COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 63, No. 2, March, pp. 388–393, 1996
ARTICLE NO. 0028

NOTE

An Algorithm for a 3D Simplicity Test

LONGIN LATECKI[1]

Department of Computer Science, University of Hamburg, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany

AND

C. MIN MA[2]

Department of Industrial Engineering and Management, MingChi Institute of Technology, Taipei, Taiwan, 243

Received May 5, 1994; accepted December 16, 1994

Every thinning (or shrinking) algorithm can be stated as a list of deletion configurations. If a neighborhood of a given image point matches one of these configurations, the point will be deleted. In order to make design of a (3D) thinning (or shrinking) algorithm easy to handle, and therefore the list of configurations as short as possible, every configuration is described in three colors, black, white, and ''don't-care,'' where the don't-care ''color'' matches either a black or white point in a given image. A thinning algorithm is connectivity preserving if and only if every set deleted by this algorithm can be ordered in such a sequence that every point is simple after all previous points in the sequence are deleted. Therefore, it is very important to determine whether a deleted point is simple or not. We present the first 3D algorithm for determining the simplicity of any three-color deletion configuration. This algorithm is memory efficient and fast. Hence it can be a very useful tool for designing 3D connectivity preserving thinning (or shrinking) algorithms.

1. INTRODUCTION

Many operations on 2D and 3D images are required to preserve connectivity, that is, there is a one-to-one correspondence between (black and white) components of the input and output image and their structure. For example, an object with a structure of ''8'' cannot be transformed into an object with a structure of ''o'' or ''l''. Thinning (or shrinking) is a kind of transformation for which connectivity must be preserved. A number of 3D thinning algorithms were proposed (see [1, 8, 9, 13, 14]). All of them require that only simple points can be deleted.

According to Kong [4], a thinning algorithm T is connectivity preserving if and only if every set deleted by T can be ordered in such a sequence that every point is simple after all previous points in the sequence are deleted. To prove a parallel thinning algorithm preserves connectivity, one must show it preserves connectivity for all possible images. But it could be difficult to check all images, since there are too many possible images. Ronsenfeld [11], Ronse [10], and Hall [3], proposed different techniques to simplify the proof that a 2D thinning algorithm preserves connectivity. Hall [2] and Ma [6] proposed corresponding 3D techniques, where Hall's 3D results were extended from his 2D results, and Ma's 3D results were based on Ronse's 2D results. All these techniques require that if a black point is deleted, the deletion of such a point does not change the connectivity of the image. Therefore, while designing a 3D thinning (or shrinking) algorithm, one has to take care that every point deleted by the algorithm is simple.

Every deletion condition can be stated as a list of deletion configurations, where if a neighborhood of a given image point matches one of these configurations, the point will be deleted. In order to make design of a (3D) thinning (or shrinking) algorithm easy to handle, and therefore the list of configurations as short as possible, every configuration is described in three colors, black, white, and ''don't-care,'' where the don't-care ''color'' matches either black or white point in a given image.

We present an algorithm which computes whether a given three-color deletion configuration is such that every image point matching it is simple. The task is not trivial due to the number of such configurations in the 3D case. There are 26 neighbors which have to be considered and each of them can have one of the three colors, therefore,

[1] E-mail: latecki@informatik.uni-hamburg.de.
[2] E-mail: jctt048@twnmoe10.edu.tw.

1077-3142/96 $18.00

we obtain 3^{26} different 3D configurations. The fastest procedure for the simplicity test uses a lookup table. A lookup table is an array in which each entry is a Boolean value (simple or nonsimple). With a lookup table, there is an associated function which assigns to every three-color configuration a unique number identifying the corresponding entry in the lookup table. Given a neighborhood configuration of some point p, the function computes the number of the entry and retrieves the Boolean value which signifies the simplicity of p in this neighborhood configuration. This procedure is very fast. However, the lookup table has to contain $3^{26} \approx 2.54 \times 10^{12}$ entries. Since each entry contains a Boolean value, we can use one bit for each entry. Thus the table uses about 2540/8 ≈ 318 Gbytes, which certainly is not feasible. Although our algorithm is also based on table lookup, the combined size or our lookup tables is only about 1.4 Mbytes.

2. DEFINITIONS

In $\mathbb{Z}^3$, two points p, q are *diagonally adjacent* if they are 18- but not 6-adjacent, and *diametrically adjacent* if they are 26- but not 18-adjacent (all definitions of n-adjacency can be found in [5]). $N_k(p)$ denotes the set containing p and all points k-adjacent to p and $N_k^*(p)$ denotes $N_k(p) - \{p\}$. $N_{26}(p)$ is also referred to as $N(p)$ and called the *neighborhood* of p, whereas $N_{26}(p) - \{p\}$ is referred to as $N^*(p)$. The points in $N(p)$ (or $N^*(p)$) which are diametrically adjacent to p are called *corner points*.

A *binary digital image* (or briefly an *image*) P is an ordered pair $(\mathbb{Z}^3, B)$, where B is a subset of $\mathbb{Z}^3$. Every point in B is called a *black point* and assigned value 1; every point in $\mathbb{Z}^3 - B$ is called a *white point* and assigned value 0. Two black points are *adjacent* if they are 26-adjacent, while two white points, or one black point and one white point are *adjacent* if they are 6-adjacent.

We will also consider *don't-care points*, which are assigned value 2. A don't-care "color" means that the point will match either a white or black color. We represent a *thinning* (*or shrinking*) *algorithm* as a list of deletion configurations in which each configuration is given as a neighborhood of its central point. The color of each point in the configuration can be either white, black, or don't-care. If a neighborhood of a black image point matches one of the configurations, the point will be deleted, i.e., its color will be changed from black to white. The definition of a simple point is as follows.

DEFINITION 2.1 ([12, 7]). Let p be a black point of a 3D image. Then p is said to be *simple* if and only if:

C1. p is (26-)adjacent to only one black (26-)component in $N^*(p)$ and

C2. p is (6-)adjacent to only one white (6-)component in $N_{18}(p)$.

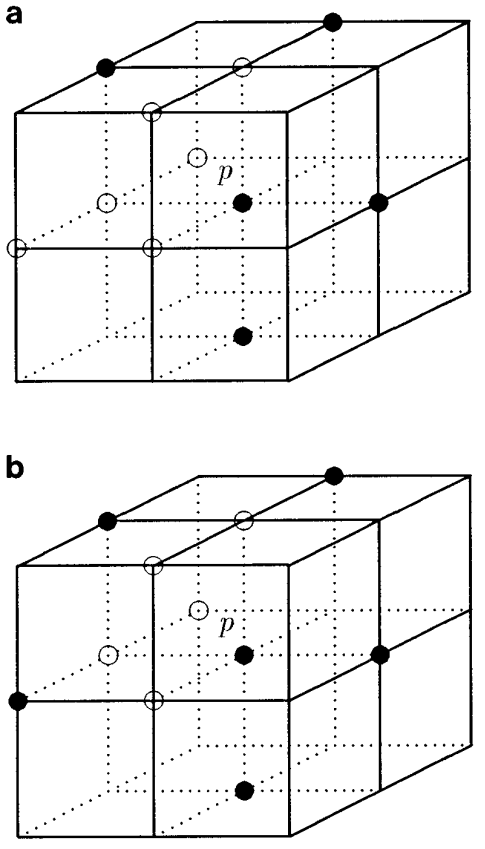

FIG. 1. In both (a) and (b), a point marked ● is black, and a point marked ○ is white. Every unmarked point is a don't-care point. Note that p is simple in configuration (a) but nonsimple in (b).

Let U be a configuration of three-color points (not necessarily the 26-neighborhood). An *assignment* for U is a function which assigns white (value 0) or black (value 1) to each don't-care point in U.

Based on Definition 2.1, we state what it means for a point p to be simple in a three-color neighborhood.

DEFINITION 2.2. Let $N(p)$ be a three-color neighborhood configuration of a black point p, i.e., each point in $N^*(p)$ is either white, black, or don't-care. Then p is said to be *simple* if and only if p is simple in the sense of Definition 2.1 for every assignment for $N(p)$.

The two configurations in Fig. 1 show that it is not that easy to determine whether a point p is simple or not in a three-color neighborhood.

3. A SPACE-SAVING ALGORITHM FOR THE SIMPLICITY TEST

Different approaches for finding efficient ways to determine the simplicity of a black point p in a binary image have been proposed. These approaches can also be extended to

determine whether a given three-color deletion configuration is such that every image point matching it is simple. For the 2D case, two major approaches are as follows.

• Computation of the number of black components in $N^*(p)$, which can be accomplished using Rutovitz and Hilditch crossing numbers (see [5]).

• Retrieval from a lookup table. Since in the 2D case the simplicity of a point depends only on its eight neighbors, each having one of the three possible colors, a lookup table with $3^8 = 6561$ entries is sufficient to hold all possible configurations. Each entry of the lookup table contains a Boolean value which indicates the simplicity of the configuration tested.

The second approach is faster than the first one, since it only requires calculation of the number of the position in the lookup table assigned to a given configuration and retrieving the Boolean value of the entry with this number. The lookup table occupies only $6561/8 \approx 820$ bytes (if each entry occupies one bit).

Our 3D test follows the second approach. However, memory space is a serious difficulty in extending this approach to the 3D case. For the same test in the 3D case, 26 points need to be considered, each having one of the three colors, so that a lookup table with $3^{26} \approx 2.54 \times 10^{12}$ entries would have to be constructed. If we use one bit for each entry, then the table will occupy about $2{,}540/8 \approx 318$ Gbytes, wich certainly is not feasible.

Here we present an algorithm for the simplicity test which uses a lookup table of size 1.4 Mbytes. The strategy of the test is the following: Given a neighborhood $N(p)$ of a black point p, we try to find an assignment for don't-care points which makes C1 or C2 fail. If such an assignment exists, p is nonsimple. If this is not possible, then C1 and C2 are satisfied, and therefore p is simple.

A Space-Saving Algorithm for the Simplicity Test

```
function simple(N(p)): Boolean;
begin
    if not C1(N(p)) or not C2(N18(p)), then return(false) else return(true);
end;
(* C1(N(p)) is true if and only if p is adjacent to only one black component in N*(p) *)
function C1(N(p)): Boolean;
begin
    (* Test 1.1. p has a black 6-neighbor *)
    if p has a black 6-neighbor q then
    begin
        Let Nq(p) be N(p) − ({p, q} ∪ { x:x is contained in the layer containing q which spans a plane perpendicular to line segment pq (the 8 unmarked points in Fig. 2) });
```

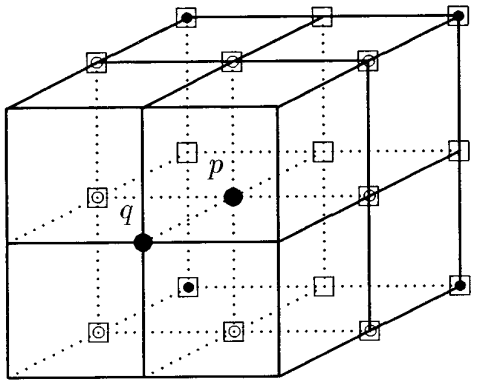

FIG. 2. If any point marked ◙ is a don't-care point, it will be converted into a black point. If any point marked ◙ is a don't-care point, it will be converted into a white point. The eight unmarked points are ignored.

```
        Let N'q(p) be Nq(p) with all don't-care corner points in N(p) that are not 26-adjacent to q (marked ◙ in Fig. 2) assigned black and all don't-care points in Nq(p) that are 26-adjacent to q (marked ◙) assigned white;
        return(Check-Table 1.1(N'q(p))); (* Function Check-Table 1.1 is defined below. *)
end;
(* Test 1.2. No 6-neighbor of p is black, but p has a black diagonal neighbor *)
if p has a black diagonal neighbor q then
begin
    Let Nq(p) be N(p) as illustrated in Fig. 3 without p, q and without 4 points marked ○ and two corner (unmarked) points of N(p) which are both 6-adjacent to q;
    Let N'q(p) be Nq(p) with all don't-care corner
```

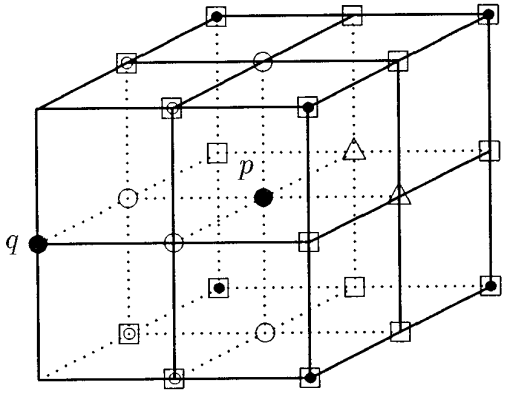

FIG. 3. If any point marked ◙ is a don't-care point, it will be converted into a black point. If any point marked ◙ is a don't-care point, it will be converted into a white point. The four points marked ○ are white points. Two points marked △ are either white or don't-care. The two unmarked points are ignored.

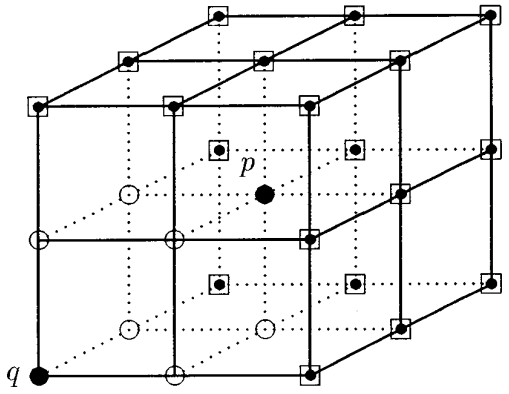

FIG. 4. The six points marked ○ are either white or are don't-care points. If any point marked ◙ is a don't-care point or is black, then C1($N(p)$) is false.

points in $N(p)$ that are not 26-adjacent to q (marked ◙ in Fig. 3) assigned black and all don't-care points in $N_q(p)$ that are 26-adjacent to q (marked ◙) assigned white;
return(Check-Table 1.2($N'_q(p)$)); (* Function Check-Table 1.2 is defined below. *)
end;
(* Test 1.3. No 18-neighbor of p is black, but p has a black diametrical neighbor *)
if p has a black diametrical neighbor q then
begin
(*If any point marked ◙ in Fig. 4 is black or don't-care, p is nonsimple *)
if (Σ values of all points in $N(p)$ that are not 26-adjacent to q) > 0
then return (false) else return (true);
end;
(* Test 1.4. No 26-neighbor of p is black *)
return (false);
end;

(* C2($N(p)_{18}$) is true if and only if p is adjacent to only one white component in $N_{18}(p)$ *)
function C2($N_{18}(p)$): Boolean;
begin
(* Test 2.1. p has a white 6-neighbor *)
if p has a white 6-neighbor q then
begin
Let $N'_{18}(p)$ be $N_{18}(p)$ with all diagonal don't-care neighbors of p (marked ◙ in Fig. 5) assigned black;
return(Check-Table 2($N'_{18}(p)$)); (* Function Check-Table 2 is defined below. *)
end;
(* Test 2.2. No 6-neighbor of p is white *)
return(false);
end;

Declarations of Check-Table Functions and Lookup Table Descriptions

Check-Table 1.1 is a Boolean function. The input $N'_q(p)$ illustrated in Fig. 2 is as described in the algorithm. Check-Table 1.1 ($N'_q(p)$) returns "true" if there is only one black component in $N'_q(p)$ adjacent to q for every assignment and it returns "false" otherwise. The function gives the output based on a lookup table *Table 1.1*. This table contains only $2^4 \times 2^8 \times 3^5 = 995,328$ entries, since 4 points marked ◙ as well as 8 points marked ◙ are black or white, and only the remaining 5 points marked □ can have one of the three different colors (see Fig. 2).

The function Check-Table 1.2 works similarly as Check-Table 1.1. Note, however, that Check-Table 1.2 will only be applied if none of 6-neighbors of p is black.

Check-Table 1.2 is a Boolean function. The input $N'_q(p)$ illustrated in Fig. 3 is as described in the algorithm. Check-Table 1.2 ($N'_q(p)$) returns "true" if there is only one black component in $N'_q(p)$ adjacent to q for every assignment and it returns "false" otherwise. The function gives the output based on a lookup table *Table 1.2*. This table contains only $2^6 \times 2^4 \times 2^2 \times 3^7 = 8,957,952$ entries, since (1) six corner points marked ◙ are black or white, (2) four points marked ◙ are black or white, (3) two points marked △ are either white or don't-care, and (4) only the remaining seven points marked □ can have one of the three different colors (see Fig. 3).

Check-Table 2 is a Boolean function. The input $N'_{18}(p)$ illustrated in Fig. 5 is as described in the algorithm. Check-Table 2 ($N'_{18}(p)$) returns "true" if there is only one white 6-component in $N'_{18}(p)$ 6-adjacent to q for every assignment and it returns "false" otherwise. The function gives the output based on a lookup table *Table 2*. This table contains only $2^{12} \times 3^5 = 995,328$ entries, since 12 points

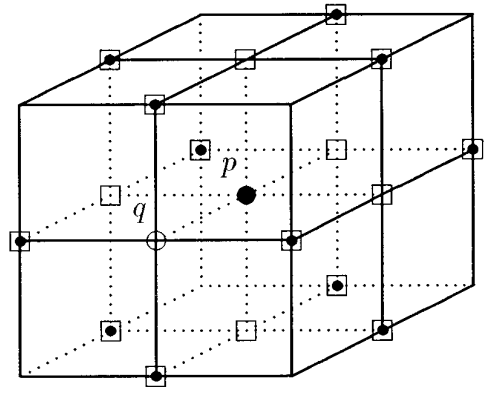

FIG. 5. If any point marked ◙ is a don't-care point, it will be converted into a black point. A point marked □ is either black, white, or don't-care.

marked ◉ are black or white, and only the remaining 5 points marked □ can have one of the three different colors (see Fig. 5).

Memory Requirements

Note that on this approach, the amount of space needed for the simplicity test has been greatly reduced. We need only three lookup tables which together have 10,948,608 entries. Thus the size of the three tables together is about $10{,}948{,}608/8 \approx 1.4$ Mbytes if one bit for each entry is used.

4. VERIFICATION

The strategy of the test is the following: Given a neighborhood $N(p)$ of a black point p, we try to find an assignment for don't-care points which makes C1 or C2 fail. If such an assignment exists, p is nonsimple. If this is not possible, then C1 and C2 are satisfied, and therefore p is simple. For C1, we try to find out whether there can be two black components in $N^*(p)$. There are two black components in $N^*(p)$ if and only if there are two black points in $N^*(p)$ that cannot be joined by a black 26-path in $N^*(p)$.

We illustrate the idea of the proof for Test 1.1. In Test 1.1, p has a black 6-neighbor q. p fails Tests 1.1 if and only if there is a black point in $N^*(p)$ which cannot be joined to q by a black 26-path in $N^*(p)$. If p fails Test 1.1 in $N'_q(p)$, then p fails Test 1.1 in $N(p)$, because an assignment for $N(p)$ is found which makes C1($N(p)$) false. If p passes Test 1.1 in $N'_q(p)$, then p passes Test 1.1 for every possible assignment of $N(p)$, by Lemmas 4.3, 4.4, and 4.5.

THEOREM 4.1. *The algorithm for the simplicity test is correct, i.e., function simple $N(p)$ returns true if and only if p is simple in $N(p)$.*

Proof. It follows from Propositions 4.2 and 4.6. ■

PROPOSITION 4.2. *The test C1 is correct, i.e., function C1($N(p)$) returns true if and only if p satisfies condition C1 in $N(p)$.*

Proof. First we show that Test 1.1 is correct. By Lemma 4.3, the color of eight unmarked points in Fig. 2 does not influence the existence of only one black component in $N^*(p)$. Therefore, we do not need to consider these points in the input to function Check-Table 1.1.

By Lemma 4.4, making black all corner points which are not 26-adjacent to q (marked ◉ in Fig. 2) and are don't-care does not influence whether there is only one black component in $N^*(p)$ or not. Therefore, we do not need to consider these points don't-care in the input to function Check-Table 1.1.

By Lemma 4.5, making white all points marked ◎ in Fig. 2 which are don't-care does not influence the existence of only one black component in $N^*(p)$. Therefore, we do not need to consider these points to be don't-care in the input to function Check-Table 1.1.

Hence, Check-Table 1.1 ($N'_q(p)$) returns true if and only if there is only one black component in $N^*(p)$ for every assignment to don't-care points.

Note that Test 1.2 applies only if all 6-neighbors of p are either white or don't-care. Therefore, if a point marked ○ in Fig. 3 is don't care, it can be colored white, by Lemma 4.5. Hence we do not need to consider the points marked ○ in the input to function Check-Table 1.2. Taking this under consideration, the correctness of Test 1.2 can be proven in the same way as for Test 1.1.

Note that Test 1.3 applies only if all 18-neighbors of p are either white or don't-care. Since q is a black corner of $N(p)$, if there exists a black point or a don't-care point x in $N(p) - N(q)$, we can assign x to be black and all don't-care points in $N(p) \cap N(q)$ to be white. With this assignment, q and x are two black points which cannot be joined by a 26-black path in $N^*(p)$, and therefore $N(p)$ fails C1. If all points in $N(p) - N(q)$ are white, then p is adjacent to only one black component for any possible assignment of $N(p)$, and therefore $N(p)$ passes C1.

Test 1.4 is trivially correct, since we can color all 26-neighbors of p white. ■

The following fact was observed in [12].

LEMMA 4.3. *Suppose p is a black point in $N(p)$. Then both of the following hold*:

1. *Let q be a black 6-neighbor of p (see Fig. 2). Then the colors of the eight unmarked points have no influence on whether $N(p)$ satisfies C1 or not.*

2. *Let q be a black diagonal neighbor of p (see Fig. 3). Then the colors of the two unmarked points have no influence on whether p satisfies C1 or not.*

Proof. (1) holds since the black point q is adjacent to p and every unmarked point is adjacent to q. Therefore, none of the unmarked points, if they are black, can belong to a different component than the component containing q. (2) holds for similar reasons. ■

LEMMA 4.4. *Suppose p is a black point having a black neighbor q. Then changing the color of all don't-care corner points of $N(p)$ to black has no influence on whether p satisfies C1 or not.*

Proof. If any corner of $N(p)$ is adjacent to some black points in $N^*(p)$, then by Lemma 4.3 its color does not influence the simplicity of p. Suppose there is a don't-care corner x such that every point in $N^*(p) \cap N^*(x)$ is either white or don't-care. We can let all points in $N^*(p) \cap N(x)$ be white. If we also let x be black, then we obtain an assignment making p adjacent to two black components ($\{x\}$ and the component containing q). Thus q is nonsimple. ■

LEMMA 4.5. *Suppose p is a black point having a black neighbor q. Then changing the color of all don't-care points in $N(p) \cap N(q)$ to white has no influence on whether p satisfies C1 or not.*

Proof. If $N(p)$ with all don't-care points in $N(p) \cap N(q)$ assigned white fails C1, then there exists an assignment such that $N(p)$ fails C1. If $N(p)$ with all don't-care points in $N(p) \cap N(q)$ assigned white passes C1, then every black or don't-care point which is not 26-adjacent to q is connected to q by a black 26-path in $N^*(p)$. Since every point in $N(p) \cap N(q)$ is 26-adjacent to q, there can be only one 26-block component in $N^*(p)$ for every assignment to don't-care points. ■

PROPOSITION 4.6. *The test C2 is correct, i.e., function $C2(N_{18}(p))$ returns true if and only if p satisfies condition C2 in $N_{18}(p)$.*

Proof. First we show that Test 2.1 is correct. By Lemma 4.7, coloring all don't-care diagonal neighbors of p black does not influence the existence of only one white 6-component in $N_{18}(p)$ which is 6-adjacent to p. Therefore, we do not need to consider these points to be don't-care in the input to function Check-Table 2. Hence, Check-Table 2 ($N'_{18}(p)$) returns true if and only if there is only one white 6-component 6-adjacent to p in $N_{18}(p)$ for every assignment to don't-care points.

Test 2.2 is trivially true, since we can color all 6-neighbors of p black. ■

LEMMA 4.7 *Suppose p is a black point having a white 6-neighbor q. Then changing the color of all don't-care diagonal neighbors of p to black has no influence on whether p satisfies C2 or not.*

Proof. If $N_{18}(p)$ with all don't-care diagonal neighbors of p assigned black fails C2, then there exists such an assignment that $N(p)$ fails C2. If $N_{18}(p)$ with all don't-care diagonal neighbors of p assigned black passes C2, then every white or don't-care 6-neighbor of p is connected to q by a white 6-path in $N_{18}(p)$. Since any assignment to the don't-care diagonal neighbors of p cannot influence the existence of only one white 6-component 6-adjacent to p, there is only one such component in $N_{18}(p)$. ■

5. CONCLUSION

Hall [2] gives a 2D computerized test such that if a 2D thinning algorithm passed this test, it is guaranteed to preserve connectivity. Ma [6] establishes sufficient conditions for 3D parallel thinning algorithms to preserve connectivity, which allow us to give a similar computerized test for the 3D case. A necessary part of the 3D computerized test is an algorithm for determining whether a 3D three-color deletion configuration is simple or not. In this paper we presented such an algorithm, which can be implemented on nearly every computer, since it requires only 1.4 Mbytes of memory for the lookup tables.

REFERENCES

1. W. X. Gong and G. Bertrand, A simple parallel 3D thinning algorithm, in *IEEE International Conference on Pattern Recognition, 1990*.
2. R. W. Hall, Connectivity preserving parallel operators in 2D and 3D images, in *Proceedings, 1992 SPIE Conference on Vision Geometry, Boston, MA, 1992*, pp. 172–183.
3. R. W. Hall, Tests for connectivity preservation for parallel reduction operators, *Topology Applic.* **46,** 1992, 199–217.
4. T. Y. Kong, On the problem of determining whether a parallel reduction operator for *n*-dimensional binary images always preserves topology, in *Proceedings, 1993 SPIE Conference on Vision Geometry, Boston, MA, 1993*, pp. 69–77.
5. T. Y. Kong and A. Rosenfeld, Digital topology: Introduction and survey, *CVGIP* **48,** 1989, 357–393.
6. C. M. Ma, On topology preservation in 3D thinning, *CVGIP: Image Understanding* **59-3,** 1994, 328–339.
7. G. Malandain and G. Bertrand, Fast characterization of 3D simple points, in *IEEE International Conference on Pattern Recognition, 1992*, pp. 232–235.
8. D. G. Morgenthaler. Three-dimensional simple points: Serial erosion, parallel thinning, and skeletonization, Technical Report TR-1005, Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, MD, 1981.
9. J. Mukherjee, P. P. Das, and B. N. Chatterji, On connectivity issues of ESPTA, *Pattern Recognit. Lett.* **11,** 1990, 643–648.
10. C. Ronse, Minimal test patterns for connectivity preservation in parallel thinning algorithm for binary images, *Discrete Appl. Math.* **21,** 1988, 67–79.
11. A. Rosenfeld, A characterization of parallel thinning algorithms, *Information Control* **29,** 1975, 286–291.
12. P. K. Saha, B. B. Chaudhuri, and D. Dutta Majumder, Principles and algorithms for 2D and 3D shrinking, Technical Report TR/KBCS/2/91, N.C.K.B.C.S. Library, Indian Statistical Institute, Calcutta, India, 1991.
13. Y. F. Tsao and K. S. Fu, A parallel thinning algorithm for 3D pictures, *Comput. Graphics Image Process.* **17,** 1981, 315–331.
14. Y. F. Tsao and K. S. Fu, A 3D parallel skeletonwise thinning algorithm, in *IEEE Pattern Recognition and Image Processing Conference, 1982*, pp. 678–683.