COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 64, No. 1, July, pp. 147–156, 1996
ARTICLE NO. 0050

NOTE

A 3D Surface Tracking Algorithm*

XIAOQING QU AND XIAOBO LI

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1

Received April 21, 1994; accepted April 28, 1994

This paper presents a 3D surface tracking algorithm which converts volumetric edge data to a surface model. Existing 3D model generation methods either threshold intensity data, which is sensitive to nonuniform illumination, or trace a surface from multiple layers of edge voxels, where a time-consuming heuristic search is used to extract boundaries. The huge search space prevents using it in 3D. Our algorithm directly constructs a surface model from edge voxels. A voxel is identified as being on the surface if its second derivative is negative and changes sign for neighbors in the gradient direction. By this definition, there can only exist one layer of surface voxels, and the tracking algorithm is simply a breadth-first search. Moreover, the definition of surface voxels is not sensitive to gradient directions, thus this approach is robust against noise. The test results on real data are also reported. © 1996 Academic Press, Inc.

1. INTRODUCTION

In 3D surface model generation, the intensity thresholding method [5, 8] is widely used to extract surfaces. However, global thresholding on intensity data is sensitive to nonuniform illumination. Even with the optimal thresholding method [4], the thresholded boundaries from subimages may be disconnected.

Identifying the surface from edge voxels is a more general approach. Some 3D edge operators based on gradient have been proposed to detect edge voxels [7, 14, 11]. A drawback to these techniques is that they produce multiple layers of surface voxels and a surface model cannot be constructed directly from multiple layers of voxels.

One solution to this problem is to use a heuristic search to find a single layer of surface voxels. In 2D, the classic algorithm by Martelli [10] defines a boundary as the minimum-cost path. In 3D, the algorithm proposed in [3] extracts 2D boundaries using the heuristic search [10] and then stacks 2D boundaries to form a surface. The huge search space prevents using it in 3D applications.

Recall that the directional second derivative of an intensity value has a steep slope around a zero-crossing, which corresponds precisely to the peak point of the gradient magnitude. Marr and Hildreth [9] suggested that zero-crossings can be used to detect edge elements in 2D. However, due to the limited computational accuracy, not many second derivatives have zero magnitudes. If a magnitude threshold is used, the multiple layer problem still exists.

This paper presents a 3D surface tracking algorithm which directly extracts a single layer of surface voxels from edge voxels. The algorithm examines the sign of the second derivative of intensity. The condition for an edge voxel being on a surface is that the second derivative is negative and changes sign for neighbors in the gradient direction. Since there can only exist one layer of surface voxels by this approach, the tracking algorithm is simply a breadth-first search. Compared with the heuristic search, the search space is greatly reduced. This procedure is robust against noise, because the second derivative is evaluated on a Gaussian function which is a lowpass filter. Also the direction of the second derivative does not need to be very precise. Even if the direction is off by a certain angle, the location of the surface voxel is not affected.

To construct a surface model, surface voxels are converted to graphics primitives which are defined in the extended cuberille model [13]. This model has four types of voxels. Each type has a face whose normal coincides with one of 26 edge directions.

Section 2 discusses the surface voxel identification method. Section 3 introduces the surface tracking algorithm and presents the experimental results. The last section gives conclusions and compares the surface tracking algorithm with other algorithms in more detail.

2. SURFACE IDENTIFICATION

This section presents a 3D surface identification method based on the sign of the second derivative of intensity

* This research is supported in part by the Canadian Natural Sciences and Engineering Research Council under Grant OGP9198.

change. We first extend the definition of zero-crossings from 2D to 3D in continuous space, then introduce the condition to identify surface voxels in discrete space.

2.1. Surface Points of Step Intensity Change

This section will show that zero-crossings correspond to a step intensity change.

Suppose there is a step intensity change across an object surface. Let $I(x, y, z)$ be the intensity function, $(x, y, z) \in R^3$, where R is the set of real numbers. Without loss of generality, suppose the intensity change occurs at the origin and in the x direction. It will be shown in the following that a zero-crossing occurs at the origin; thus the origin is the desired surface point.

For a bright object surrounded by a darker background, the intensity around the surface can be modeled as a one-dimensional step change, $I(x, y, z) = c(\pi/2 - \arctan(kx))$ for a sufficiently large constant $k > 0$. The constant c describes the magnitude change and k describes the intensity change rate.

The smoothing filter $G(x, y, z)$ is a Gaussian filter with variance σ. Let $w(x, y, z)$ be the second derivative in the x-direction of the smoothed function $I(x, y, z)$,

$$w(x, y, z) = \frac{\partial^2}{\partial x^2}[G(x, y, z) * I(x, y, z)], \tag{1}$$

that is,

$$w(x) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-x'^2/2\sigma^2}\frac{2ck^3(x - x')}{(1 + k^2(x - x')^2)^2}\,dx', \tag{2}$$

which is a function of x. At $x = 0$, $w(0) = 0$ and

$$w'(0) = \frac{2ck^3}{\sqrt{2\pi}\sigma^3}\int_{-\infty}^{\infty}\frac{x'^2 e^{-x'^2/2\sigma^2}}{(1 + k^2x'^2)^2}\,dx' > 0. \tag{3}$$

Hence $w(x)$ is monotonically increasing in the neighborhood of $x = 0$. In other words, $w(x)$ changes sign from negative to positive as it crosses zero at $x = 0$. $x = 0$ is therefore called a zero-crossing.

Let $v = (x, y, z) \in R^3$ and l be a directed line at v pointing outside of the surface. Writing the above equations in terms of v and l, if v is on the surface, it must satisfy the following conditions:

$$\begin{aligned} w(v) &= 0 \\ w(v - \delta l) &< 0 \\ w(v + \delta l) &> 0. \end{aligned} \tag{4}$$

2.2. Sensitivity of Zero-Crossing Location to Derivative Direction

It is worthwhile to point out that the location of zero-crossings is not sensitive to the direction of the second derivative of an intensity change. Suppose the second derivative is taken in the direction l. The direction cosines of l are $\cos\alpha$, $\cos\beta$, and $\cos\theta$ for some α, β, θ, $<\pi/2$. (2) then becomes

$$w(x) = \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-x'^2/2\sigma^2}\frac{2ck^3(x - x')\cos^2\alpha}{(1 + k^2(x - x')^2)^2}\,dx'.$$

Because $\cos^2\alpha > 0$, the direction of l does not affect the zero location nor the sign of $w(x)$ but only the magnitude of $w(x)$.

It is desirable, though, to take the second derivative in a direction that has a maximum rate of change. It has been shown that the gradient direction is the one in which the second derivative of the intensity function has the maximum change rate [12].

Let the gradient at v be $g(v)$ with the direction g. Equation (1) can be written in terms of v and g,

$$w(v) = G''_g(v) * I(v), \tag{5}$$

and the conditions for a point on a surface become

$$\begin{aligned} w(v) &= 0 \\ w(v - \delta g) &< 0 \\ w(v + \delta g) &> 0. \end{aligned} \tag{6}$$

It should be emphasized that the above procedure is robust against noise and quantization error. First, the second derivative is evaluated on a Gaussian function which is a lowpass filter itself. Second, the direction of the second derivative does not need to be very precise. Even if the direction is off by a certain angle, the location of the surface point is not affected.

2.3. Surface Voxels in Discrete Space

In this section, we first introduce the concept of the *positive layer* and the *negative layer* of voxels, and then define surface voxels in a discrete space.

Let $I(v) \in N$, $N = \{0, 1, 2, \ldots\}$ be an intensity function defined on a discrete domain, $v = (x, y, z) \in Z^3$, $Z = \{0, \pm 1, \pm 2, \ldots\}$. For simplicity, v is called a voxel. Let $G''_g(x, y, z)$ be a finite scale discrete Gaussian filter, resulting from sampling $G''_g(v)$ in (5) in a finite interval $(-3\sigma, 3\sigma)$, and let $w(x, y, z)$ be the discrete convolution of $G''_g(x, y, z)$ with $I(x, y, z)$,

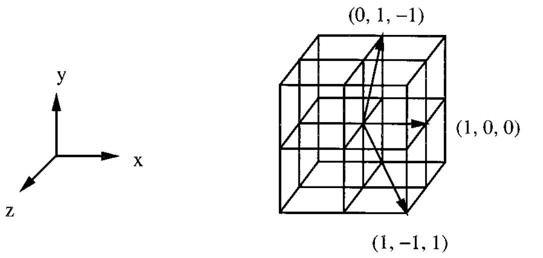

FIG. 1. Three of the 26 gradient directions.

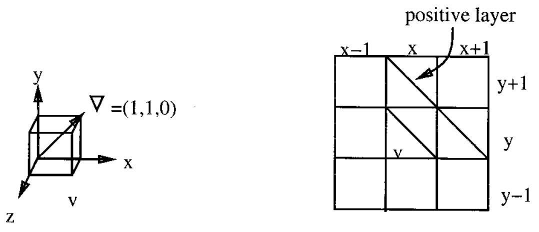

FIG. 2. If the gradient components, ∇_x, ∇_y, of a surface voxel v are nonzero, $w(v)$ has to change sign in both x, y directions.

$$w(x,y,z) = G''_g(x,y,z) * I(x,y,z). \tag{7}$$

If $v = (x, y, z)$ is a surface voxel, $w(x, y, z) = 0$.

The function $w(x, y, z)$ in (7) is defined on voxels of integer coordinates and is undefined between voxels. Because of the discrete nature of voxels, not many voxels have $w(x, y, z) = 0$. Condition (6) however, implies that for a bright object surrounded by a darker background and for those voxels close to a surface, $w(x, y, z)$ is negative inside the object and positive outside. There can only exist one layer of voxels on which $w(x, y, z)$ is negative and changes sign for neighbors in the gradient direction. This layer is called the negative layer. Similarly, there exists exactly one positive layer of voxels. It is possible to define either the negative layer or the positive layer as the surface. Since the negative layer is part of the object, the surface is defined as the negative layer of voxels. Zero-crossing voxels can be treated as either positive or negative and are also included in the surface set. The first condition of (6) therefore becomes $w(x, y, z) \leq 0$.

To express the next two conditions of (6) in the discrete space, it is necessary to determine the neighbor voxels on which the condition needs to be tested. The gradient of $I(v)$ at v is approximated by $\nabla_v = (\nabla_x, \nabla_y, \nabla_z)$ and is quantized to one of 26 directions; see Fig. 1. In each of the directions, voxels are either face connected, edge connected, or vertex connected. If $\nabla_v = (1, 0, 0)$ and v is on the surface of the negative layer, the voxel $(x + 1, y, z)$ is on the positive layer, $w(x + 1, y, z) > 0$, and the voxel $(x - 1, y, z)$ is inside, with $w(x - 1, y, z) < 0$. For $\nabla_v = (1, 1, 0)$, however, if v is on the surface, the voxel $(x + 1, y + 1, z)$ is not on the positive layer but $(x + 1, y, z)$ and $(x, y + 1, z)$ are, as shown in Fig. 2. Therefore it is necessary to verify that $w(x - 1, y, z) < 0$, $w(x, y - 1, z) < 0$, $w(x + 1, y, z) > 0$, and $w(x, y + 1, z) > 0$. These cases reveal that the condition (6) should be tested on face-connected neighbor voxels for every nonzero component of ∇_v.

To summarize, for a bright object surrounded by a darker background, if $v(x, y, z)$ is a surface voxel, it must simultaneously satisfy the following inequalities:

$$\begin{aligned} &w(x,y,z) \leq 0, \\ &w(x-1,y,z)w(x+1,y,z) < 0, \quad \text{if } \nabla_x \neq 0, \\ &w(x,y-1,z)w(x,y+1,z) < 0, \quad \text{if } \nabla_y \neq 0, \\ &w(x,y,z-1)w(x,y,z+1) < 0, \quad \text{if } \nabla_z \neq 0. \end{aligned} \tag{8}$$

Similarly, for a dark object surrounded by a brighter background, the surface voxels can be defined as the positive layer of voxels.

The inequalities (8) is the condition to identify surface voxels in a discrete space. Obviously, there is only one layer of voxels that will satisfy the condition. This makes subsequent surface tracking much easier.

The design of a discrete filter for 26 vectors and computation of $w(x, y, z)$ in a discrete space will not be addressed in this paper, but the interested reader may refer to [12].

2.4. Experimental Results

Figure 3 shows one slice of an experimental result of $w(x, y, z)$ values on test data of volume $40 \times 40 \times 40$. The

```
w(x,y,z) slice y = 21

 \ x 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
 z
 10   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 11   0  0  0  0  0  0  0  0  0  0 -4  0  0  0  0  0  0  0  0  0  0
 12   0  0  0  0  0 -5 -5 -8 -7 -7 -7 -7 -7 -8 -5 -5  0  0  0  0  0
 13   0  0  0  0 -6 -6 -4 -4 -3 -2 -2 -2 -3 -4 -4 -6 -6  0  0  0  0
 14   0  0  0 -6 -6 -4 -1  1  3  4  4  4  3  1 -1 -4 -6 -6  0  0  0
 15   0  0 -5 -6 -4 -1  2  5  8  8  8  8  8  5  2 -1 -4 -6 -5  0  0
 16   0  0 -5 -4 -1  2  5  6  6  0  0  0  6  6  5  2 -1 -4 -5  0  0
 17   0  0 -8 -4  1  5  6  6  0  0  0  0  0  6  6  5  1 -4 -8  0  0
 18   0  0 -7 -3  3  8  6  0  0  0  0  0  0  0  6  8  3 -3 -7  0  0
 19   0  0 -7 -2  4  8  0  0  0  0  0  0  0  0  0  8  4 -2 -7  0  0
 20   0 -4 -7 -2  4  8  0  0  0  0  0  0  0  0  0  8  4 -2 -7 -4  0
 21   0  0 -7 -2  4  8  0  0  0  0  0  0  0  0  0  8  4 -2 -7  0  0
 22   0  0 -7 -3  3  8  6  0  0  0  0  0  0  0  6  8  3 -3 -7  0  0
 23   0  0 -8 -4  1  5  6  6  0  0  0  0  0  6  6  5  1 -4 -8  0  0
 24   0  0 -5 -4 -1  2  5  6  6  0  0  0  6  6  5  2 -1 -4 -5  0  0
 25   0  0 -5 -6 -4 -1  2  5  8  8  8  8  8  5  2 -1 -4 -6 -5  0  0
 26   0  0  0 -6 -6 -4 -1  1  3  4  4  4  3  1 -1 -4 -6 -6  0  0  0
 27   0  0  0  0 -6 -6 -4 -4 -3 -2 -2 -2 -3 -4 -4 -6 -6  0  0  0  0
 28   0  0  0  0  0 -5 -5 -8 -7 -7 -7 -7 -7 -8 -5 -5  0  0  0  0  0
 29   0  0  0  0  0  0  0  0  0  0 -4  0  0  0  0  0  0  0  0  0  0
 30   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

FIG. 3. Sphere slice $y = 21$.

```
w(x,y,z) slice y = 21

\ x  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
z
10    0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
11    0  0  0  0  0  0  0  0  0  0 -4  0  0  0  0  0  0  0  0  0  0
12    0  0  0  0  0 -5 -5 -8 -7 -7 -7 -7 -7 -8 -5 -5  0  0  0  0  0
13    0  0  0  0 -6 -6 -4 -4 -3 -2 -2 -2 -3 -4 -4 -6 -6  0  0  0  0
14    0  0  0 -6 -6 -4 -1  \  |  |  |  |  |  / -1 -4 -6 -6  0  0  0
15    0  0 -5 -6 -4 -1  \  5  8  8  8  8  8  5  / -1 -4 -6 -5  0  0
16    0  0 -5 -4 -1  \  5  6  6  0  0  0  6  6  5  / -1 -4 -5  0  0
17    0  0 -8 -4  \  5  6  6  0  0  0  0  0  6  6  5  / -4 -8  0  0
18    0  0 -7 -3  -  8  6  0  0  0  0  0  0  0  6  8  - -3 -7  0  0
19    0  0 -7 -2  -  8  0  0  0  0  0  0  0  0  0  8  - -2 -7  0  0
20    0 -4 -7 -2  -  8  0  0  0  0  0  0  0  0  0  8  - -2 -7 -4  0
21    0  0 -7 -2  -  8  0  0  0  0  0  0  0  0  0  8  - -2 -7  0  0
22    0  0 -7 -3  -  8  6  0  0  0  0  0  0  0  6  8  - -3 -7  0  0
23    0  0 -8 -4  /  5  6  6  0  0  0  0  0  6  6  5  - -4 -8  0  0
24    0  0 -5 -4 -1  /  5  6  6  0  0  0  6  6  5  \ -1 -4 -5  0  0
25    0  0 -5 -6 -4 -1  /  5  8  8  8  8  8  5  \ -1 -4 -6 -5  0  0
26    0  0  0 -6 -6 -4 -1  /  |  |  |  |  |  \ -1 -4 -6 -6  0  0  0
27    0  0  0  0 -6 -6 -4 -4 -3 -2 -2 -2 -3 -4 -4 -6 -6  0  0  0  0
28    0  0  0  0  0 -5 -5 -8 -7 -7 -7 -7 -7 -8 -5 -5  0  0  0  0  0
29    0  0  0  0  0  0  0  0  0  0 -4  0  0  0  0  0  0  0  0  0  0
30    0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

FIG. 4. Surface voxels of sphere slice $y = 21$.

object is a sphere of radius 7 with 16 values. The sphere has a gray value of 0 and a background of 16. Since the sphere is darker than the background, the surface is chosen to be the positive layer of voxels.

Figure 4 shows the surface voxels, the positive layer of voxels, as identified using condition (8). The surface voxels are labeled $\swarrow$, $\leftarrow$, $\searrow$, and $\downarrow$, which signify the gradient orientations. For instance, for the voxel $x = 23$, $y = 21$, $z = 14$, the function $w(23, 21, 14) = 1$, which is positive. Its gradient is directed to $\swarrow$. To be a surface voxel, its neighbors' w has to change sign in both $\leftarrow$ and $\downarrow$ directions. Examining the horizontal neighbors shows that the w values change sign from negative to positive. The vertical neighbors' w values also change sign from negative to positive. Since $w(23, 21, 14) > 0$ and its neighbor changes sign in the gradient direction, the voxel (23, 21, 14) satisfies condition (8), therefore it is a surface voxel. For those surface voxels with nonzero ∇_y, the w values of their y-neighbors have to change sign as well.

Except for the labeled surface voxel layer, none of the remaining voxels with positive w values has neighbors that change sign in the gradient direction. Thus none of them are on the surface. This clearly shows that there is only one positive layer of surface voxels.

3. THE SURFACE TRACKING ALGORITHM

The surface tracking algorithm extracts a single layer of surface voxels using inequalities (8) and converts surface voxels to graphical primitives to construct a surface model. Because the algorithm traverses one layer of surface voxels, it is a breadth-first search. By redefining the adjacent voxels the search space can be further reduced.

The graphics primitives are defined in the extended cuberille model [13]. We will briefly review the model in the next section.

3.1. The Extended Cuberille Model

Four volume primitives were defined in the extended cuberille model (Fig. 5). Besides a cube, voxels are extended to include three other polyhedra so that each voxel has a face whose orientation is compatible with one of the 26 gradient vectors. This face is termed a face primitive. Hence a volume primitive can be referred to as either a voxel or a face.

An edge voxel is converted to a face primitive by using a one-byte location/direction code, or loc_dir code for short. Since there are 26 edge orientations, the orientation of an edge is recorded in the lower six bits of its loc_dir code with one bit for each x, $-x$, y, $-y$, z, and $-z$ direction. Figure 5 shows the loc_dir code for the four types of faces. Bit six of the loc_dir code is the inside/outside bit, which indicates the voxel is inside or outside the object. Bit 7 of the loc_dir code is used to mark a voxel as visited in the surface tracking algorithm.

A type_1 voxel is always set to inside. A type_2 voxel can be set to either inside or outside. In this implementation, it is set to outside. Since type_3 and type_4 faces have the same normal, they are distinguished by the fact that a type_3 voxel is outside an object and a type_4 voxel is inside. Because a type_4 voxel is inside, bit six of its loc_dir code is set to one.

3.2. Steps to Construct a Surface Model

An example is given in Fig. 6 to demonstrate the process of constructing a surface model. The object is a $4 \times 4 \times 4$ cube as defined by thresholding. According to the inequalities (8), however, the object surface is the set of voxels shown in Fig. 6b. The surface tracking algorithm extracts the surface voxels and converts the voxels to face primitives as shown in Fig. 6c. Because each voxel is converted to one face primitive, the resulting surface in Fig. 6c is not closed. There is one more step to fill in the missing faces to close the surface. In this paper, we only discuss the surface tracking algorithm. Readers interested in the surface closure algorithm may refer to [12].

3.3. The Surface Tracking Algorithm

The surface voxels and adjacency relations between voxels can be interpreted as a graph where nodes are surface voxels and edges are adjacency relations between pairs of voxels. According to the inequalities (8), only one layer of surface voxels exists, so a breadth-first search algorithm [1] can be used to track the surface voxels. The procedure is outlined in Fig. 7.

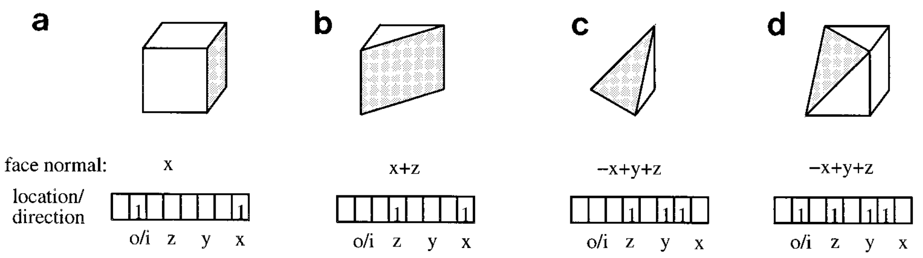

FIG. 5. The location/direction codes for the four volume primitives: (a) Type_1, (b) Type_2, (c) Type_3, (d) Type_4.

The inputs to the algorithm are an array of loc_dir codes `grad_loc_dir`, and an array of `w` values.

The underlying data structure is a queue, where each queue cell has three integers, `x`, `y`, `z`, which are the coordinates of a surface voxel.

In the procedure, `get_start_face` in line 1 reads the coordinates of a start surface voxel, and the procedure `span_start_face()` searches its adjacent surface voxels, marks them as visited and queues them.

The `while` loop in line 4 starts the tracking of surface voxels. A cell is obtained from the queue and is considered the current voxel. In line 9, the procedure `Neighbor_face()` is called to search its adjacent surface voxels. It scans all adjacent voxels, testing if any satisfies the inequalities (8). If a surface voxel is reached and is unmarked, mark it as visited and queue it. After the procedure `Neighbor_face()` returns, `add_to_display_table()` is called to add the current voxel coordinates, `x`, `y`, `z`, and its loc_dir code, `index`, to a display table to use when displaying the face. While the queue is not empty, the loop continues to the next cell in the queue. The `while` loop stops once the queue is empty.

The time taken by `Surface_tracking()` depends on the `while` loop and the procedure call `Neighbor_face()`. Since bit 7 of `grad_loc_dir` is designated for marking, checking and marking a voxel visited can be done in constant time. As a result, the time to execute `Neighbor_face()` depends on the number of adjacent voxels that a surface voxel could have. Since each unmarked surface voxel is placed in the queue once, the `while` loop is executed only once for every surface voxel. Denoting the number of adjacent voxels that a surface voxel has as k, and the total number of the surface voxels as n_B, the time complexity of `Surface_tracking()` is $O(kn_B)$.

For a given object, n_B is determined by inequalities (8) and is set accordingly, but k varies with the number of adjacent voxels that a surface voxel could have. A way to define the adjacency relations between pairs of voxels is by the digital topology [6], where two voxels are adjacent to each other if they are either face, edge, or vertex connected. By this definition, each voxel has 26 adjacent voxels hence the constant k is about 26.

Observe from Fig. 6c, however, that in the extended cuberille model, every surface voxel is converted to a face. Intuitively, a face normal should not have a dramatic change from one surface voxel to an adjacent one because the surface should be smooth. This suggests that the face normal can be used to assist in searching adjacent voxels. In the next section, adjacent voxels and related definitions based on the normals of the four types of faces will be introduced. This results in less than 26 adjacent voxels and reduces the constant k to 14.

3.4. The Definition of Adjacent Voxels

This section introduces definitions of the current voxel and the current face, outways, and adjacent voxels for three face primitives used in the surface tracking algorithm. By this definition, a type_1 face has 24 adjacent voxels, a type_2 face has 22 adjacent voxels, and a type_3 face has 18 adjacent voxels.

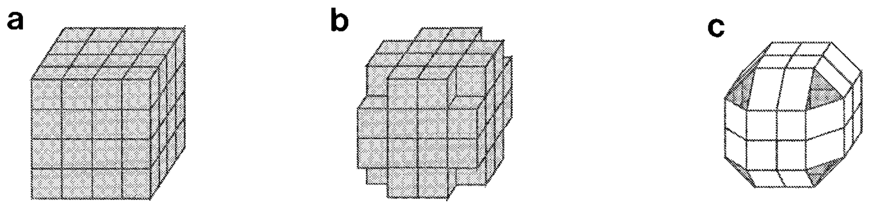

FIG. 6. An example to show the surface construction steps. (a) An object as defined by thresholding, (b) the surface voxels defined by inequalities (8), (c) the surface voxels converted to face primitives as defined in the extended cubrille model.

```
int      x,y,z;                       /*current voxel coordinates*/
Q_CELL   *current_voxel;

Surface_tracking()
{
  unsigned char   index;              /* the table index         */

  get_start_face();                   /* start voxel coordinates */
  span_start_face();                  /* span the start voxel    */

  while(current_voxel == remove_q_head()) {
    x=current_voxel->x;
    y=current_voxel->y;
    z=current_voxel->z;
    index=grad_loc_dir[x][y][z] & ~MARK;
    Neighbor_face(index);               /* search for surface voxels */
    add_to_display_table(x,y,z,index);  /* add to display table    */
  }
}
```

FIG. 7. The surface tracking procedure.

3.4.1. *The Current Voxel and the Current Face*

Recall that the surface tracking algorithm is a breadth-first search and the data structure is a queue of surface voxels. While tracking, the algorithm obtains a voxel from the queue and this voxel is called the *current voxel.* The current voxel is converted to a face by its loc_dir code. This face is called the *current face.* The algorithm then traverses the adjacent surface voxels of the current face by calling procedure `Neighbor_face()`.

3.4.2. *Outways of the Current Voxel*

As shown in Fig. 5, face primitives in the extended cuberille model are either square or triangular, and a face primitive has at most four edges. As a result, there are at most four ways to connect the current face to the next face. Equivalently, there are at most four ways to traverse from the current voxel to the next voxel, and each possibility crosses an edge. The traversal directions are called *outways* of the current voxel. Figure 8 shows the four outways of a type_1 face, the four outways of a type_2 face, and the three outways of a type_3 outside face.

3.4.3. *Adjacent Voxels of an Outway*

Those voxels located in front of an outway may be defined as the *adjacent voxels* of the outway. Searching adjacent voxels can therefore be divided into searching outways. Observe in Fig. 8 that outways of a type_1 face are symmetric. It is sufficient to define adjacent voxels for the first outway. For the other three outways, adjacent voxels can be obtained by 90° rotations of those of the first outway. Similar reasoning applies for a type_3 face. For a type_2 face, outways 1 and 3 are symmetric, and outways 2 and 4 are symmetric, thus it is sufficient to define adjacent voxels for outways 1 and 2.

The definition of adjacent voxels is based on the following assumption of a smooth surface:

ASSUMPTION 3.1. *The face normal difference between the current surface voxel and an adjacent voxel is less than* 90°.

Since tracking surface voxels is the main concern of the algorithm, among the 26 face, edge, or vertex connected voxels, only those voxels potentially on a surface layer are

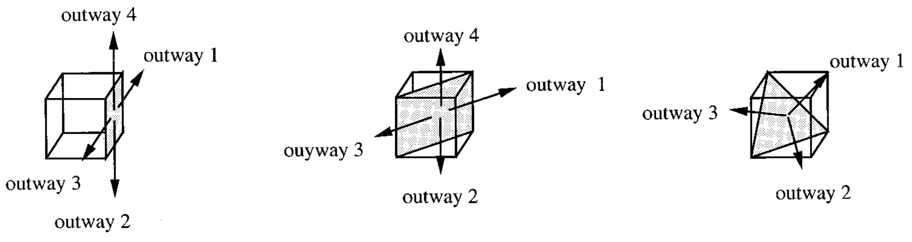

FIG. 8. The four outways of a type_1 face, the four outways of a type_2 face, and the three outways of a type_3 outside face.

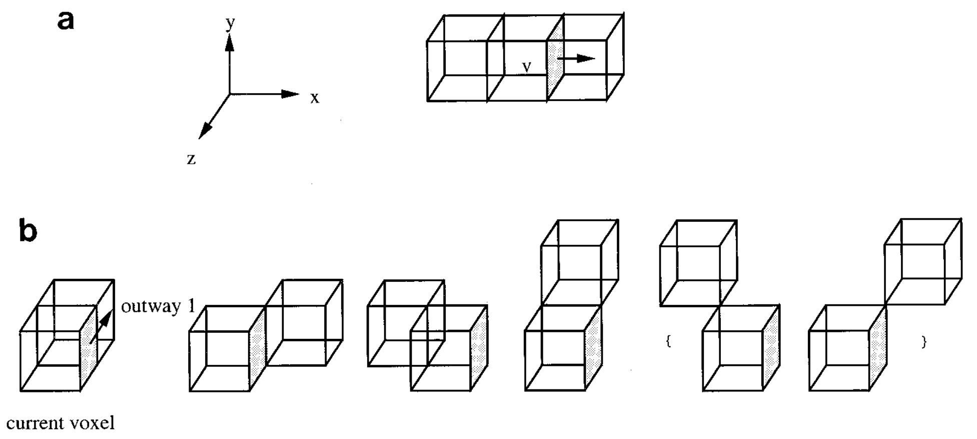

FIG. 9. The adjacent voxels defined for outway 1 of a type_1 face. (a) A type_1 surface voxel v. The voxel to the right of v is on the positive layer, and the left voxel is inside. (b) Six adjacent voxels defined for outway 1 of a type_1 face.

defined as adjacent voxels. It will be shown that a type_1 face has 24 adjacent voxels, a type_2 face has 22 adjacent voxels, and a type_3 face has 18 adjacent voxels.

Suppose voxel v is a type_1 surface voxel which is on the negative layer as shown in Fig. 9a. According to the inequalities (8), the voxel to the right of v has a positive w value, and the voxel to the left of v has a negative w value. If the left voxel is on the surface, the voxel v would have $w(v) > 0$ by Assumption 3.1. This contradicts the fact that voxel v is on the negative layer surface, hence the left voxel is inside. This leaves 24 face, edge, and vertex connected voxels. Because the four outways of a type_1 face are symmetric, the 24 voxels are divided into four disjoint sets, with one for each outway. Figure 9b shows the adjacent voxels defined for outway 1 of the type_1 face.

For a type_2 surface voxel v, see Fig. 10a, if v is on a surface of a negative layer, voxels $v1$ and $v2$ are on the positive layer. Voxels $v3$ and $v4$ are not on the surface by Assumption 3.1. This leaves 22 face, edge, and vertex connected voxels. Dividing them into four disjoint sets for four outways results in four adjacent voxel sets. Figures 10b and 10c show the adjacent voxels defined for outway 1 and outway 2 of the type_2 face.

For a type_3 surface voxel v, see Fig. 11a, if v is on a surface of a negative layer, v_1 v_2, and v_3 are on the positive layer. By the same argument, voxels v_4 to v_8 are not on the surface. This leaves 18 face, edge, and vertex connected voxels that are divided into three adjacent voxel sets, with one for each outway. Figure 11b shows the six adjacent voxels for outway 1 of the type_3 face.

The adjacent voxels of each outway of the three faces are in turn divided to subsets. A subset with more than one voxel is enclosed by braces. During surface face tracking, adjacent voxels of a subset will be searched as a unit.

3.5. The Table Structure

A type_1 face has six orientations. A type_2 face has 12 orientations, and a type_3 face 8 orientations. The adjacent voxels for any face orientation can be obtained by 90° rotations of those in Figs. 9 to 11. The rotation matrices, the adjacent voxel coordinates, and related information is stored in a table structure. The `table` has 128 entries. The seven-bit table index corresponds to the loc_dir code and is accessible by a face's loc_dir code. The `table` structure and the implementation details are given in [12].

3.6. Complexity Analysis

From the analysis given above, the time complexity of the surface tracking algorithm is $O(kn_b)$, where n_b is the number of surface voxels. The constant k depends on the number of adjacent voxels that a surface voxel has and the method with which the procedure `Neighbor_face()` searches the adjacent voxels for surface voxels. As discussed in the previous section, the search neighborhood of the current voxel is split into search outways. Adjacent voxels of an outway are separated into subsets and these subsets are searched in a given order. Whenever a subset has been scanned and at least one surface voxel has been found, the search stops and proceeds to the next outway.

If the possibility that an adjacent voxel is a surface voxel is uniformly distributed, for the type_1 face whose adjacent voxels are shown in Fig. 9, the average time to search an outway is approximately $\frac{1}{6}(1 + 2 + 3 + 4) + \frac{2}{6} \times 6 = 3\frac{2}{3}$. The total time to search four outways is approximately 14.33. For the type_2 face whose adjacent voxels are shown in Fig. 10, the average search time is approximately 15.50. For a type_3 face, it is approximately 12.50. Therefore, the constant k is $(14.33 + 15.50 + 12.50)/3 \approx 14$. To summarize, the time complexity of the `Surface_tracking()` procedure is $O(kn_B)$, where n_B is the number of surface

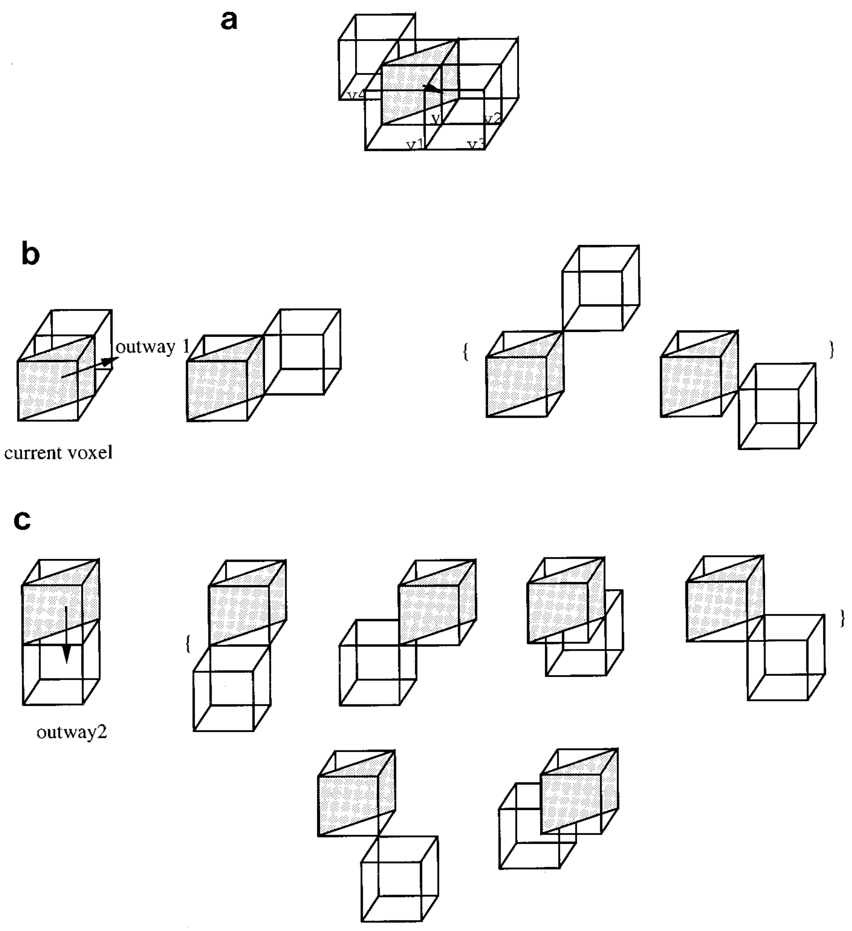

FIG. 10. The adjacent voxels defined for a type_2 face. (a) A type_2 surface voxel v. Voxels $v1$ and $v2$ are on the positive layer. Voxel $v3$ and $v4$ are not on the surface. (b) Four adjacent voxels of outway 1 of the type_2 face. (c) Seven adjacent voxels for outway 2 of the type_2 face.

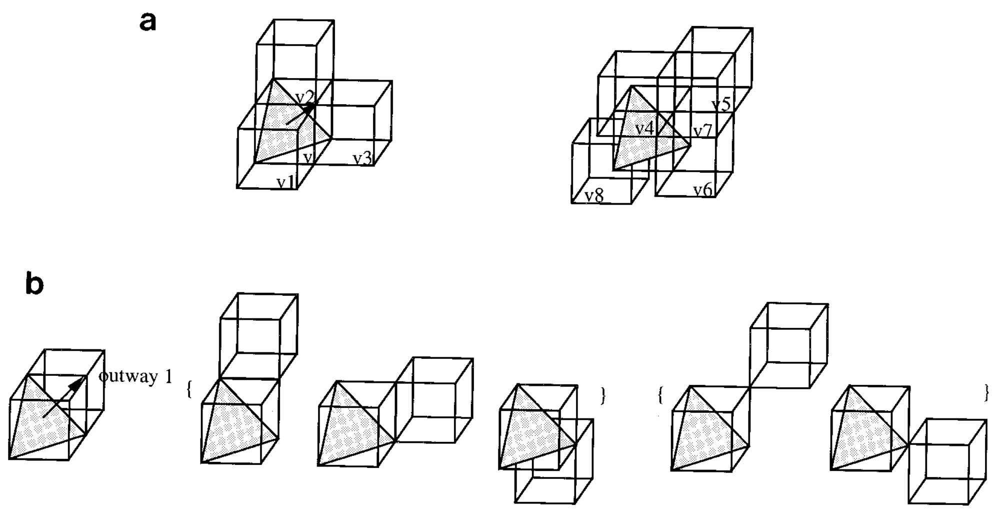

FIG. 11. Adjacent voxels defined for a type_3 face. (a) A type_3 surface voxel v. Voxels $v1$, $v2$, and $v3$ are on the positive layer. Voxels $v4$ to $v8$ are not on the surface. (b) Six adjacent voxels for outway 1 of the type_3 face.

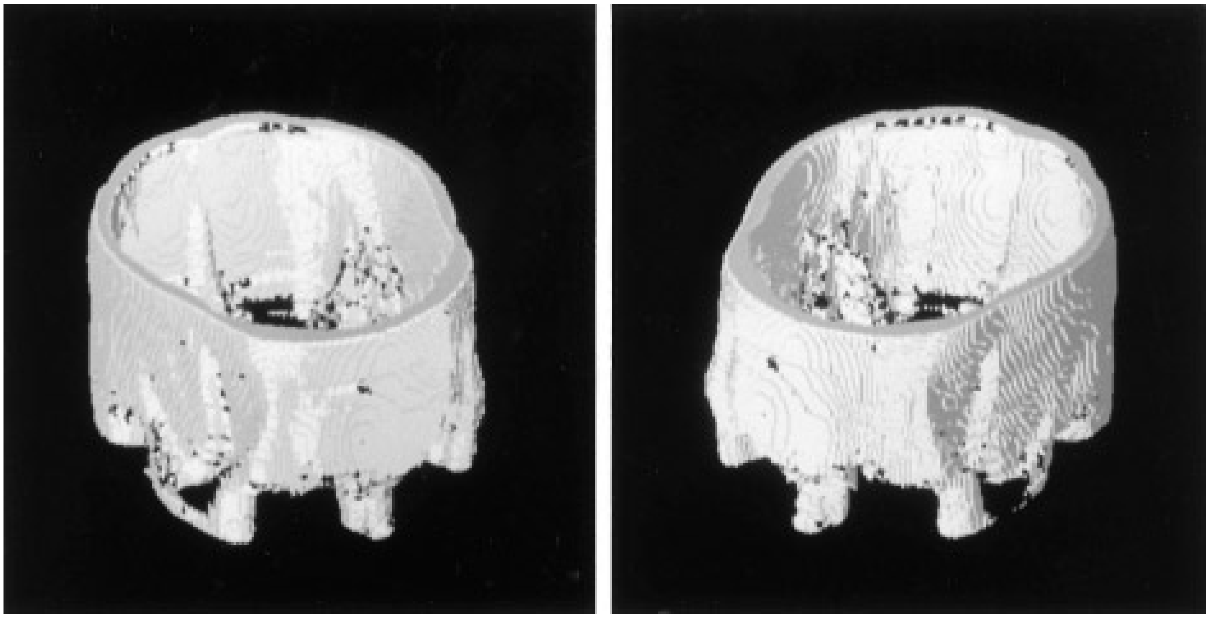

FIG. 12. A medical object from two viewpoints.

voxels of a given object and the constant k is approximately 14.

3.7. The Result of Surface Tracking

Figure 12 shows the surface of a medical object and the data size is $208 \times 208 \times 79$. The gray values of the data set, originally ranged from -1024 to 1660, were linearly mapped to the range 0 to 255. Executing the surface tracking algorithm results in 84,708 surface voxels.

4. CONCLUSION

This paper presents a 3D surface tracking algorithm which converts volumetric edge data to a surface model. A surface is defined as either the *positive* or the *negative* layer of voxels, which are identified by the inequalities (8). By this definition, there can only exist one layer of surface voxels, and the tracking algorithm is simply a breadth-first search. By redefining adjacent voxels, each surface voxel has less than 26 adjacent voxels, the algorithm can be even faster. This approach is relatively stable against noise. Since the second derivative is evaluated on a Gaussian function which is a lowpass filter, a minor change in the direction of the second derivative does not affect the locations of the surface voxels.

There are basically two ways to generate a surface model, by thresholding intensity or by tracking edge voxels generated by edge detection. The thresholding approach produces a single layer of surface voxels, but it is sensitive to nonuniform illumination. Identifying surface voxels by edge detection is more general. The surface tracking algorithm in [2, 5] and the marching cubes algorithm in [8] both use thresholding. These identification approaches are restricted compared with ours.

Edge detection generates multiple layers of surface voxels, which cannot be used to directly construct a surface model. The algorithm in [3] uses a heuristic search to extract a surface, but this search is extremely time consuming since it involves a huge search space. Our algorithm extracts a single layer of surface voxels and uses a simple breadth-first search, thus the time complexity is only of $O(kn_B)$, where n_B is the number of surface voxels of a given object and the constant k is approximately 14. Our algorithm is therefore significantly faster than a heuristic search.

The surface tracking algorithm in [2, 5] also traverses one layer of surface voxel extracts by thresholding, but it must traverse six faces of a surface voxel. Our algorithm traverses one face per surface voxel, thus it is once again faster.

Since our algorithm converts each surface voxel to a face, and the resulting surface may not be closed, a surface closure algorithm can be used to fill missing faces. The interested reader may refer to [12].

REFERENCES

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison–Wesley, Reading, MA, 1983.
2. E. Artzy, G. Frieder, and G. T. Herman, The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm, *Comput. Vision Graphics Image Process.* **15,** 1981, 1–24.
3. J. D. Cappelletti and A. Rosenfeld, Three-dimensional boundary

following, *Comput. Vision Graphics and Image Process.* **48,** 1989, 80–92.

4. R. C. Gonzalez and P. Wintz, *Digital Image Processing*, Addison–Wesley, Reading, MA, 1987.
5. D. Gordon and J. K. Udupa, Fast surface tracking in three-dimensional binary images, *Comput. Vision Graphics Image Process.* **45,** 1989, 196–214.
6. T. Y. Kong and A. Rosenfeld, Digital topology: Introduction and survey, *Comput. Vision Graphics Image Process.* **48,** 1989, 357–393.
7. H. K. Liu, Two- and three-dimensional boundary detection, *Comput. Vision Graphics Image Process.* **6,** 1977, 123–134.
8. W. E. Lorensen and H. E. Cline, Marching cubes: A high resolution 3d surface construction algorithm, *ACM Comput. Graphics* **21**(4), 1987, 163–169.
9. D. Marr and E. Hildreth, Theory of edge detection, *Proc. of Royal Soc. London* **207,** 1980, 187–217.
10. A. Martelli, An application of heuristic search methods to edge and contour detection, *Commun. ACM* **19**(2), 1976, 73–83.
11. D. G. Morgenthaler and A. Rosenfeld, Multidimensional edge detection by hypersurface fitting, *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-3**(4), 1981, 187–217.
12. X. Qu, Identification and display of 3D objects from 3D gray value data, Ph.D. thesis, University of Alberta, 1993.
13. X. Qu and W. Davis, An extended cuberille model for identification and display of 3d objects from 3d gray value data, *Proceedings, Graphics Interface '92*, 1992.
14. S. W. Zucker and R. A. Hummel, An optimal three-dimensional edge operator, *Proceedings, IEEE Computer Science Conference on Pattern Analysis and Image Processing, 1979*, pp. 162–168.