Rapid #: -25502943

CROSS REF ID: **2688745270006826**

LENDER: **MFL (California State Univ Moss Landing) :: Main Library**

BORROWER: **UK4ZN (Staffordshire University) :: Main Library**

TYPE:	Article CC:CCL
JOURNAL TITLE:	The visual computer
USER JOURNAL TITLE:	The Visual computer
ARTICLE TITLE:	Adaptive marching cubes
ARTICLE AUTHOR:	Shu, Renben
VOLUME:	11
ISSUE:	4
MONTH:	
YEAR:	1995-04
PAGES:	202-217
ISSN:	0178-2789
OCLC #:	

Processed by RapidX: 10/27/2025 10:12:15 AM

The Visual Computer

Adaptive marching cubes

Renben Shu, Chen Zhou, Mohan S. Kankanhalli

Institute of Systems Science,
National University of Singapore,
Kent Ridge, Singapore 0511

The marching cubes algorithm (MC) is a powerful technique for surface rendering that can produce very high-quality images. However, it is not suitable for interactive manipulation of the 3D surfaces constructed from high-resolution volume datasets 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 dataset 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 arbitrarily sized cracks and then reducing the handling of these configurations to a simple rule. It requires only $O(n^2)$ working memory for a $n \times n \times n$ volume data set.

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

Correspondence to: R. Shu

1 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. An extraction step and then conventional computer graphics techniques are used 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 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. 1990; 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) can create very high-quality images by generating a set of triangles that closely approximates a surface of interest. Much effort has been devoted to improving the MC in various ways. The original MC proposed by Lorensen and Cline (1987) suffers from the hole problem, which is caused by ambiguities in the method of approximating the surface. This has been dealt with by Baker (1989), Cline and Lorensen (1988), Durst (1988), Fuchs et al. (1990), Nielson and Hamann (1991), and 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 it. However, 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

The Visual Computer (1995) 11:202–217

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 on a polygonal mesh, and then discards the old points to create a new mesh. This reduces 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 pre-specified tolerance of a plane that approximates the surface near the vertex. Their method also identifies sharp edges and sharp corners and makes sure such features are retained to represent the original data better. Both methods post-process the triangulated surface generated by MC.

More recently, Muller and Stark (1993) propose a method that adaptively generates the surface, i.e., adapts the size of triangles to the shape of surface. The problem with their method is that certain large 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 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 algorithm such as that of Clay and Moreton (1988). Although Muller and Stark (1993) solve the crack problem, their method is at the expense of losing certain features of objects, sometimes even large features. Their method deals with the crack problem by processing the common faces of neighbouring 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 large 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 of problem underlying the Muller and Stark algorithm.

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

2 Adaptive marching cubes

2.1 Background

We briefly summarize the MC here to simplify the presentation of our algorithm. The MC 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 (Fig. 1). The method processes one voxel at a time. The values of grid points and linear interpretation 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 Fig. 2, where a grid point that is marked indicates its value as being above the threshold. While there are $2^8 = 256$ possible configurations, there are only 15 shown in Fig. 2. This is because some configurations are equivalent with respect to certain operations. First, the number can be

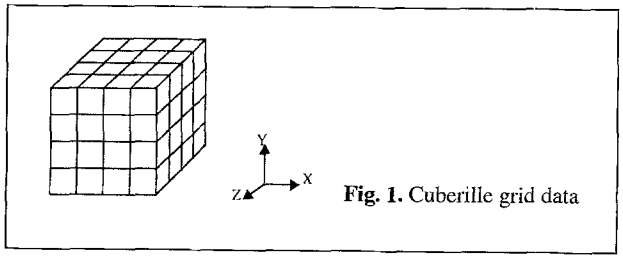
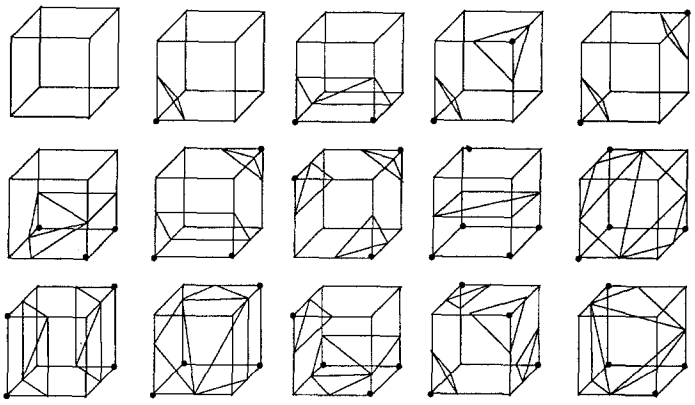
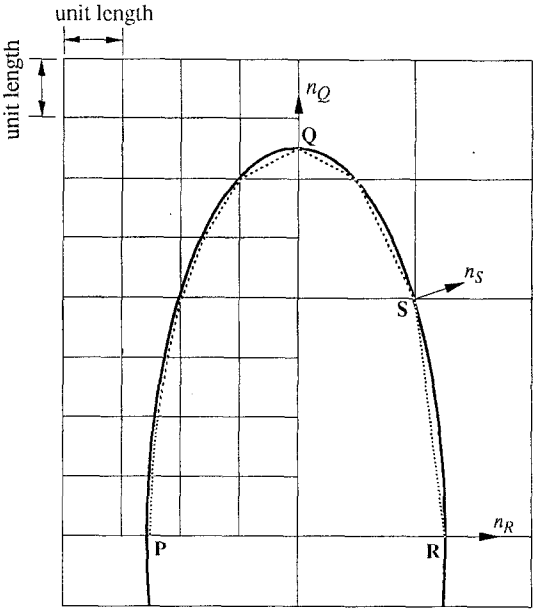


Fig. 1. Cuberille grid data



2

Fig. 2. Configurations of triangulated cubes



3

Fig. 3. How the adaptive marching cubes algorithm (AMC) works in 2D

reduced to 128 by assuming the two configurations are equivalent if the marked grid points and the 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 the 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 of data set). As we mentioned in section 1, 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 $m \leq min(i,j,k)$. Next we use the MC surface configuration approach described in Lorensen and Cline (1987) to triangulate the cubes, considering only the values of the eight vertices of cubes. Then we partition these cubes recursively into smaller and smaller cubes based on the smoothness of the surface within them. For each cube, if the actual inside surface is flat enough, the triangles of the MC surface configurations are used to approximate it. Conversely, if the actual surface has a high curvature, the cube will be partitioned into eight subcubes. The process will be repeated until all surfaces in the subcubes are flat enough to be approximated with the MC configurations, or until 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 seven straight lines to approximate it. This is shown by the dashed lines along PQ. However if AMC is used, only three 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 four containing ellipse QS is partitioned into four subsquares, and two lines are used to approximate it.
This idea can be extended easily to the 3D case that 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 Sect. 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 tri-
         angles with normals n0, n1 and n2,
         ARCCOS(ni • n(i+1)mod3) < δ  for any
         i ∈ {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 Sect. 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, the angle is less than δ. In our implementation, the value of δ was set to 30°.

3 The crack problem

3.1 Definition

Consider an example to see how a crack is formed (Fig. 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" 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 a cube is one that is "inside" the actual surface, while a 0-vertex (white) of a 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 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 subdivided 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, 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 Fig. 5, whereas no cracks appear at the same region in Fig. 6. Other cracks can also be seen in Fig. 5. Before we discuss the solution to the crack problem in depth, we define the following terms:

A *patch face* is 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 Fig. 4, face F is a patch face, a common face of two neighboring cubes, C_1 (divided) and C_2 (undivided).
An *intersection point* is the approximate intersection point between the actual surface and a cube edge with different colors at the two ends; e.g., in Fig. 4, A, B and C are intersection points.

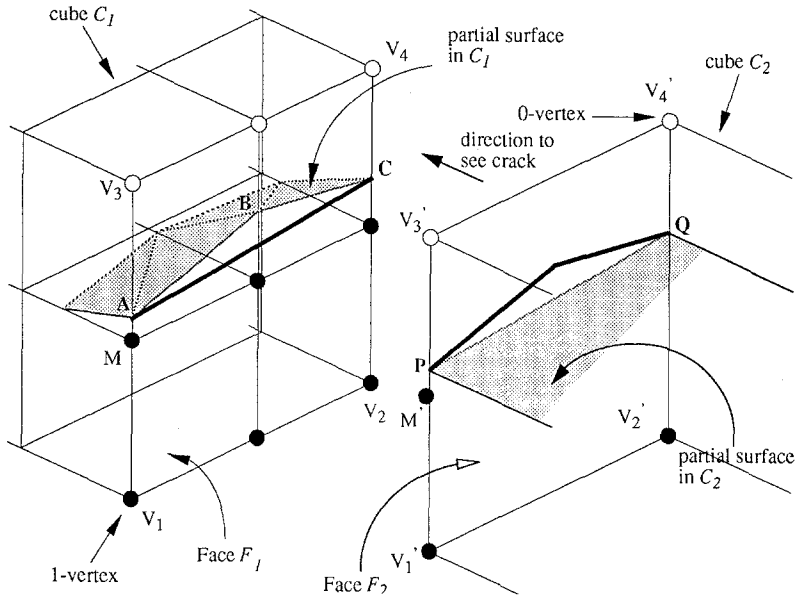


Fig. 4. How a crack originates

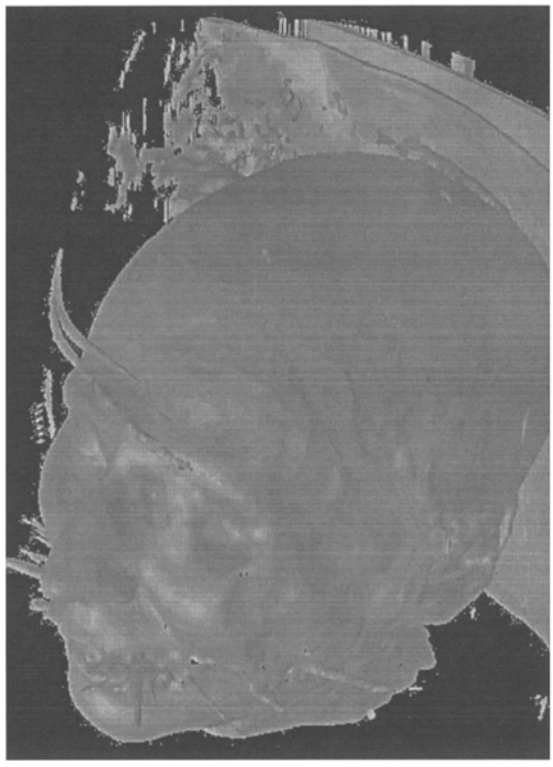
An *intersection edge* is the linking line of two intersection points; e.g., in Fig. 4, *AB*, *BC* and *PQ* are intersection edges.

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 edge segments of the patch face as well as intersection edges. In addition, there is another reason for the occurrence of the crack problem. If a 3D object in the undivided cube along the patch face is small enough, it is neglected, while the polyline contributed solely by the divided cubes alone the patch face is exactly a closed polyline, which is the crack.

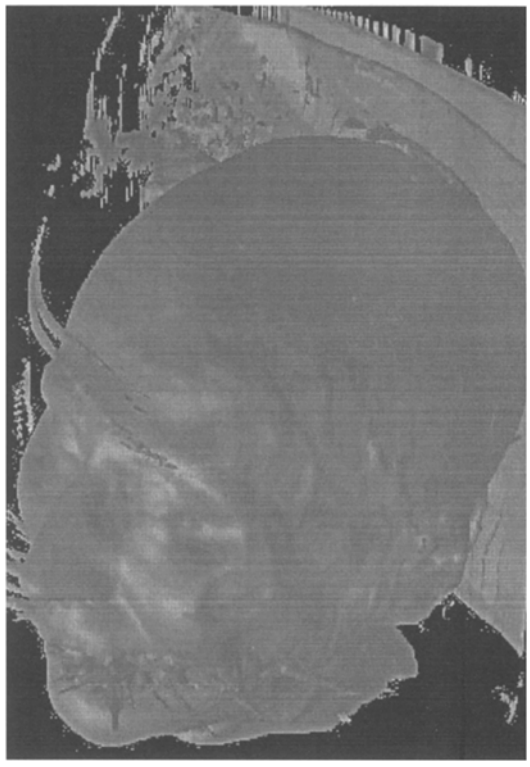
3.2 Solution

To solve the crack problem, we generate polygons with the same shapes 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 values 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 the cases based on these relatively smaller configurations. Because the size of a patch face may be arbitrary, we cannot use the method in MC that 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 initially focuses 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 Fig. 7 if we consider intersection edges contributed only by the undivided cube. Then for both cases we can add the intersection edges contributed by the divided cubes to the patch face to form the basic configurations of cracks. We classify the subcases in terms of the



5



6

Fig. 5. The occurrence of cracks, here in a skin surface, caused by a straightforward adaptive approach

Fig. 6. An image (skin surface) generated by the marching cubes algorithm (MC)

number N of patch face edge segments that are needed to form the polygon to patch the cracks. Note that the case of two diagonally opposite 1-vertices can be treated on the same basis as the patch face of Fig. 7a.

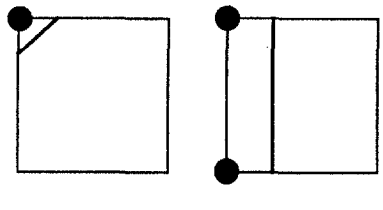
For the example in Fig. 7a we can reduce the number of cases of crack shapes by taking advantage of the symmetry along the diagonal of a patch face which passes through the 1-vertex and the symmetry along the intersection edge itself. These cases are shown in Fig. 8. Note that a solid line represents 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 Fig. 9 and for example, the two cases in Fig. 10 are treated equivalently.

The letter C in each square in Fig. 8 shows the region of a crack. All the subsequent figures in this section also follow the conventions just given.

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

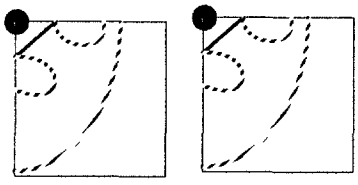
1. At each end of a solid line, there is one dotted arc ending.

This statement can be expressed in another way: 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 Fig. 4, 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

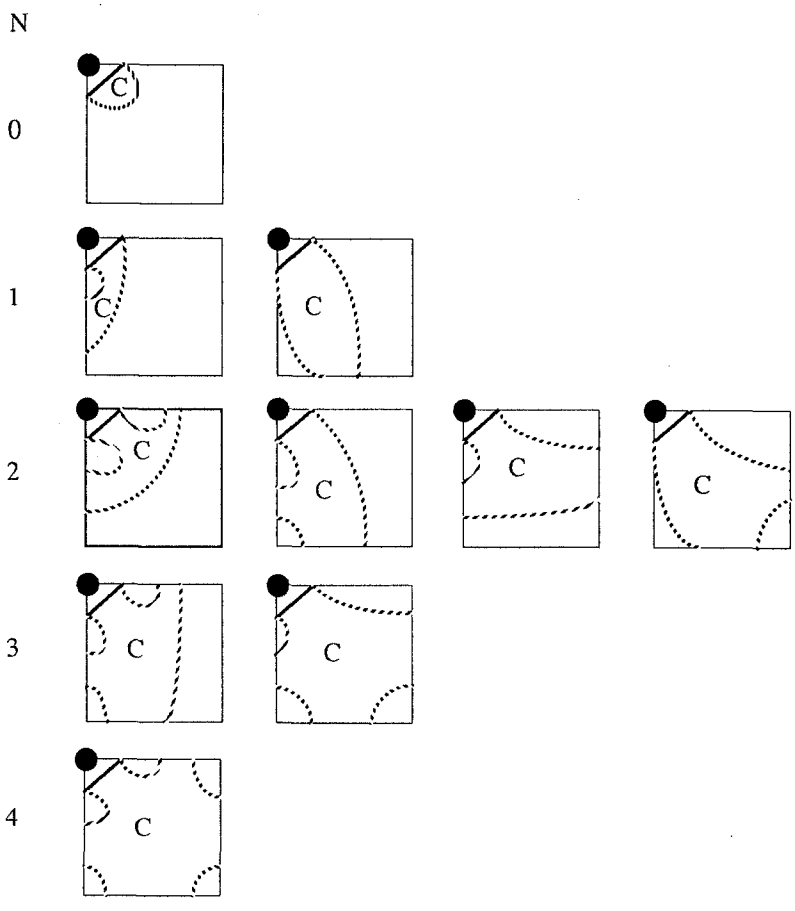


7a 7b

9



10a 10b



8

Fig. 7a, b. Patch faces

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

Fig. 9. Polyline

Fig. 10. Two equivalent cases

calculate the intersection point just based on this unit edge, which is the same as V_3M. The further reason is that first we bisect to find the unit edge with different colors at two ends along the patch-face edge. Then we locate the intersection point in MC to calculate the intersection point within that unit edge. This approach ensures that statement 1 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 $2^a + 1$ cube vertices on it for some $n \geq 0$. Our proof is by induction on n.

PROOF.

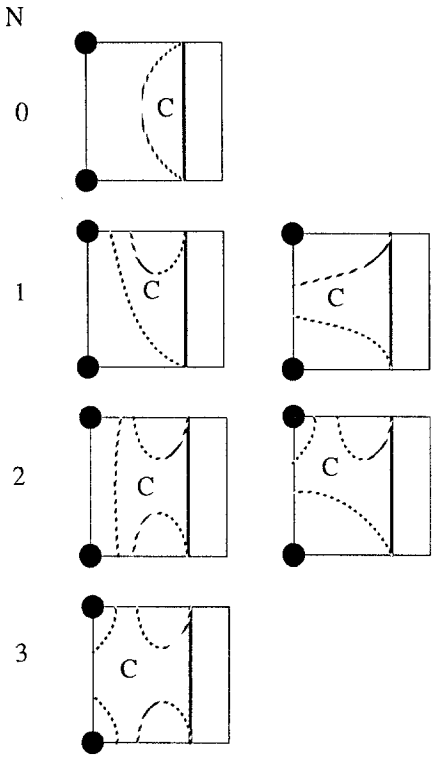
When $n = 0$, statement 2 is true.

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

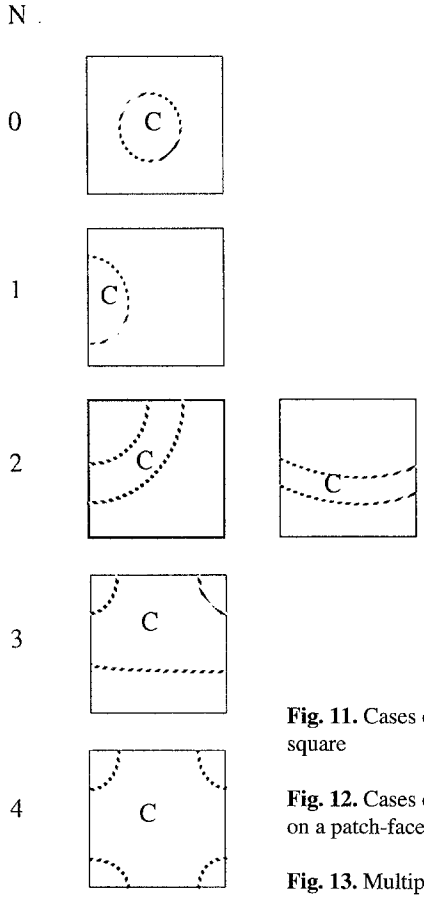
Suppose $n = k + 1$.

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:

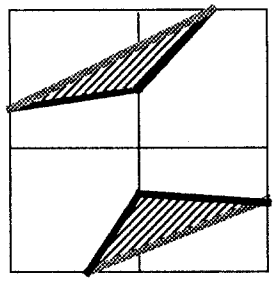
A. E has vertices of the same color at both ends. If P_{mid} has the same color at both ends, then both E_1 and E_2 have vertices of the same color at both



11



12



13

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

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

Fig. 13. Multiple cracks on one patch face

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 at each end, then E_1 and E_2 have vertices of different colors at each end. Therefore by induction, they both have an odd number of vertices, so E has an even number of points.

B. E has vertices of different colors at each end. No matter which *color* P_{mid} is, one of E_1 and E_2 has vertices of the same color at both ends, and the other has vertices of different colors.
Then by induction, one edge has an even number of vertices, and the other has an odd number, 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 Fig. 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 Fig. 12 with the same considerations as before on polyline substitution. Multiple cracks on one patch face are possible (Fig. 13). For the case of a patch face with two diagonally opposed 1-vertices, there are at least

two cracks. It is also possible that any case in Fig. 12 can be combined to form multiple cracks with any case in Figs. 10 and 11 and with any case of a patch face with two diagonally opposed 1-vertices.

Based on the 22 basic configurations of arbitrarily sized cracks in Figs. 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. We shall present this crack-patching algorithm in the following section.

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

3.3 The crack-patching algorithm

The information that must be kept for crack patching includes:

- The position of a cube face that is used to identify multiple patch faces on the face of an initial cube
- The size of a cube face for finding a patch face that 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 all intersection points of those edges

It should be noted that AMC travels cubes in a scanline order, i.e., X → Y → Z (Fig. 14). When the AMC processes each cube, it keeps a record of the information of intersection edges and normals on any face. Also, for each of the left, lower, and back faces of the cube, it checks the size information of the neighboring cube to see if a crack occurs. If it does, both size and position of the larger cube of the two neighboring cubes that define a patch face will be stored. For each of the other three faces, both the size and the position of the current cube face will be stored.

Cracks are patched 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 defining the patch face so that we can pick the intersection edges correctly. For each patch face on an initial cube face, polygons are formed to cover all the cracks exactly, 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 are given 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 finishing end;
4. First push starting end into a polyline stack, and then push finishing end into the stack;
5. Assign polyline-closed-flag as open;
6. **While** (polyline-closed-flag = = open)
7. **If** (successfully find another intersection edge connected with preceding edge at the terminal end)
8. **Then**
9. Specify the connected end of preceding edge and this new intersection edge as the 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 finishing end into the stack;
20.      Assign polyline-closed-flag as closed;
21.    Else/* polyline-closed-flag retained as
       open*/
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. However, 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 at least one intersection point exists that belongs to two cracks simultaneously. There are three kinds of such points. Let us 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 since the edge on which the intersection point is found belongs to only two cubes, only two intersection edges pass through that intersection point. Thus no extra intersection edge passes through that point to be contributed to cover another crack.

2. The intersection point is on the patch face edge, as is 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 $O(n^2)$, where n is the number of intersection edges on a patch face. This depends on the difference between the division levels of the two sides of the patch face.

4 Implementation and results

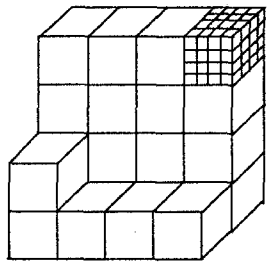
4.1 Space required for crack patching

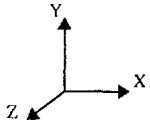
As mentioned in the pseudocode of the AMC in Sect. 2.2, cracks are patched when AMC just finishes processing one initial cube. This is essential for efficient data structure for crack patching. The space for crack-patching information can be shared in the following way. All the information on the faces of initial cubes for crack patching on the x, y plane share the common space, of which size is just large enough to store information for cracks on faces in one slice. All the information on the faces of the initial cubes on the x, z plane share the common space, with a size just large enough for cracks on the faces in one strip. All the information on the faces of the initial cubes on the x, z plane share a common space with a size just large enough for cracks on the face of one initial cube. Finally, all the information on the faces within the initial cubes can share the common space, with a size just large enough for cracks on the faces within one initial cube. As a result, only $O(n^2)$ working memory is needed for crack patching. Figure 14 shows the traversed initial cubes, and Fig. 15 shows the common space for crack patching.

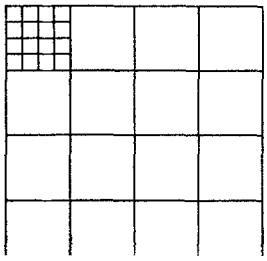
4.2 Results

We implemented both the original MC and our adaptive AMC on an IBM RS-6000/320 workstation. Tables 1–3 show the performance comparisons between them when various volumetric data sets are used and Figs. 16–18 are the corresponding images. Note that the time item includes both the running time for surface extraction and the rendering phases. It can be seen that the average time, as well as the number of triangles, is 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







14 15a 15b 15c 15d

Fig. 14. An example of traveled cubes with an initial cube size of four and a volumetric data set size of 16 x 16 x 16

Fig. 15a–d. The size of the volumetric data set is 16×16×16: **a–c** the common space for cracks on the *x*, *y* *x*, *z*, and *y*, *z* planes respectively; **d** the common space for cracks on faces within the initial cube of size four

Table 1. Performance comparison (skin surface) using the data set of a 256 × 256 × 113 CT scan of the human head. Note: AMC-X denotes the AMC with the initial cube size of X

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

Table 2. Performance comparison (bone surface) using the data set of a 256 × 256 × 113 CT scan of the human head.

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

Table 3. Performance comparison using the data set of a 256 × 256 × 113 CT scan of a machine part

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

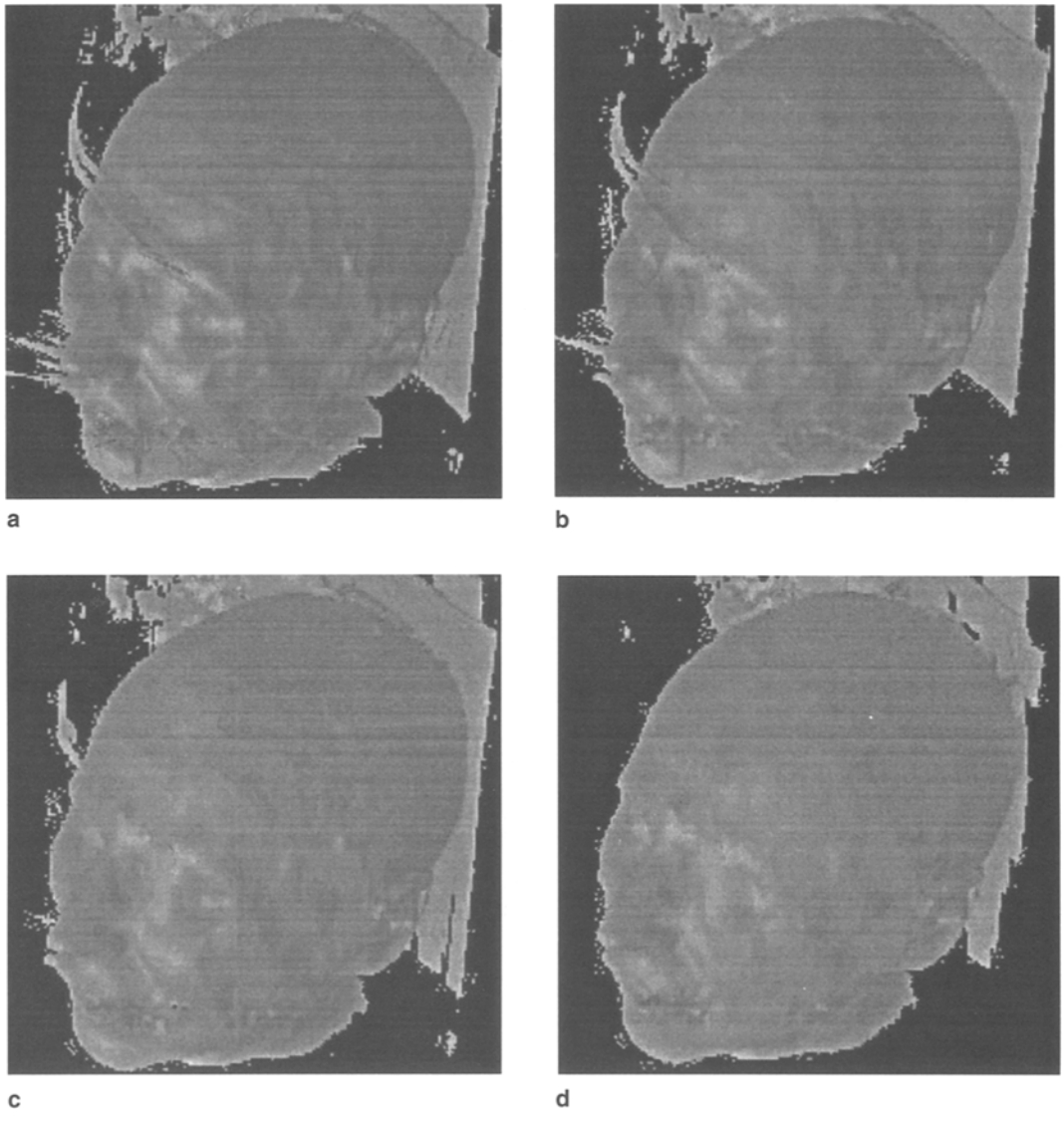


Fig. 16a-d. Skin surface generated by using the data set of a 256×256×113 CT scan of a human head. The algorithms used are: **a** MC; **b** AMC-2; **c** AMC-4; **d** AMC-8

and the volumetric data set of $256 \times 256 \times 113$ CT scan of a human head. It was found that there were about 10000 cracks, but the number of cracks that were visible was much smaller (Fig. 11).

5 Conclusions and discussion

A new algorithm for high-resolution 3D surface construction has been presented. The basic idea is to adapt the size of triangles of representation to

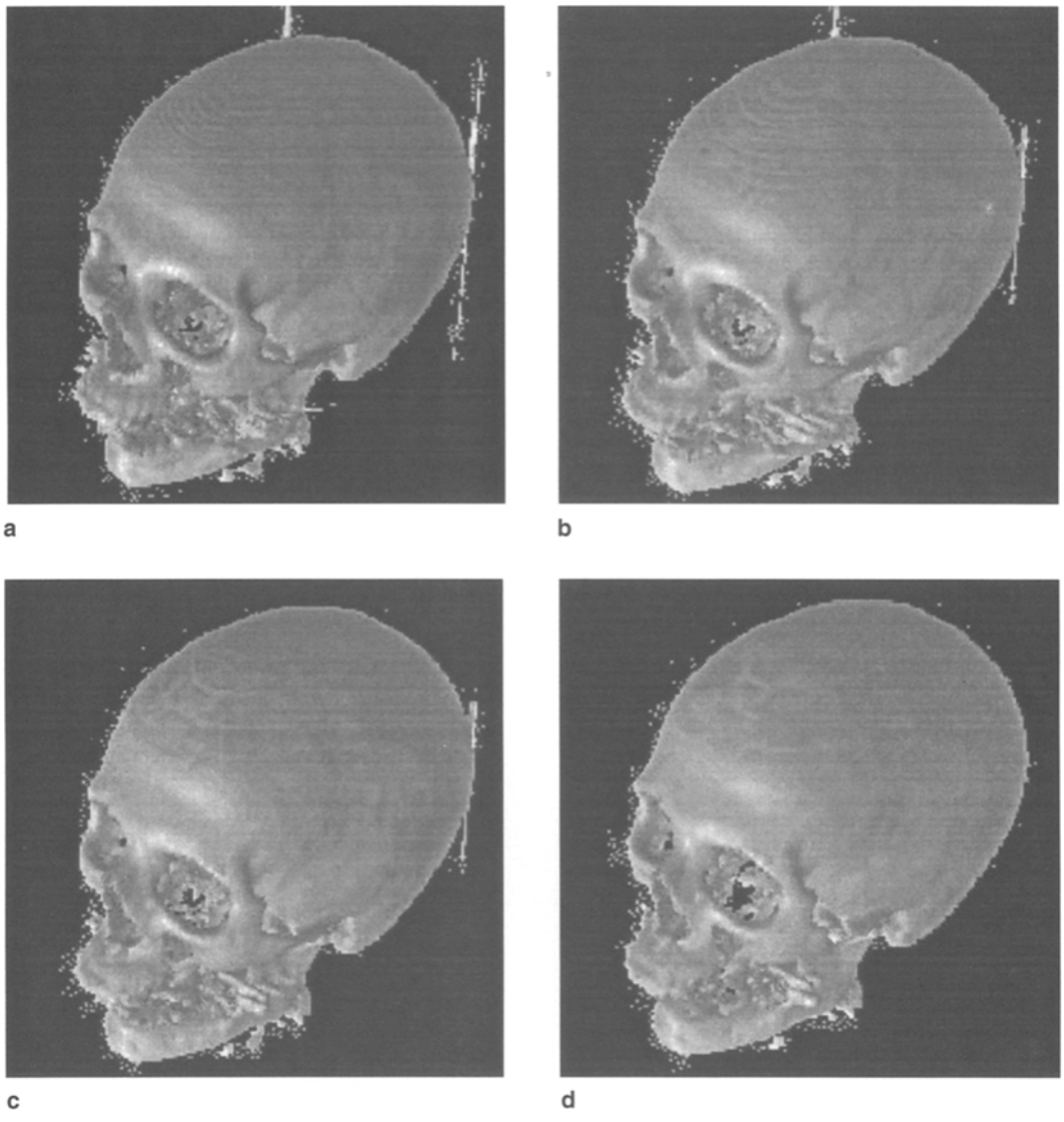


Fig. 17a-d. Bone surface generated by using the data set of a 256×256×113 CT scan of a human head. The algorithms used are: **a** MC; **b** AMC-2; **c** AMC-4; **d** AMC-8

the shape of the surface. The speed of surface construction was improved, and the number of triangles representing the surface was significantly reduced. The latter is essential for interactive manipulation of 3D surfaces.

The images generated by AMC are almost as good as those generated by MC if an initial cube size of two is chosen (Figs. 16b, 17b, and 18b). The result of AMC is better with a relatively smooth surface than with a surface with sharp features. The larger

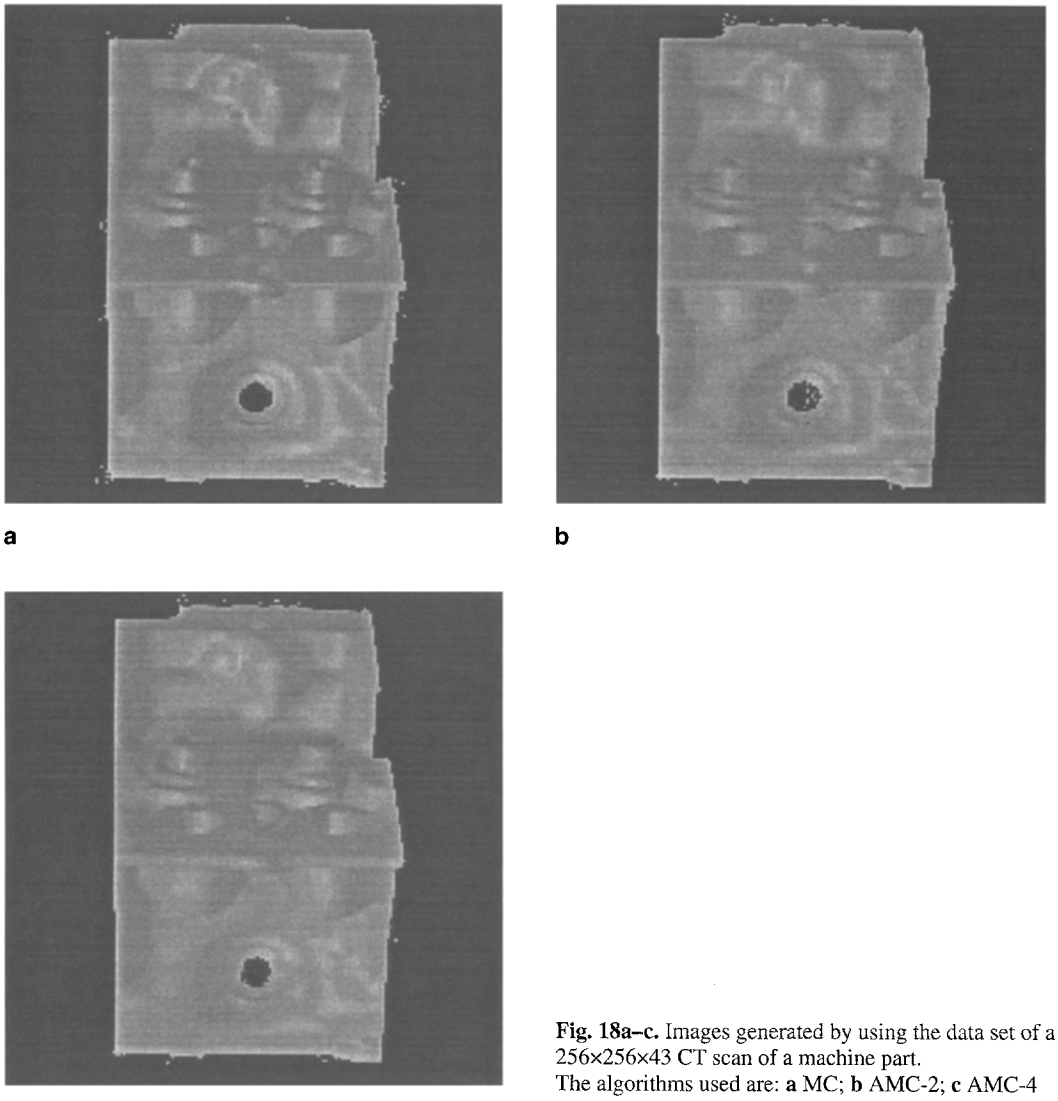


Fig. 18a–c. Images generated by using the data set of a 256×256×43 CT scan of a machine part.
The algorithms used are: **a** MC; **b** AMC-2; **c** AMC-4

the size of the initial cube, the better the result achieved. It is true that the image quality drops with larger sizes of the initial cube (Figs. 16b, c, 17b, c, and 18b). Thus 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 situations. One such scenario occurs 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 cracks of arbitrary size and then reducing the handling of these configurations to a simple rule. We use only $O(n^2)$ working memory for patching the cracks.

In terms of image quality, choosing the initial cube size of two is suggested. With such a choice, the bounded error in the surface is atmost two voxels. Another parameter that can control the image quality is δ, which is related to the angle of two triangle normals (Sect. 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 eight corners of a nonunit cube are checked. A simple solution for this is to combine the octree technique (Wilhelms and Van Gelder 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 nonunit cube.

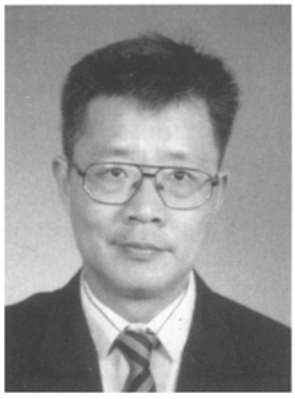
If the isosurface contains real cracks, i.e., the isosurface itself has some cracks, it should not be a problem for our algorithm to obtain the correct representation. This is because our method is not based on the shape of the original surface. Instead, it is based solely on the specific configuration 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, 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, and multientry cubical, AMC produces the lowest polygon count with consistent topology, and belongs to the high implementation complexity range.

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

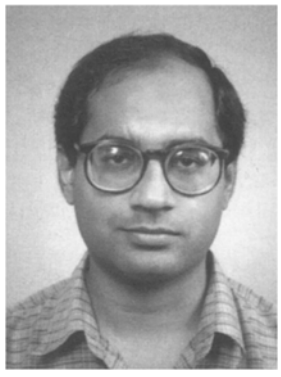
References

1. Artzy E, Frieder G, Herman GT (1981) The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm Comput Graph Image Processing 15:1–24
2. Baker HH (1989) Building surfaces of evolution: the weaving wall. Int J Comput Vision, 3:51–71
3. Chen L, Herman GT, Reynolds RA, Udupa JK (1985) Surface shading in the cuberille environment. IEEE Comput Graph Appl 5:33–43
4. Cline HE, Lorensen WE, Ludke S, Crawford CR, Teeter BC (1988) Two algorithms for the three-dimensional reconstruction of tomograms. Med Phys 15:320–327
5. Clay RD, Moreton HP (1988) Efficient adaptive subdivision of Bézier surfaces. Proceedings of the Eurographics'88 Conference, North-Holland, Amsterdam, pp 357–371
6. Durst MJ (1988) Additional reference to "marching cubes". Comput Graph 22:72–73
7. Fuchs H, Levoy M, Pizer SM (1989) Interactive visualization of 3D medical data. Computer 22:46–51
8. Gordon B, Udupa JK (1989) Fast surface tracking in three-dimensional binary images. Comput Vision Graph Image Processing 29:196–214
9. Herman GT, Liu HK (1979) Three-dimensional display of human organs from computed tomograms. Comput Graph Image Processing 9:1–21
10. Kaufman A (1990) Volume visualization. IEEE Computer Society Press, Los Alamitos, Calif.
11. Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph 21:163–169
12. Muller H and Stark M (1993) Adaptive Generation of surfaces in volume data. Visual Comput 9:182–199
13. Ning P, Bloomenthal J (1993) An evaluation of implicit surface tilers. IEEE Comput Graph Appl 13:33–41
14. Nielson GM, Hamann B (1991) The asymptotic decider: resolving the ambiguity in marching Cubes. Proceedings of Visualization'91 Conference, San Diego, Calif. pp 83–91
15. Schroeder WJ, Zarge JA and Lorensen WE (1992) Decimation of triangle Meshes. Comput Graph 26:65–70
16. Sobierajski L., Kaufman A, Cohen D, Yagel R, Acker D (1993) A fast display method for volumetric data. Visual Comput 10:116–124
17. Trivedi SS, Herman GT, Udupa JK (1986) Segmentation into three classes using gradients. IEEE Trans Med Imaging 5:116–119
18. Turk G (1992) Re-tiling of polygonal surfaces. Comput Graph 26:55–64
19. Wilhelms J, Van Gelder A (1990) Topological considerations in isosurface generation. Comput Graph 24:79–86
20. Wilhelms J, Van Gelder A (1992) Octree for faster isosurface generation. ACM Trans Graph 11:201–227
21. Udupa JK, Hung HM (1990) Surface versus volume rendering: a comparative assessment. Proceedings of the First Conference on Visualization in Biomedical Computing. pp 83–91, IEEE Computer Society Press, Los Alamitos, Calif.



RENBEN SHU received his bachelor's degree (math) at the Fudan University, PRC in 1970. From 1970 to 1983, he was one of the principal system designers at the Fudan University for two computer systems. From 1983 to 1985, as a research fellow he was involved with the University Computer Center and Supercomputer Institute, University of Minnesotan. He was awarded his Ph.D. degree in computer science by the University of Minnesota in 1989. Since 1989, he has been a Member of the Research Staff at the Institute of System Science (ISS), National University of Singapore. He is interested in parallel processing, computer architecture and organization, numerical analysis, scientific visualization, and character recognition. He has published a number of research papers in these areas. He has lead the scientific visualization group at the ISS for two years and is now involved in a off-line hand-written character recognition project at the ISS. He is a member of IEEE.

CHEN ZHOU is currently a masters student at the National University of Singapore (NUS). Prior to joining the NUS in 1992, he worked at an institute in the Aerospace Ministry, Beijing, China. He undertook some projects involving graphics. His current research interests are in scientific visualization. He received his Bachelor's degree in computer science from Tsinghua University, China in 1989.



MOHAN S. KANKANHALLI received his B. Tech degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur, in 1986. He obtained his M.S. and Ph.D. degrees, both in Computer and Systems Engineering, in 1988 and 1990 from Rensselaer Polytechic Institute, Troy, New York. His doctoral work involved developing techniques for designing parallel algorithms for geometric problems. These techniques have been applied to computer graphics, CAD, and solid modeling. Since then, he has been a researcher with the Institute of Systems Science, National University of Singapore. He initially worked with the Visualization Group, helping them design and develop medical applciations. He is now with the Multimedia Group. He is working on projects in graphics modeling and shape computation. His research interests include computer graphics and imaging, geometric algorithms, and parallel algorithms.