Table3. Performance comparison using the data set of a 256 x 256 x 43 CT scan of a machine part.

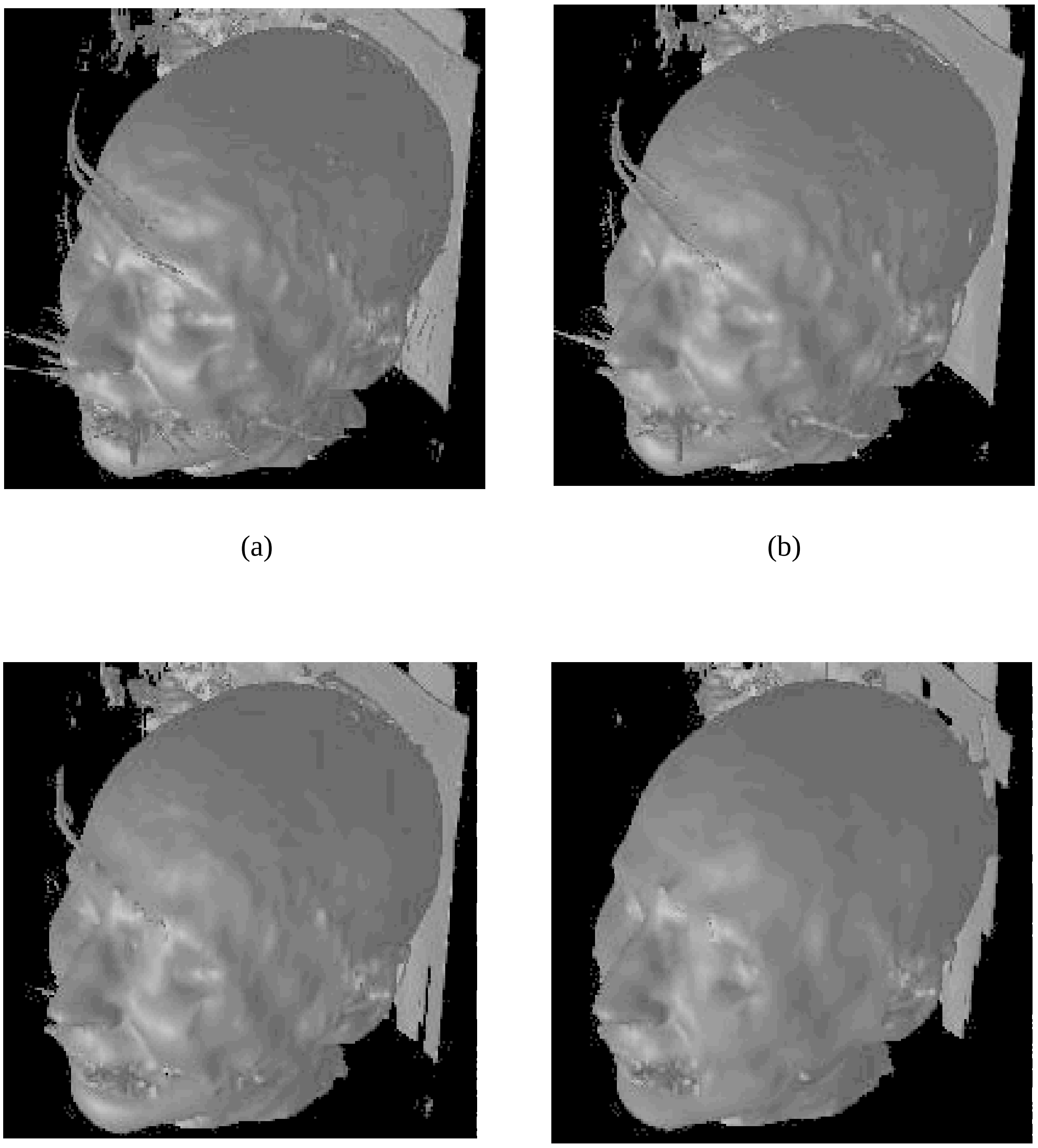
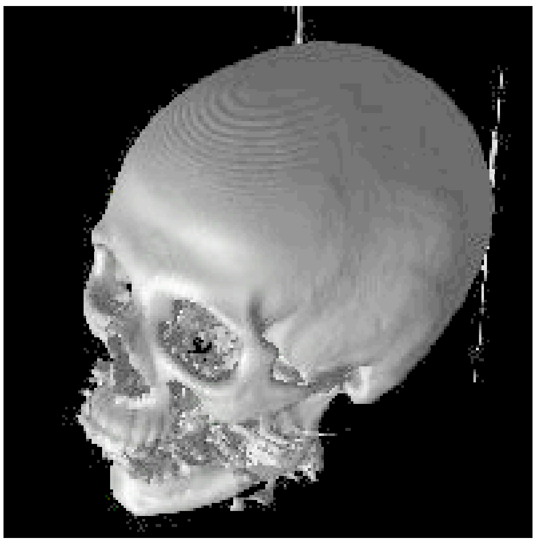
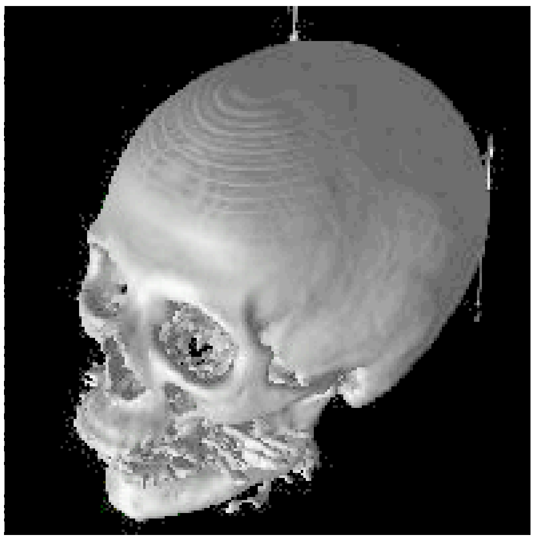


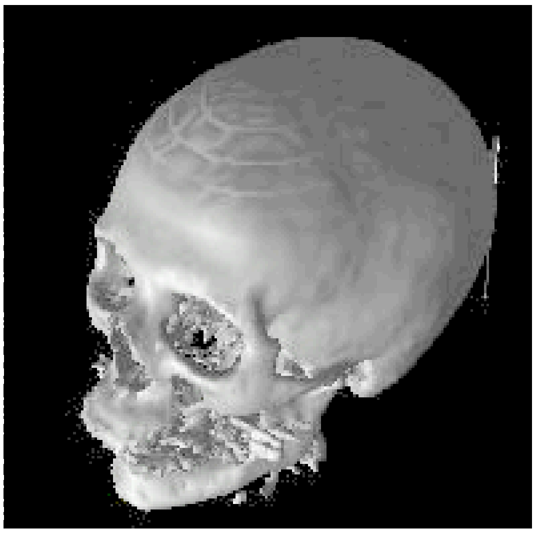
Figure 16. Skin surface generated by using the data set of a 256 x 256 x 113 CT scan of a human head.(a),(b),(c) and (d) are generated by algorithm MC, AMC-2, AMC-4 and AMC-8 respectively.



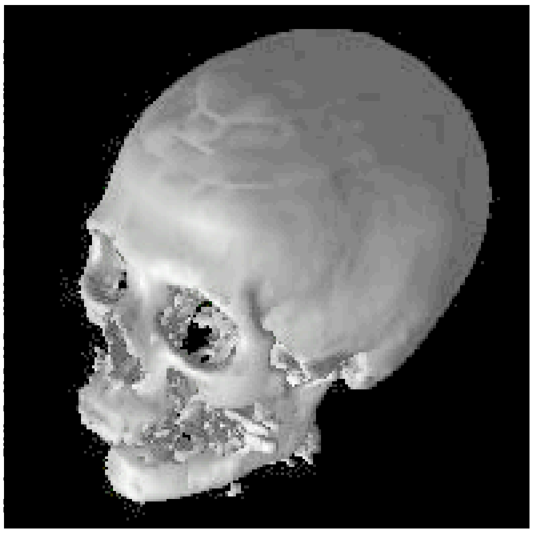
(a)



(b)

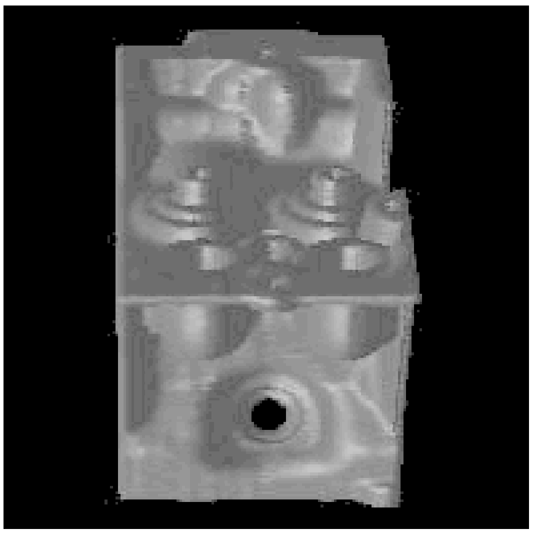


(c)

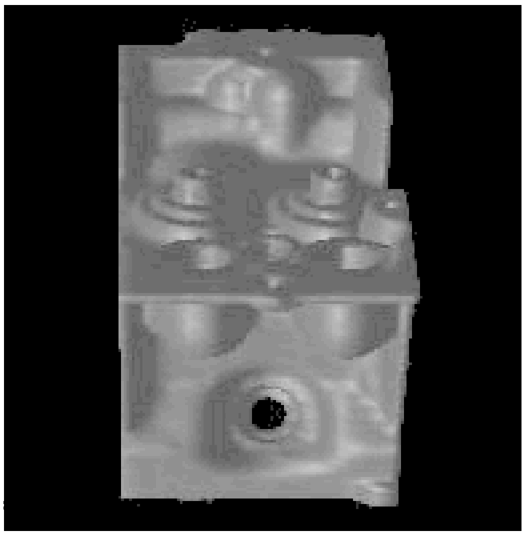


(d)

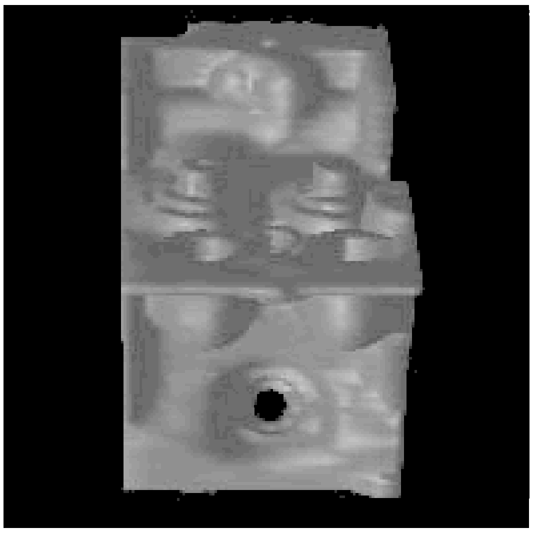
Figure 17. Bone surface generated by using the data set of a 256 x 256 x 113 CT scan of a human head. (a),(b),(c) and (d) are generated by algorithm MC, AMC-2, AMC-4 and AMC-8 respectively.



(a)



(b)



(c)

Figure 18. Images generated by using the data set of a 256 x 256 x 43 CT scan of machine part.(a), (b) and (c) are generated by algorithm MC,AMC-2, and AMC-4 respectively.

there existed about 10,000 cracks, but the number of cracks that were visible was much smaller (see Figure 11).

5.0 Conclusions and discussions

A new high resolution 3D surface construction algorithm has been presented. The basic idea is to adapt the size of triangles of representation to the shape of the surface. The speed of surface construction is improved, and the number of triangles representing the surface is significantly reduced. The latter is essential for interactive manipulation of 3D surfaces.

The image generated by AMC are almost as good as those generated by MC if the initial cube size of two is chosen (Figure 16b, 17b, and 18b). The result of AMC is better with smoother surface than with a surface with sharp features. The larger the size of an initial cube is, the better the result can be achieved. It is true that the image quality drops with larger sizes of the initial cube (Figure 16b & 16c, 17b & 17c, and 18b). So the size of initial cube can be chosen based on requirements. If image quality is of prime concern, the initial cube size of two can be chosen to get an image as good as the one generated by MC. Alternatively, better time and space results can be obtained with an initial cube of larger size at the expense of image quality, which may be acceptable in some practical application situation. One such a scenario is in choosing the right threshold -- experimentation can be done at low quality while the final image can be displayed at full resolution.

The major contribution of our adaptive algorithm is in the solution of the crack problem. We have proposed a simple but complete method by first abstracting 22 basic configurations of arbitrary size cracks and then reduce the handling of these configurations to a simple rule. We use only $\boldsymbol{O}(n^2)$ working memory for patching the cracks.

In terms of image quality, choosing initial cube size as two is suggested. Under such a choice, the bounded error in the surface is almost 2 voxels. Another parameter that can control the image quality is δ, which is related to the angle of two triangle normals (see section 2.2). In our implementation the value of δ chosen is $30°$. It is possible that a part of an object is missed. This is because only 8 corners of a non-unit cube are checked. A simple solution for this is to combine the octree technique (Wilhelms J, Van Gelder A 1992) with this adaptive MC approach. The octree technique is used to store summary information of any size of cubes, and thus it helps to see if any object exists within a non-unit cube.

If the isosurface contains real cracks, i.e. the isosurface itself has some cracks, this should not be a problem for our algorithm to obtain the correct representation. This is because our method is not based on the shape of the original surface. Instead, it is solely based on the specific configuration of the cube in question. However, it is true that the isosurface produced by our AMC is discontinuous in the region where cracks appear. This is because even though the crack is patched, there are different levels of subdivision on the common face between the neighboring cubes. This problem can be solved by post-processing the surface to smooth the discontinuous region of the surface, if needed.

Recently, Ning and Bloomenthal (1993) evaluate the principal polygonization algorithms according to topological issues, implementation complexity, and polygon count. Compared to the three major algorithms they evaluate, i.e. tetrahedral decomposition, single-entry cubical, multi-entry cubical, AMC produces the fewest polygon count with consistent topology, and belongs to high implementation complexity range.

6.0 Acknowledgment

We would like to thank our colleague Chun Pong Yu for his help in writing this paper. We would also like to thank the anonymous reviewers for the very detailed, helpful comments and suggestions. The paper quality has significantly improved due to their inputs

References

Artzy E, Frieder G, Herman, GT (1981) The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm., *Computer Graphics and Image Processing*, 15(1):1-24.

Baker HH (1989) Building Surfaces of Evolution: The Weaving wall, *International Journal of Computer Vision*, 3(1):51-71.

Chen L, Herman, GT, Reynolds RA, Udupa JK (1985) Surface shading in the cuberille environment, *IEEE Computer Graphics and Applications,* **5**(12):33-43.

Cline HE, Lorensen WE, Ludke S, Crawford CR, Teeter BC (1988) Two algorithms for the three-dimensional reconstruction of tomograms, *Medical Physics,* **15**(3):320-327.

Clay RD, Moreton HP (1988) Efficient adaptive subdivision of Bézier surfaces. In *Proceedings of the Eurographics'88 Conference*, pages 357─371, North-Holland, Amsterdam.

Durst MJ (1988) Additional Reference to “Marching Cubes”. *Computer Graphics*. 22(2):72-73.

Fuchs H, Levoy M, Pizer SM (1989) Interactive Visualization of 3D Medical Data, *Computer*, 22(8):46-51.

Gordon B, Udupa JK (1989) Fast surface tracking in three-dimensional binary images, *Computer Vision, Graphics and Image Processing,* 29(2):196-214.

Herman GT, Liu HK (1979) Three-dimensional display of human organs from computed tomograms, *Computer Graphics and Image Processing,* **9**(1):1-21.

Kaufman A (1990) Chapter 1: Introduction to volume visualization. *Volume Visualization.* IEEE Computer Society Press, Los Alamitos, California.

Lorensen WE, Cline HE (1987) Marching Cubes: a high resolution 3D surface construction algorithm. *Computer Graphics,* 21(4):163-169.

Muller H and Stark M 1993, Adaptive Generation of surfaces in Volume Data, *The Visual Computer*, (1993)9:182-199.

Ning P and Bloomenthal J (1993) An Evaluation of Implicit Surface Tilers, *IEEE Computer Graphics & Applications*, Nov. pp. 33-41

Nielson GM, Hamann B (1991) The Asymptotic Decider: resolving the ambiguity in Marching Cubes. In *Proceedings of Visualization'91 Conference*, pp. 83-91, San Diego, California.

Schroeder WJ, Zarge JA and Lorensen WE (1992), Decimation of Triangle Meshes, *Computer Graphics*, 26(3): 65-70.

Sobierajski L., Kaufman A, Cohen D, Yagel R, Acker D (1993) A Fast Display Method for Volumetric Data, *The Visual Computer*, 10(2):116-124.

Trivedi SS, Herman GT, Udupa JK (1986) Segmentation into three classes using gradients. *IEEE Transactions on Medical Imaging*, MI-**5**(2):116-119.

Turk G. (1992) Re-Tiling of Polygonal Surfaces, *Computer Graphics*, 26(3):55-64.

Wilhelms J, Van Gelder A (1990) Topological Considerations in Isosurface Generation, *Computer Graphics*, 24(5):79-86.

Wilhelms J, Van Gelder A (1992) Octree for Faster Isosurface Generation, *ACM Transaction on Graphics*, 11(3):201-227.

Udupa JK, Hung HM (1990) Surface versus volume rendering: a comparative assessment. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, pp. 83-91, IEEE Computer Society Press, Los Alamitos, California.