Moment-Preserving Sharpening — A New Approach to Digital Picture Deblurring

LING-HWEI CHEN

Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China

AND

WEN-HSIANG TSAI

Department of Information Science, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China

Received September 18, 1985; revised February 25, 1987

A new approach to sharpening blurred pictures using the moment-preserving principle is proposed. The new gray value assigned to each pixel in the desired picture is computed deterministically in such a way that the moments of the $n \times n$ neighborhood of the pixel are preserved. The selection of the neighborhood size n depends on the blurring degree of the picture. The sharpening operation can also be applied iteratively to an input picture to get better sharpening effect. Experimental results show that the approach is indeed effective for picture deblurring. © 1988 Academic Press, Inc.

I. INTRODUCTION

Whenever a picture is converted from one form to another, it is degraded in most cases. Deblurring is often necessary before using the picture because degradation usually blurs the original picture which is assumed to have been sharp. One approach to picture deblurring is to determine the blurring function introduced on the blurred picture, and then to restore the picture using a suitable restoration algorithm [1]. Another approach to picture sharpening makes no attempt to estimate the degradation function but directly applies sharpening operations to the picture, possibly based on certain subjective interpretation of the degradation process [1–3].

Many operators for picture sharpening have been discussed in the literature [1]. The Laplacian operator and the high-emphasis spatial frequency filtering are two common ones for sharpening blurred and noiseless pictures [1]. Rowe [2] proposed some nonlinear operators for picture sharpening which make picture details sharper than the Laplacian can. Lee [3] developed another method to sharpen two-dimensional image arrays, based on image local statistics.

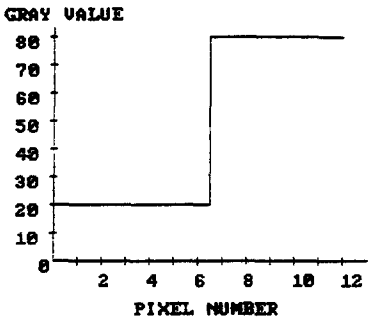
In this paper, we propose a new sharpening method based on the moment-preserving principle which has also been applied to subpixel edge detection [4] and moment-preserving thresholding [5]. No attempt is made to estimate the picture degradation process. The proposed moment-preserving sharpening is directly applied to an observed picture. And its effectiveness is verified by experimental results.

The method is applied to the picture pixel by pixel. The new gray value assigned to each pixel in the desired picture is obtained by preserving the first three moments of the $n \times n$ neighborhood of the pixel, under the assumption that there exist only two or several gray values in the neighborhood. It is found that the sharpening effect of the method is closely related to the neighborhood size n, as will be illustrated later in the paper. The method can also be applied iteratively to obtain better sharpening results.

In the remainder of this paper, we first describe the proposed moment-preserving method for picture sharpening. Some examples are also given to illustrate the sharpening effect. Iteration of the method is discussed next, with some further illustrative examples included. Finally, experimental results are shown to support the feasibility of the proposed method.

20 20 20 20 20 20 80 80 80 80 80 80

(a) Gray values of the image.

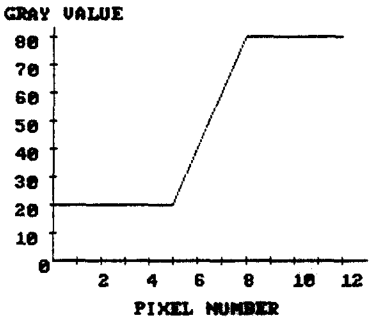


(b) A waveform for the image.

FIG. 1. An unblurred image.

20 20 20 20 20 60 80 80 80 80 80

(a) Gray values of the blurred image.

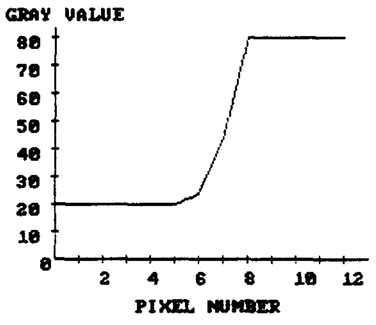


(b) A waveform for the blurred image.

FIG. 2. A blurred image of Fig. 1 with a 2-pixel blurring width.

20 20 20 20 20 24 44 80 80 80 80 80

(a) Gray values of the sharpened image.

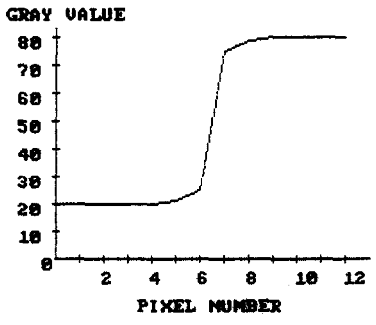


(b) A waveform for the sharpened image.

FIG. 3. Sharpening effect on the blurred image of Fig. 2 for $n = 3$.

20 20 20 20 21 25 75 79 80 80 80 80

(a) Gray values of the sharpened image.



(b) A waveform for the sharpened image.

FIG. 4. Sharpening effect on the blurred image of Fig. 2 for $n = 5$.

II. MOMENT-PRESERVING SHARPENING

A. *The Method*

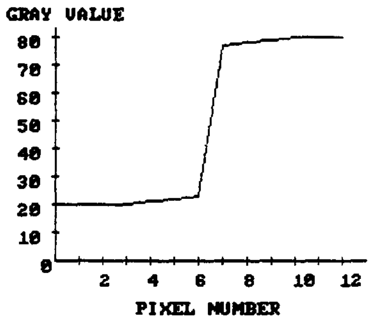
Given a blurred image G, let (x, y) be a pixel with gray value $g(x, y)$ in G. Let N be the $n \times n$ neighborhood of pixel (x, y). The ith moment m_i of N in G is defined as

$$m_i = (1/n^2) \sum_{j=1}^{n^2} g^i(x_j, y_j), \qquad i = 1, 2, 3, \ldots,$$

where $(x_j, y_j) \in N$.

20 20 20 21 22 23 77 78 79 80 80 80

(a) Gray values of the sharpened image.

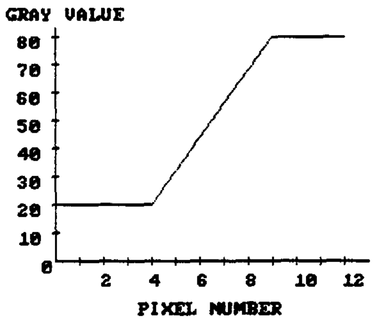


(b) A waveform for the sharpened image.

FIG. 5. Sharpening effect on the blurred image of Fig. 2 for $n = 7$.

20 20 20 20 32 44 56 68 80 80 80 80

(a) Gray values of the blurred image.

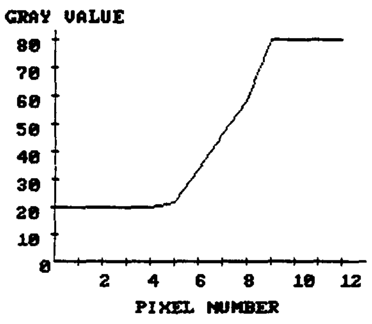


(b) A waveform for the blurred image.

FIG. 6. A blurred image of Fig. 1 with a 4-pixel blurring width.

20 20 20 20 22 34 46 58 80 80 80 80

(a) Gray values of the sharpened image.

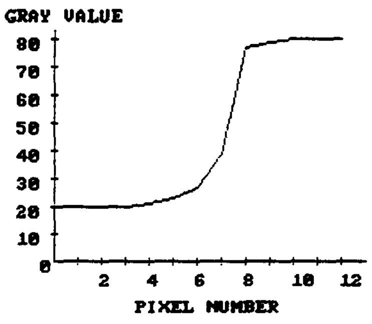


(b) A waveform for the sharpened image.

FIG. 7. Sharpening effect on the blurred image of Fig. 6 for $n = 3$.

20 20 20 21 23 27 39 77 79 80 80 80

(a) Gray values of the sharpened image.



(b) A waveform for the sharpened image.

FIG. 8. Sharpening effect on the blurred image of Fig. 6 for $n = 5$.

Suppose that N in the original unblurred picture F has only two gray values. And we regard G as a blurred version of F. The proposed method defines an operator that, when applied to N in G, generates a bilevel block N', such that the first three moments of N in G is preserved in N'. Let h_1, h_2 be the two gray values in N', P_1 be the fraction of pixels with gray value h_1 in N', and P_2 be the fraction of the other pixels with gray value h_2 in N'. Then, we can obtain three equalities as follows:

$$P_1 h_1 + P_2 h_2 = m_1,$$
$$P_1 h_1^2 + P_2 h_2^2 = m_2,$$
$$P_1 h_1^3 + P_2 h_2^3 = m_3,$$

where the terms on the left-hand sides are the moments of N'. Note that

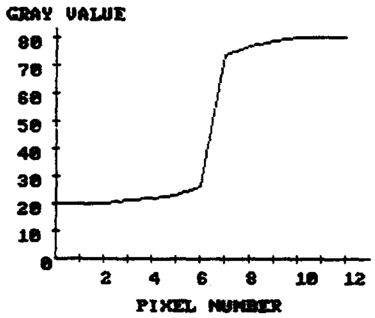
$$P_1 + P_2 = 1.$$

Now, we have four unknown parameters P_1, P_2, h_1, and h_2, and four equations. So, we can solve the equations to obtain the values for the unknowns as follows [5]:

$$h_1 = (1/2)\left[-c_1 - \left(c_1^2 - 4c_0\right)^{1/2}\right],$$
$$h_2 = (1/2)\left[-c_1 + \left(c_1^2 - 4c_0\right)^{1/2}\right],$$
$$P_1 = (1/P_d)\begin{vmatrix} 1 & 1 \\ m_1 & h_2 \end{vmatrix},$$
$$P_2 = 1 - P_1,$$

20 20 21 22 23 26 74 77 79 80 80 80

(a) Gray values of the sharpened image.

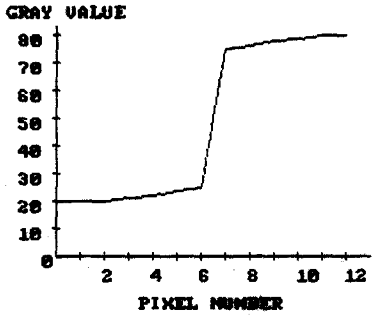


A waveform for the sharpened image.

FIG. 9. Sharpening effect on the blurred image of Fig. 6 for $n = 7$.

20 20 21 22 24 25 75 76 78 79 80 80

(a) Gray values of the sharpened image.



(b) A waveform for the sharpened image.

FIG. 10. Sharpening effect on the blurred image of Fig. 6 for $n = 9$.

where

$$P_d = \begin{vmatrix} 1 & 1 \\ h_1 & h_2 \end{vmatrix},$$

$$c_0 = (1/c_d)\begin{vmatrix} -m_2 & m_1 \\ -m_3 & m_2 \end{vmatrix},$$

$$c_1 = (1/c_d)\begin{vmatrix} 1 & -m_2 \\ m_1 & -m_3 \end{vmatrix},$$

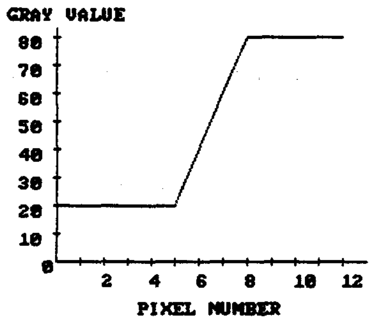
$$c_d = m_2 - m_1^2.$$

20 20 20 20 20 40 60 80 80 80 80 80

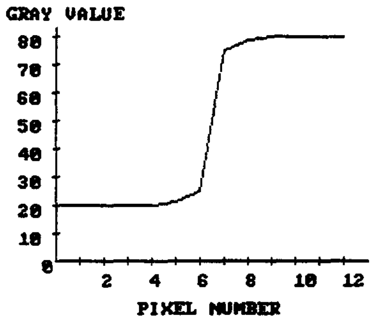
(a) The original blurred picture.

20 20 20 20 21 25 75 79 80 80 80 80

(b) The result of iteration 1.



(a′) A waveform for (a).



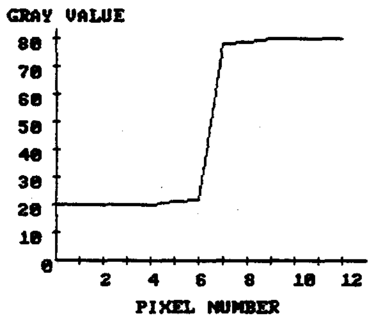
(b′) A waveform for (b).

20 20 20 20 21 22 78 79 80 80 80 80

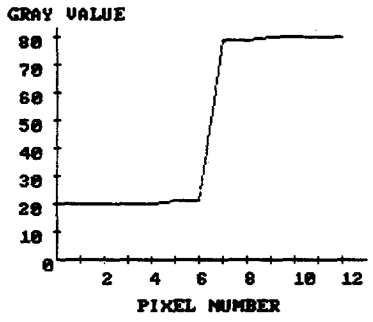
(c) The result of iteration 2.

20 20 20 20 21 21 79 79 80 80 80 80

(d) The result of iteration 3.



(c′) A waveform for (c).



(d′) A waveform for (d).

FIG. 11. The sharpening results of using the iterative process with neighborhood size fixed as 5.

Then, we choose h as the P_1-tile of the histogram of N in G. In practice, there may exist no discrete gray value which is exactly the P_1-tile of the histogram. Then, h should be chosen as the gray value closest to the P_1-tile. What remains now is to use the above result to assign a new gray value to pixel (x, y), such that the neighborhood N in the blurred picture G can be sharpened. This can be done by assigning a new gray value $g'(x, y)$ to (x, y) according to the following rule:

$$\begin{aligned} g'(x, y) &= h_2 \quad \text{if } g(x, y) > h, \\ &= h_1 \quad \text{if } g(x, y) \leq h. \end{aligned}$$

The above process is applied to each pixel (x, y) in G.

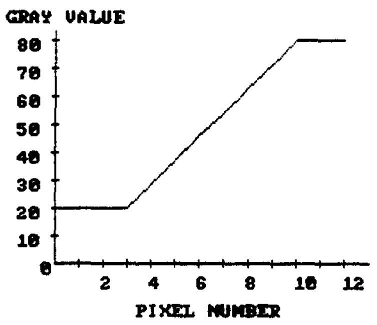
In the above discussion, nothing is said about how to select the neighborhood size n. In fact, n determines the sharpening effect. If the blurring on the boundary of a

20 20 20 29 37 46 54 63 71 80 80 80

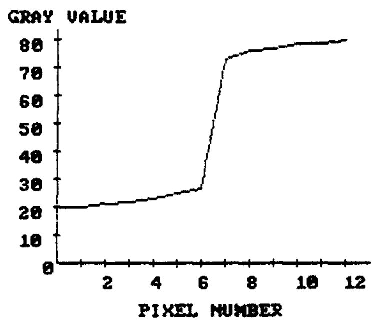
(a) The original blurred picture.

20 21 22 23 25 27 73 76 77 79 79 80

(b) The result of iteration 1.



(a′) A waveform for (a).



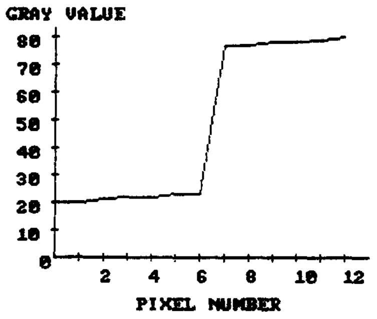
(b′) A waveform for (b).

20 21 22 22 23 23 77 77 78 78 79 80

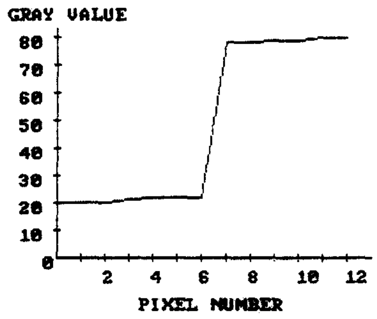
(c) The result of iteration 2.

20 20 21 22 22 22 78 78 79 79 80 80

(d) The result of iteration 3.



(c′) A waveform for (c).



(d′) A waveform for (d).

FIG. 12. The sharpening results of using the iterative process with neighborhood size fixed to be 9.

black shape in a white background, for example, is "narrow" (i.e., it occurs in a band with only a few pixels wide along the boundary), then n need not be too large. In other words, the selection of n depends on the blurring effect on the picture details. Later in this section, we will give some examples to show the relationship between the value n and the deblurring effect of the proposed method.

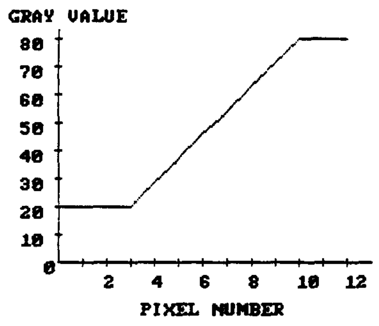
Additionally, we assume in the above discussion that each neighborhood N includes only two gray values in the unblurred picture F. This need not be true if the picture has finer details. However, it is easy to extend the above consideration to the case that N has k gray values with $k > 2$. The approach is similar except that we now have to preserve the first $2k - 1$ moments. A detailed description of the general solution to the desired k gray values can be found in [5]. The value k may be chosen experimentally, but according to our experimental experience, $k = 2$ is

20 20 20 29 37 46 54 63 71 80 80 80

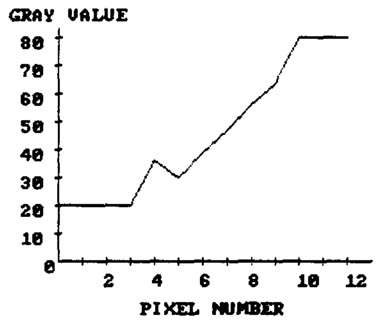
(a) The original blurred picture.

20 20 20 36 30 39 47 56 64 80 80 80

(b) The result of iteration 1.



(a′) A waveform for (a).



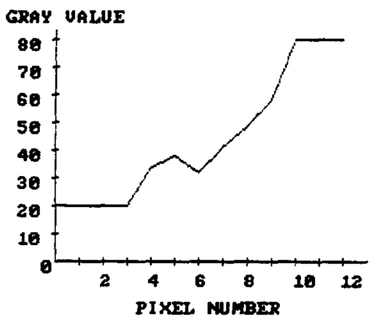
(b′) A waveform for (b).

20 20 20 34 38 32 41 49 58 80 80 80

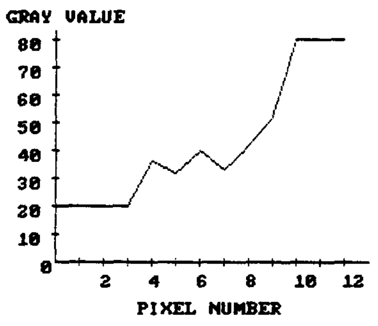
(c) The result of iteration 2.

20 20 20 36 32 40 33 42 52 80 80 80

(d) The result of iteration 3.



(c′) A waveform for (c).



(d′) A waveform for (d).

FIG. 13. The sharpening results of using the iterative process with neighborhood size fixed to be 3.

appropriate for most cases. If a more systematic method is desired, then the following approach can be used. Define

$$E_m = \sum_{xy}\sum |g(x, y) - g'_m(x, y)|,$$

where $(x, y) \in N$ and $g'_m(x, y)$ is the resulting gray value with $k = m$. Enumerate m from 2 to a preselected level number M, each time applying the proposed approach to N and calculating E_m until an m_0 is found such that E_{m_0} satisfies the condition $E_{m_0} < \varepsilon$, where ε is a preselected threshold value. If no such m_0 exists, then we choose an m_0 such that $E_{m_0} = \min_m(E_m)$. Finally, k is set to be m_0.

B. *Illustrative Examples*

The effectiveness of moment-preserving sharpening is demonstrated with several examples here. Figure 1 shows the original pixel gray values of a given unblurred one-dimensional image, which includes two areas with constant gray values 20 and 80. Figure 2 shows the gray values of a blurred version of Fig. 1, which includes two

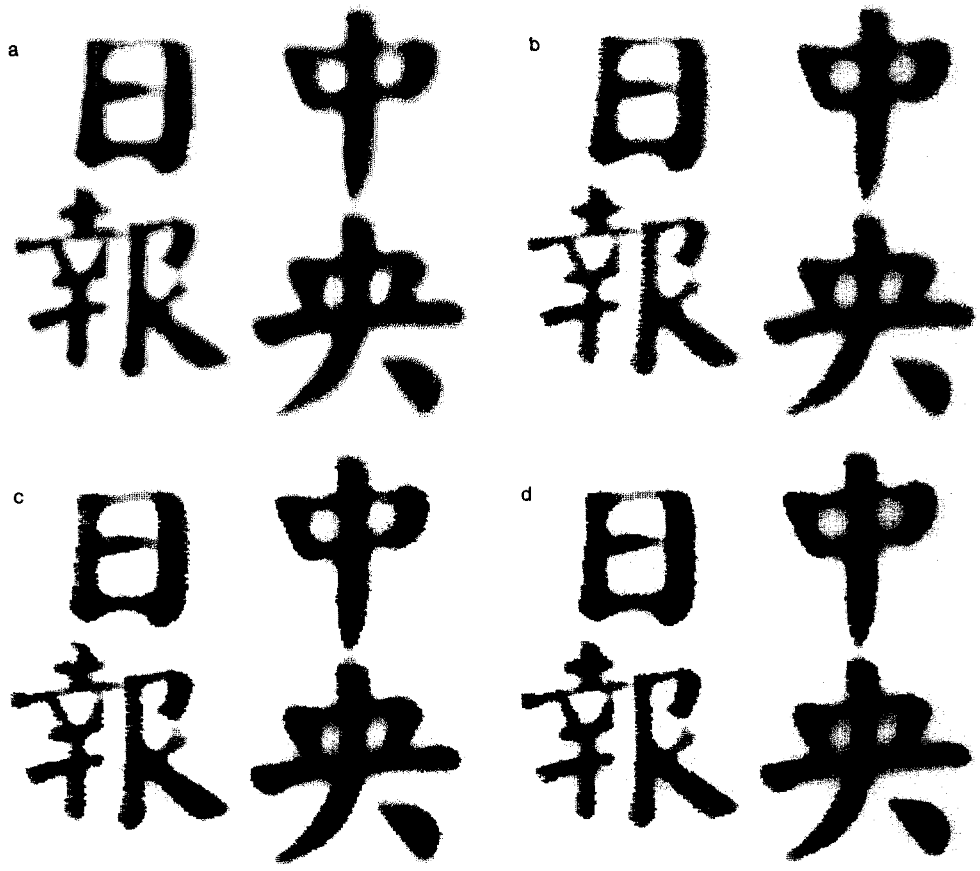


FIG. 14. Noniterative sharpening results of a character image: (a) the blurred image; (b) sharpening result for $n = 3$; (c) sharpening result for $n = 5$; (d) sharpening result for $n = 7$.

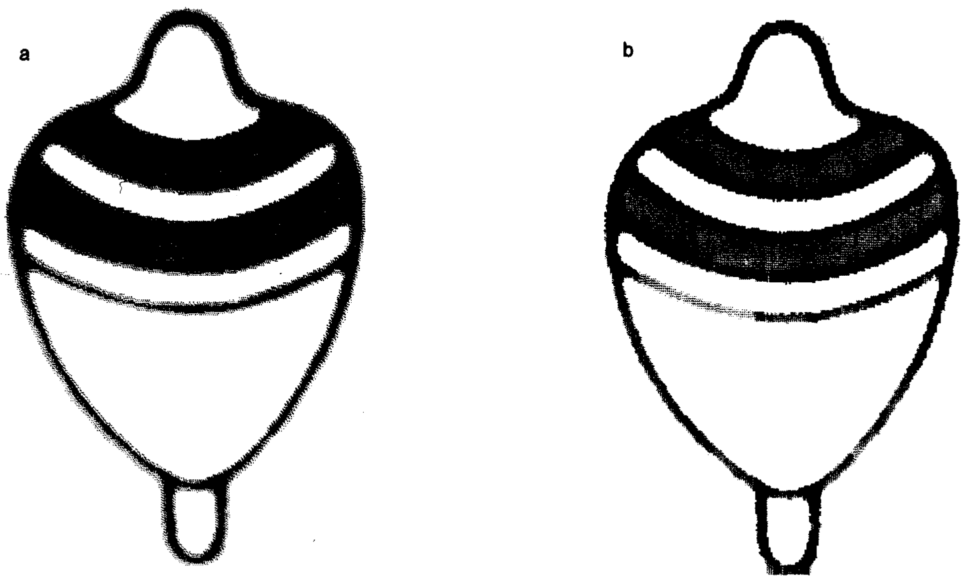


FIG. 15. Noniterative sharpening results of a toy image: (a) the blurred image; (b) sharpening result for $n = 7$.

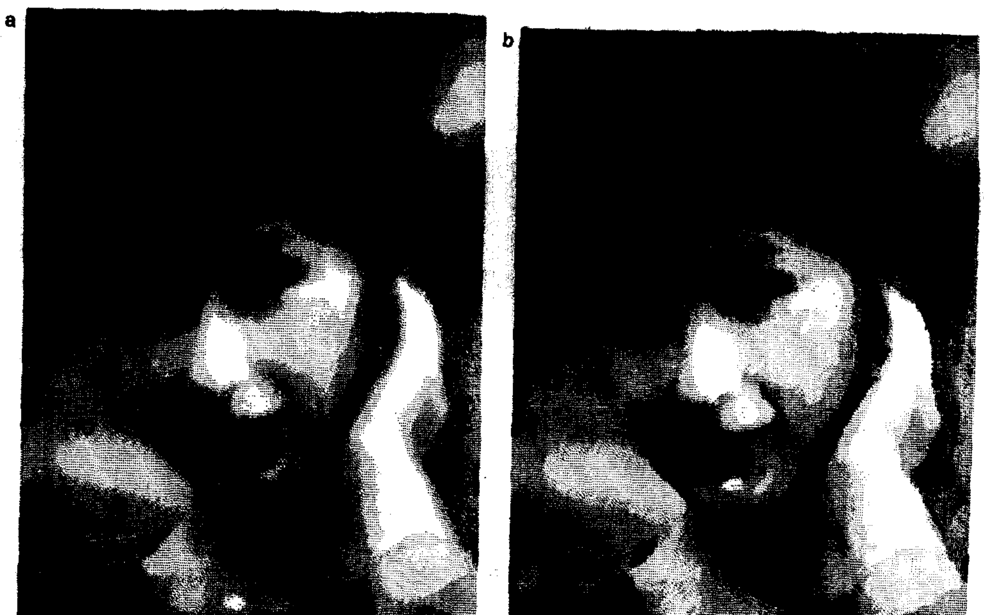


FIG. 16. Noniterative sharpening results of a girl image: (a) the blurred image; (b) sharpening result for $n = 7$.

areas with constant gray values (the leftmost 5 pixels and the rightmost 5 pixels) and a blurred boundary (the central 2 pixels). Figures 3, 4, and 5 show the sharpening results of Fig. 2 for $n = 3, 5, 7$. Figure 6 shows a blurred image of Fig. 1, which includes a blurred boundary 4 pixels wide. Figs. 7, 8, 9, and 10 show the sharpening results for $n = 3, 5, 7, 9$. Figures 1(b)–10(b) are the waveform diagrams of Figs. 1(a)–10(a). From these examples, we see that for the sharpening effect to be satisfactory, the size n should be selected to be about 3 pixels larger than the width of the blurred boundary.

III. ITERATIVE MOMENT-PRESERVING SHARPENING

In the above, moment-preserving sharpening is applied to each pixel of the blurred image for a single time. It is found in this study that if the sharpening operation is applied to the picture repetitively, the sharpening effect can be improved. This is reasonable, because each sharpened picture can be regarded as a less but still blurred version of the original unblurred picture. At each iteration step, we use the prior result as the input picture. As the number of iterations increases, the blurring effect is removed gradually. Some illustrative examples are shown next to illustrate the feasibility of this idea.

Figures 11(a), 12(a), and 13(a) are the blurred versions of Fig. 1 with 2, 6, and 6 blurred pixels, respectively. Figures 11(b), 12(b), and 13(b) are the results of iteration 1 of the sharpening process applied to Figs. 11(a), 12(a), and 13(a) with neighborhood sizes chosen to be 5, 9, and 3, respectively. Figures 11(c), 12(c), and 13(c) are the results of iteration 2 of the sharpening process applied to Figs. 11(b), 12(b), and 13(b). Figures 11(d), 12(d), and 13(d) are the results of iteration 3 of the sharpening process applied to Figs. 11(c), 12(c), and 13(c). Figures 11(a′)–13(d′) are the waveform diagrams of Figs. 11(a)–13(d).

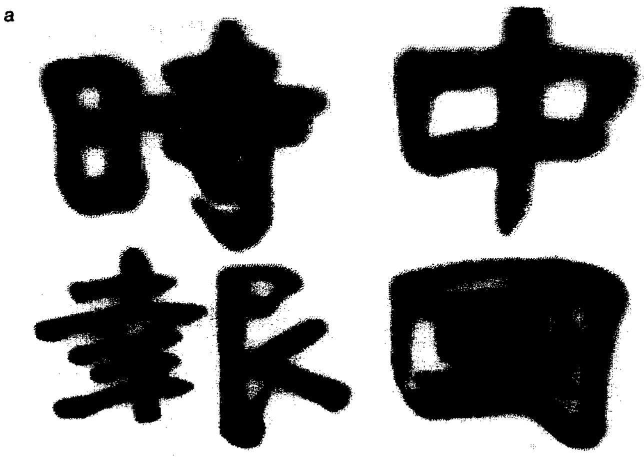
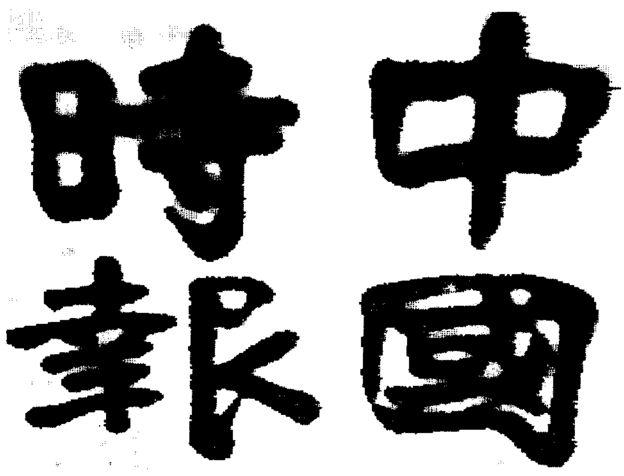
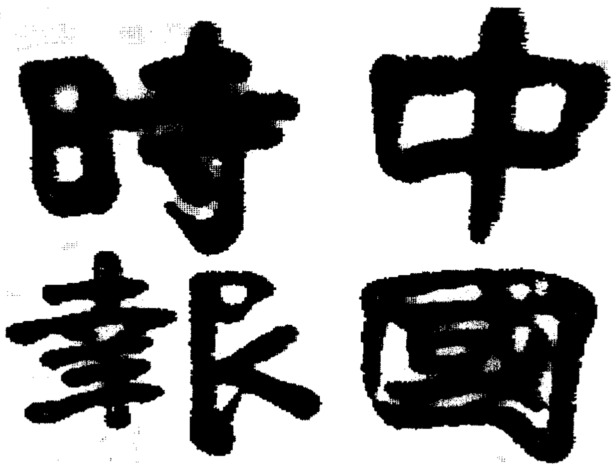


FIG. 17. Iterative sharpening results of a character image with fixed size $n = 7$: (a) the blurred image; (b) result of iteration 1; (c) result of iteration 2; (d) result of iteration 3.

b



c



d

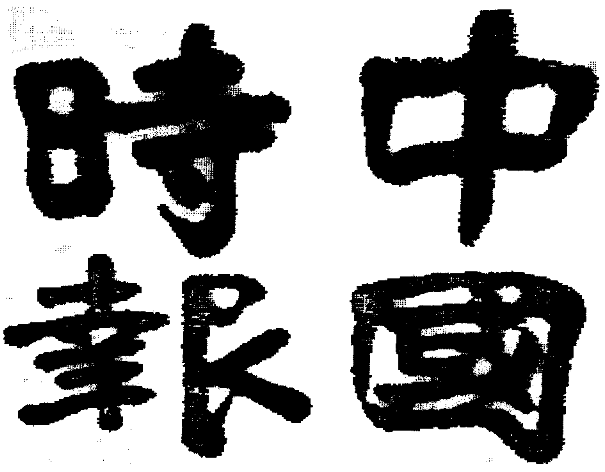


FIG. 17—Continued

From Figs. 11–13, we can see the fact that the iterative process indeed can improve the sharpening effect. On the other hand, from Fig. 13, we find that if n is too small, the iterative process cannot give stable and smooth results. Finally, it is seen that only a few iterations of moment-preserving sharpening are necessary before the result becomes satisfactory.

V. EXPERIMENTAL RESULTS

The proposed approach has been tried on a lot of images, with some results shown in Figs. 14–18. Each figure shown includes a tested image in (a), the sharpening results in (b), (c), and (d) with $k = 2$. Figures 14–16 are the results of noniterative sharpening. Figures 17–18 are the results of iterative sharpening. Figure 14(a) includes some blurred characters, and Figs. 14(b), (c), and (d) are the sharpening results with $n = 3$, 5, and 7. Figure 15(a) includes a blurred toy, and Fig. 15(b) is the sharpening result with $n = 7$. Figure 16(a) includes a blurred girl, and Fig. 16(b) is the sharpening result with $n = 7$. It is seen that better sharpening

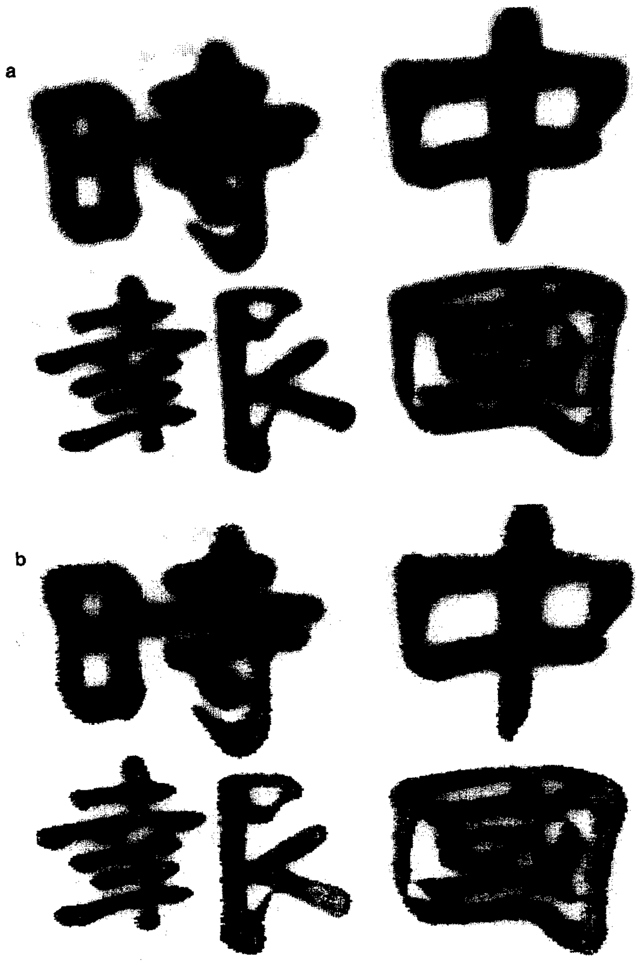


FIG. 18. Ineffective iterative sharpening results of the same character image as in Fig. 17 with $n = 5$ (too small): (a) the blurred image; (b) result of iteration 1; (c) result of iteration 2; (d) result of iteration 3.

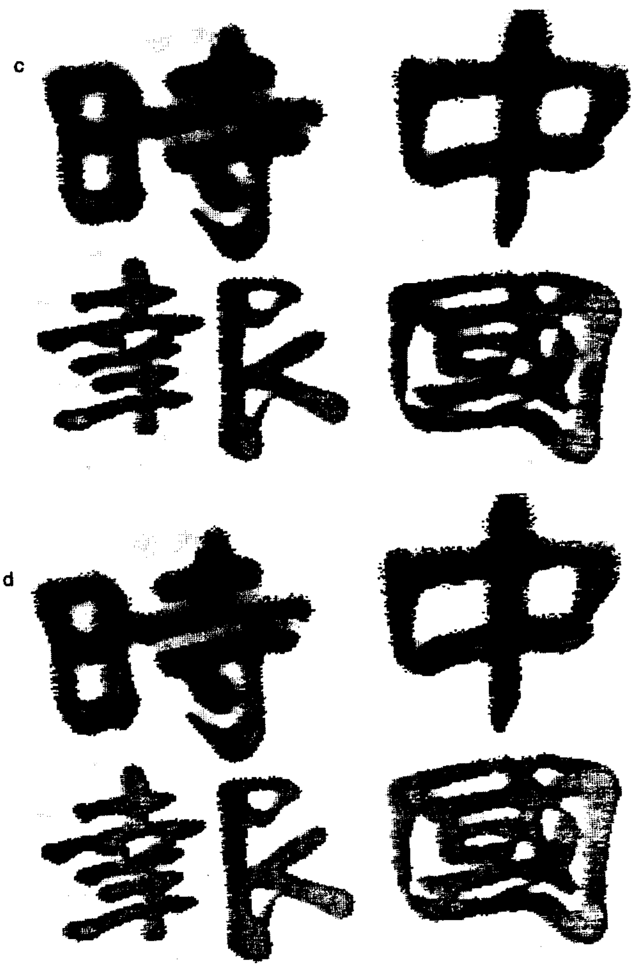


FIG. 18—Continued

results are found in the first two cases (with $n = 7$). This is expected because in the two cases, the shapes (character strokes and toy line drawings) presumably include high-contrast boundaries in their unblurred versions. Figures 17(b)–(d) are the results of iterations 1, 2, and 3 of the sharpening process applied to Fig. 17(a) with fixed $n = 7$. Figures 18(b)–(d) are the results of iterations 1, 2, and 3 of the sharpening process with $n = 5$ applied to Fig. 18(a). Figure 17 shows a good result, but Fig. 18 shows an unsmooth result. The reason is that n is chosen too small.

REFERENCES

1. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Vol I, Academic Press, New York, 1982.
2. P. P. Rowe, Some nonlinear operators for picture processing, *Pattern Recognition*, **11**, 1979, 341–342.
3. J. S. Lee, Digital image enhancement and noise filtering by use of local statistics, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-2**, 1980, 165–168.
4. A. J. Tabatabai and O. R. Mitchell, Edge location to subpixel values in digital imagery, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**, 1984, 188–201.
5. W. H. Tsai, Moment-preserving thresholdings: A new approach, *Comput. Vision Graphics Image Process.* **29**, 1985, 377–393.

COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 41, 14–27 (1988)

A Time- and Space-Optimal Algorithm for Boolean Mask Operations for Orthogonal Polygons*

PETER WIDMAYER

Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, Postfach 6980, 7500 Karlsruhe, West Germany

AND

DERICK WOOD

Data Structuring Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Received March 10, 1986; revised March 24, 1987.

Boolean combinations of VLSI masks are important in VLSI design. We describe an optimal algorithm for two layers and rectangles having the same orientation in the plane. The algorithm runs in time $O(n \log n + p)$ using space $O(n)$, where n is the number of rectangles and p is the number of edges in the output. Note that this algorithm avoids spending computation time on edges that do not contribute to the output, unlike its counterpart for arbitrary polygons. Moreover, the contour of the Boolean combination is computed in the form of contour cycles rather than single contour edges. Also, we extend the algorithm to more than two layers and to orthogonal polygons having the same orientation. The time bound remains the same for any fixed number of layers; it is $O(k2^k \cdot n \log n + p)$, for $k \geq 2$ layers when k is an input parameter. © 1988 Academic Press, Inc.

1. INTRODUCTION

Boolean combinations of VLSI layout masks for several layers are needed for a number of purposes in the VLSI physical design process, such as design rule checking, connectivity checking, device recognition, or feature extraction. For instance, a transistor may be realized on a VLSI chip whenever polysilicon intersects diffusion, as given by the two masks. Because of the high number of objects involved, the efficiency of algorithms for Boolean mask combinations turns out to be very important. Therefore, a number of practitioners and researchers have devoted their attention to the development and implementation of fast algorithms for the Boolean masking problem, see [1, 2, 3, 5, 6, 7, 9, 11, 12, 17].

While the solutions described for the general case of arbitrary polygons seem to be reasonably efficient, the same cannot be said about the orthogonal or isothetic case (the term "isothetic" is recommended by [13] to denote objects with sides parallel to any of the coordinate axes). Specifically, assume that the mask of each VLSI layer is given as a (multi-) set of isothetic rectangles. For most VLSI physical layouts, this is still the situation today. In such a special case, it is certainly highly undesirable to spend as much computation time for Boolean mask combinations as is acceptable in the general case. The best known solution for the general case takes time $O((n + k)\log n)$ for two masks and, for instance, the Boolean AND operation,

*The first author was supported under a grant from the Deutsche Forschungsgemeinschaft Wi 810/1-1 and the second author under Natural Sciences and Engineering Research Council of Canada Grant A-5692.

where n is the total number of polygon edges involved, and k is the total number of edge intersections. Note that k may be proportional to n^2, even though a description of the algorithm's output may be very small, for instance, proportional to n or constant. In other words, the algorithm for the general case spends computation time on intersection points that do not contribute to the final result. Moreover, it would certainly be preferable to spend only constant time per output item rather than $\log n$, as in the general case.

We describe an algorithm without these two flaws in the next section, that is, an algorithm with $O(n \log n + p)$ running time for two masks with a total of n isothetic rectangles, the Boolean operation AND, and p edges in the contour of the resulting polygons. Our algorithm reports the resulting contour in the form of contour cycles and not just contour edges, but uses no more than $O(n)$ space. The algorithm is optimal in both time and space. In addition, we describe how to compute other Boolean combinations of two masks. Finally, in Section 3 we generalize our algorithm to more than two masks and to isothetic polygons.

2. BOOLEAN COMBINATIONS OF TWO LAYERS

We are given two layers (multisets) of isothetic or orthogonal rectangles in the plane, say a layer $\mathscr{R}$ of red ones and layer $\mathscr{G}$ of green ones. Let $\mathscr{R} = \{\mathscr{R}_i | 1 \le i \le n_r\}$ and $\mathscr{G} = \{\mathscr{G}_j | 1 \le j \le n_g\}$, where $\mathscr{R}_i$ ($\mathscr{G}_j$) denotes a red (green) rectangle. Each rectangle is given by the four coordinates of its four sides, that is, by a quadruple $(x_{\text{left}}, x_{\text{right}}, y_{\text{bottom}}, y_{\text{top}})$. A rectangle represents a set of points in the plane in the usual way: rectangle $(x_{\text{left}}, x_{\text{right}}, y_{\text{bottom}}, y_{\text{top}})$ represents the set $\{(x, y) | x_{\text{left}} \le x \le x_{\text{right}}, y_{\text{bottom}} \le y \le y_{\text{top}}\}$. For given $\mathscr{R}$ and $\mathscr{G}$ a point $p = (x, y)$ in the plane is called *red* (*green*) if there exists a rectangle $\mathscr{R}_i \in \mathscr{R}$ ($\mathscr{G}_j \in \mathscr{G}$) with $p \in \mathscr{R}_i$ ($p \in \mathscr{G}_j$).

The *contour* of $\mathscr{R}$ is a description of $\bigcup_{i=1}^{n_r} \mathscr{R}_i$, the set of all red points, in the form of an isothetic or orthogonal polygon (possibly with holes) for each connected set of red points. A polygon is described by an alternating sequence of vertical and horizontal polygon edges in cyclic order around the polygon. The contour problem for a set of isothetic rectangles has been studied by a number of authors [8, 4, 16]. A recent solution [16] uses a simple data structure, the *visibility tree*, to solve the problem in optimal time. In [16] the contour is reported as a set of edges of the contour polygon, not in the form of contour cycles. However, we show, in [15], how the solution of [16] can be modified so that it reports contour cycles in optimal time and space. We base our solution of the Boolean masking problem on a modified version of the visibility tree, and the observation of optimality in [15].

2.1. The Boolean AND

We illustrate the basic idea of our algorithm by computing the Boolean AND of the two layers $\mathscr{R}$ and $\mathscr{G}$. In other words, we show how to compute the contour of the set of points that are both red and green, that is $\bigcup_{i=1}^{n_r} \mathscr{R}_i \cap \bigcup_{j=1}^{n_g} \mathscr{G}_j$. All operators we use are *regularized* in the sense of [14]. This means the results of an operation is taken to be the closure of the interior of the nonregularized ("usual") operation. The effect of regularizing operations is to eliminate isolated or dangling lines or points that may otherwise result from polygons touching each other (for details, consult [14]).

2.1.1. *An Outline of the Algorithm*

We solve the problem by means of a *plane-sweep algorithm*. Imagine a vertical line (the *sweep line*) moving from $x = -\infty$ to $x = +\infty$ through the plane, that is, from left to right. The line meets left vertical edges and right vertical edges of the rectangles in order of ascending x-values. Between any two adjacent x-values, the intersection of the sweep line with the rectangles is the same for any two positions of the sweep line. Therefore it is sufficient to stop the sweep line at the vertical edges of rectangles; we call these x-values the *sweep points* of the sweep line.

This well-known principle, for example, see [10], amounts to solving a two-dimensional problem by solving a series of one-dimensional problems and combining the answers.

During the sweep, we keep track of the horizontal edges (top and bottom edges of rectangles) that currently intersect the sweep line. It is obvious that not all edges form part of the output of the algorithm, but instead only those that are on the boundary of an area consisting of points that are red and green (a red–green area). For simplicity of the discussion only, let us assume that no two horizontal edges (vertical edges) share the same y-value (x-value). However, this assumption is not a crucial one in solving the problem. A straightforward modification of the algorithm given below will suffice when this assumption is dropped.

Upon meeting a vertical edge, we decide which parts of that edge belong to the output of the algorithm. In addition to horizontal edges, we store contour cycles that have started but not yet ended, that is, contour cycles that currently intersect the sweep line. Storing each contour cycle, for instance, as a doubly linked list with two pointers to its ends, we can easily update the contour cycles (start a new one, concatenate two adjacent existing ones, terminate an existing one) in time bounded by a constant for each update operation, that is, in total time $O(p)$. When a contour cycle is terminated, its description is output, and hence it no longer consumes storage space. It has been shown in [15] that the total number of edges of all contour cycles intersected by some vertical line is $O(n)$. Consult [15] for a more detailed explanation of how to maintain contour cycles and a proof of the cited fact. Given the bound on the number of edges in all intersected contour cycles, we can afford to maintain all of the started but not yet terminated contour cycles without asymptotically increasing the space. Therefore, in the following presentation, we shall only describe how to find the horizontal edges of the contour; it is understood that the contour cycles are always computed in the described way.

It is obvious that horizontal contour edges of the red–green area are exactly those parts of edges that belong to the contour of the red area, but fall into a green area, or that belong to the contour of the green area, but fall into a red area; see Fig. 1.

A contour edge of the former type is called a *red–green-visible edge* and a contour edge of the latter type is called a *green–red visible edge*; rg- and gr-edge for short. An edge belonging to the contour of the red (green) area, which is neither an rg- nor a gr-edge, is called a *red-visible* (*green-visible*) *edge*, r- (g-) edge for short. Note that contour edges differ from rectangle edges in that the former may be parts of the latter. In the example depicted in Fig. 1, e_1 is an r-edge, e_2 is an rg-edge, e_3 is a g-edge, e_4, e_5, and e_6 are of none of the mentioned types, and e_7 is a gr-edge.

The four classes of r-, g-, rg-, and gr-edges provide enough information to properly keep track of the rg- and rg-edges during the line sweep, in the following

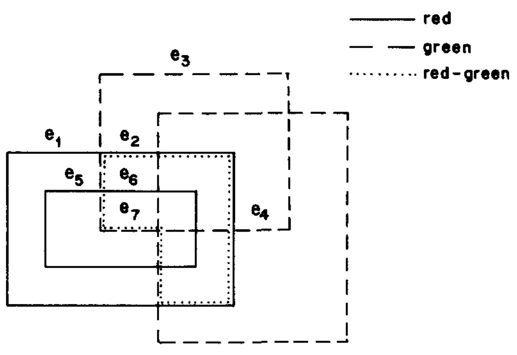


FIG. 1. A red–green contour.

way. Consider what happens to the horizontal edges currently cut by the sweep line, that is, the *active edges*, when a vertical red edge is encountered (similarly for green).

First, consider the case that the edge is the left edge of some rectangle R_i; a left edge for short. At the current x-value, this edge extends from y_{bottom} to y_{top} in the y-direction. All active horizontal edges with y-values not in the interval $[y_{\text{bottom}}, y_{\text{top}}]$ are clearly unaffected. So we restrict our attention to those horizontal edges that have their y-values in this interval. Any edge belonging to the red contour before can no longer belong to the red contour after this left red edge, hence, any r-edges and any rg-edges terminate. Any edge belonging to the green contour will still belong to the green contour after the sweep point, but it will also be in a red area. Hence, the number of gr-edges is enlarged by these g-edges; the g-edges terminate.

Second, consider the case when the sweep line encounters a vertical right red edge. A right red edge corresponds to a red rectangle that is being passed by the sweep line. The y-interval covered by this red rectangle may (partly) still be covered by some other red rectangle. We again restrict our attention to horizontal edges with y-values in the y-interval of the right red edge. Some of the active horizontal edges may still lie in a red area, and others may no longer lie in a red area. Consider red horizontal edges first. Some of them may become visible, but not necessarily all of them. In Fig. 2 at sweep point h of the sweep line, e_1 and e_2 become visible,

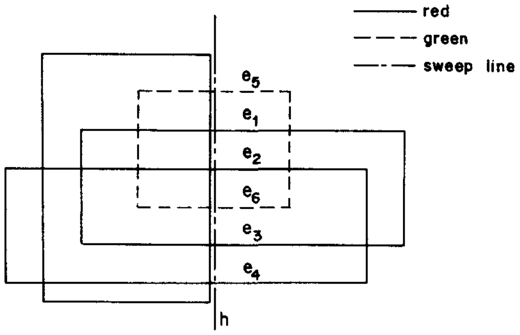


FIG. 2. Horizontal edges becoming visible.

whereas e_2 and e_3 do not. Of those edges becoming visible, some may become red-visible, other may become red–green visible. In our example in Fig. 2, e_1 becomes red–green visible because it lies in a green area, whereas e_4 becomes red-visible. Now consider green horizontal edges. The only possible change is that some of the gr-edges may become g-edges because they exit from a red area, whereas others may stay in a red area and therefore remain gr-edges. In our example, e_5 becomes a g-edge, but e_6 remains a gr-edge at sweep point h.

Besides changing the status of some horizontal edges, a left vertical edge initiates two horizontal edges, namely the top and bottom edges of the corresponding rectangle; that is, these two horizontal edges become active. Their status with respect to the four classes (r-, g-, rg-, gr-edges) has to be determined. For a right vertical edge, the corresponding top and bottom edges of the rectangle become inactive. The edges have to be added to or removed from their classes.

The desired output can now easily be generated by keeping track of the changes in the classes of rg-edges and gr-edges. Whenever an edge joins one of these two classes, that is, becomes green–red-visible or becomes red–green-visible, an edge of the desired red–green contour is started; whenever an edge is removed from these classes, the corresponding edge (which must have started previously) is terminated.

2.1.2. The Algorithm in Detail

The following algorithmic description summarizes the discussion so far.

ALGORITHM *Red-and-Green Contour.* {We are given two sets

$$\mathcal{R} = \{\mathcal{R}_i | 1 \le i \le n_r\}$$

and

$$\mathcal{G} = \{\mathcal{G}_j | 1 \le j \le n_g\}$$

of isothetic rectangles. We output a set of horizontal edges (of maximal length) that lie on the contour of $\bigcup_{i=1}^{n_r} \mathcal{R}_i \cap \bigcup_{j=1}^{n_g} \mathcal{G}_j$. The contour cycles are obtained by the minor modification described above.}

```
begin
  1. Sort the vertical edges of rectangles in $\mathcal{R} \cup \mathcal{G}$ into ascending $x$-order. Each
     $x$-value is called a sweep point. Initialize sets $R$, $G$, $RG$, and $GR$, denoting sets
     of active r-edges, g-edges, rg-edges and gr-edges, to be empty.
  2. For all sweep points in $x$-order corresponding to vertical edges $e$ do: Let $e$ be
     a red vertical edge (without loss of generality; for $e$ green, just exchange red
     with green in the following description). Let $R(e)$, $G(e)$, $RG(e)$, and $GR(e)$
     denote the subsets of those horizontal edges in $R$, $G$, $RG$, and $GR$ that have
     their $y$-values in the range between $e$'s bottommost and topmost points. Let
     $e_{\text{bottom}}$, $e_{\text{top}}$ be the bottom and top horizontal edges of the rectangle of which $e$
     is a vertical edge.
     (a) If $e$ is a left edge, then: output start of edges $G(e)$; output end of edges
         $RG(e)$; $GR := GR \cup G(e)$; $G := G - G(e)$; $R := R - R(e)$; $RG := RG$
         $- RG(e)$; add $e_{\text{bottom}}$, $e_{\text{top}}$ to the corresponding class of active horizontal
         edges, if any; and output start for whichever of the two edges fall into $RG$
         or $GR$.
```

```
    (b) If $e$ is a right edge, then: Let $R'$ and $RG'$ be the sets of edges that were not visible and now become red-visible and red–green-visible, respectively. Let $G'$ be the set of green–red-visible edges in $GR(e)$ that become green-visible. Then: output start of edges $RG'$; output end of edges $G'$; $R := R \cup R'$; $RG := RG \cup RG'$; $GR := GR - G'$; $G := G \cup G'$; remove $e_{\text{bottom}}, e_{\text{top}}$ from the corresponding class of active horizontal edges, if any; and output end for whichever of the two edges are in $RG$ or $GR$.
end
```

Let us now specify in more detail how to perform the (crucial) update operations for sets R, G, RG, and GR of active horizontal edges, that is, how to determine the sets $G(e)$, $R(e)$, $RG(e)$, $GR(e)$ for a left vertical edge, and R', G', RG', GR' for a right vertical edge. We also discuss in the next subsection how to implement the set union and difference operations used in the algorithm.

It is easy to determine $G(e)$, $R(e)$, $RG(e)$, $GR(e)$ for a left vertical edge e by simply testing whether the y-value of a horizontal edge lies within the y-interval from y_{bottom} to y_{top}, the bottom and top ends of e.

It is not as easy to find red (green) edges that become red-visible or red–green-visible (green-visible or green–red-visible). To determine whether an active horizontal red edge is red-visible or red–green-visible it is sufficient to know whether or not the intersection of the edge with the sweep line lies within the intersection of an active red rectangle with the sweep line, that is, whether or not the red point p on the sweep line is covered by a red interval. However, when a right vertical red edge e covering red point p is encountered in the sweep, this binary information is not sufficient to determine whether p will still be covered by some red interval after e has been removed. We, therefore, use a counter to indicate how many red intervals cover a point.

More precisely, we partition the sweep line into a set of contiguous intervals, induced by the set of active red horizontal edges, or red points on the sweep line as we shall say. With each of these intervals, we associate an integer value *red-cover*; this value indicates by how many red rectangles, or intervals on the sweep line, the corresponding interval is covered. A set of *green-cover* values is defined similarly. The cover values are maintained during the sweep as follows.

ALGORITHM *Maintain Cover Values.*

```
begin
  1. Initialize the set of intervals for the red-cover values to be $(-\infty, +\infty)$. Initialize the red-cover and the green-cover values for this interval to be 0.
  2. Upon meeting a red vertical edge in the sweep, do the following (similarly for green):
    (a) If $e$ is a left vertical edge, then: The bottommost and topmost endpoints of $e$ each split a red-cover interval into two intervals. That is, the number of intervals is increased by two. Each of the split intervals inherits the cover from the interval being split. For all intervals between $y_{\text{bottom}}$ and $y_{\text{top}}$, increase the red-cover value by 1.
    (b) If $e$ is a right vertical edge, then: The two endpoints, $y_{\text{bottom}}$ and $y_{\text{top}}$, occur as interval boundaries in the set of red-cover intervals. Decrease the red-cover for all intervals between $y_{\text{bottom}}$ and $y_{\text{top}}$ by 1. Now the two
    intervals sharing endpoint $y_{\text{bottom}}$ have the same red-cover value. Remove
    $y_{\text{bottom}}$ and merge the two intervals into one, inheriting the red-cover from
    the merged intervals. Do the same for $y_{\text{top}}$. This decreases the number of
    red-cover intervals by two.
end
```

The labels of the steps of this algorithm correspond to those of *Red-and-Green Contour*; when interleaving both algorithms according to the labels, each step of *Maintain Cover Values* should be taken first and the corresponding step of *Red-and-Green Contour* second, for each label.

Note that each active red point on the sweep line forms a border between exactly two red-cover intervals with cover values differing by exactly 1. We can now easily express the classification of a red horizontal edge in terms of these covers, for any fixed sweep line position. An active horizontal red edge is an r-edge or an rg-edge if and only if one of its bordering red-cover intervals has red-cover value 0, and the other has red-cover value 1. It is an rg-edge if and only if it lies within a green-cover interval with green-cover value greater than 0. The classification is similar for green edges.

For any fixed sweep line position and an active red (green) horizontal edge e, let gc(e) (rc(e)) denote the green-cover (red-cover) value of the green-cover (red-cover) interval into which e falls. Let $RC(e)$ (GC(e)) be the set of the two red-cover (green-cover) values of the two bordering red-cover (green-cover) intervals. In Fig. 2, at sweep line position h, $r(e_5)$ changes from 1 to 0, $RC(e_1)$ changes from $\{1,2\}$ to $\{0,1\}$, for instance.

At a fixed sweep line position, we can now define classes R, G, RG, and GR in terms of cover values:

(1) $R := \{e \mid e \text{ is red active, } RC(e) = \{0,1\}, \text{gc}(e) = 0\}$;

(2) $RG := \{e \mid e \text{ is red active, } RC(e) = \{0,1\}, \text{gc}(e) > 0\}$;

(3) $G := \{e \mid e \text{ is green active, } GC(e) = \{0,1\}, \text{rc}(e) = 0\}$;

(4) $GR := \{e \mid e \text{ is green active, } GC(e) = \{0,1\}, \text{rc}(e) > 0\}$.

The changes in these sets, as needed in Step 2a of *Red-and-Green Contour*, can be described in terms of a restriction by the range of the vertical right edge e:

(1′) $R' := R$, restricted to edges in the range of e;

(2′) $RG' := RG$, restricted as in (1′);

(3′) $G' := G$, restricted as in (1′);

(4′) $GR' := GR$, restricted as in (1′).

In total, the following information has to be maintained properly during the plane sweep:

(a) The set of active red points on the sweep line, and the set of active green points;

(b) The cover intervals for red-cover and green-cover values, and the red-cover and green-cover values themselves for these intervals;

(c) The sets R, G, RG, and GR.

In the next subsection, we demonstrate how this can be done efficiently, using a data structure which is a modification of the visibility tree [16].

2.1.3. *An Efficient Implementation*

We use a single data structure, an augmented visibility tree, for storing all information necessary during the sweep. The projections of the $2n$ horizontal edges of the given n rectangles on the y-axis yield $2n$ distinct y-values, $y_1, y_2, \ldots, y_{2n}$. These divide the y-axis into $2n + 1$ disjoint contiguous intervals $I_0 = (-\infty, y_1)$, $I_1 = [y_1, y_2), \ldots, I_i = [y_i, y_{i+1}), \ldots, I_{2n} = [y_{2n}, \infty)$. Let T be an ordered binary tree with $2n$ leaves and with minimal height; identify the leaves from left to right with $y_1, y_2, \ldots, y_{2n}$. Leaf y_i also represents interval I_i. Each internal node u represents an interval $I(u)$ which is the union of all intervals in the leaves of $T(u)$, the subtree of T rooted at u. That is, $I(u) = [y_i, y_j)$, where y_i is the leftmost leaf in $T(u)$, and y_{j-1} is the rightmost. Specifically, the root of T represents the interval $[y_1, +\infty)$. With each node u (internal or leaf node) of T we associate the following information:

(i) $I(u)$, the interval represented by u;

(ii) active(u), for leaf nodes $u = y_i$ only, indicating whether y_i is currently active;

(iii) $RC(u)$, the number of red intervals I_r (intersections of red rectangles with the sweep line) that cover $I(u)$, but do not cover $I(\text{parent}(u))$; that is, $RC(u) := |\{I_r | I(u) \subseteq I_r,\ I(\text{parent}(u)) \not\subseteq I_r\}|$;

(iv) $GC(u)$, similar to (iii) for green instead of red;

(v) $R(u)$, the set of active red points in $T(u)$ that correspond to r-edges with respect to $T(u)$, that is, if $T(u)$ is considered independently of T;

(vi) $G(u)$, the set of active green points in $T(u)$ that correspond to g-edges with respect to $T(u)$;

(vii) $RG(u)$, the set of active red points in $T(u)$ that correspond to rg-edges with respect to $T(u)$;

(viii) $GR(u)$, the set of active green points in $T(u)$ that correspond to gr-edges with respect to $T(u)$.

For each node u, $I(u)$ is set to its proper value at the beginning of the operation of the algorithm and it remains unchanged throughout; all other values may change. Initially, for each node u in T, active(u) := false, $RC(u) := GC(u) := 0$, and $R(u) := G(u) := RG(u) := GR(u) := \varnothing$. Note that R, G, RG, and GR of the preceding subsection are represented as R(root), G(root), RG(root), and GR(root).

Let I be a y-interval obtained by intersecting a rectangle with the sweep line. While the sweep line intersects that rectangle, information on I is stored in T at all nodes u for which $I(u) \subseteq I$ and $I(\text{parent}(u)) \not\subseteq I$. These nodes u are called the nodes of the canonical covering of I. The crucial observation behind this way of representing I is that only $O(\log n)$ nodes belong the canonical covering of I, and also only $O(\log n)$ nodes lie on the union of the paths from the root of T to the nodes in the canonical covering. Hence, starting at the root, the nodes of the canonical covering of I are found in $O(\log n)$ steps. We store an active rectangle in the corresponding cover values of the canonical covering nodes. If I stems from a red rectangle, say $I = I_r$, then I_r contributes an increment of one to $RC(u)$ for all canonical covering nodes u of I_r (similarly for I_g and $GC(u)$).

It is important to see how the sets $R(u)$, $G(u)$, $RG(u)$, and $GR(u)$ can be defined in terms of these sets for u's children, leftchild(u) and rightchild(u), provided the children exist:

```
(A) If $RC(u) = 0$ and $GC(u) = 0$
       then $R(u) := R(\text{leftchild}(u)) \cup R(\text{rightchild}(u))$;
            $G(u) := G(\text{leftchild}(u)) \cup G(\text{rightchild}(u))$;
            $RG(u) := RG(\text{leftchild}(u)) \cup RG(\text{rightchild}(u))$;
            $GR(u) := GR(\text{leftchild}(u)) \cup GR(\text{rightchild}(u))$;
(B) If $RC(u) > 0$ and $GC(u) = 0$
       then $R(u) := RG(u) := G(u) := \varnothing$;
            $GR(u) := G(\text{leftchild}(u)) \cup G(\text{rightchild}(u)) \cup GR(\text{leftchild}(u))$
                      $\cup\ GR(\text{rightchild}(u))$;
(C) Similarly for $RC(u) = 0$ and $GC(u) > 0$;
(D) If $RC(u) > 0$ and $GC(u) > 0$
       then $R(u) := G(u) := RG(u) := GR(u) := \varnothing$;
```

These relationships are crucial in that they allow efficient transitions at all sweep points, even though at first glance this might not appear to be the case. We describe below how the set union operations used in the above invariant can be implemented to take only constant time, instead of the linear time one might expect at first glance.

Let us illustrate the maintenance of T when the sweep line meets a left red vertical edge and when it meets a right red vertical edge; for green edges, the procedure is similar. Upon encountering a left vertical red edge with y-interval I_r, we change the information for all nodes u in the canonical covering in the following way:

```
{left vertical red edge}
1. $RC(u) := RC(u) + 1$;
2. {Update output edges}
   If $RC(v) = 0$ for all nodes $v$ in $T$ on the path from the root to $u$
   (but not including $u$) and $RC(u) = 1$
     then for all $e \in RG(u)$ do:
          terminate output of edge $e$;
          if $GC(v) = 0$ for all nodes $v$ in $T$ on the path from the root to $u$
          (including $u$)
          then {$I(u)$ is not covered by green}
            for all $e \in G(u)$ do:
              start output of edge $e$
          else {$I(u)$ is covered green: do nothing}
     else {$I(u)$ was already covered red: do nothing}
3. If $RC(u) = 1$
     then {locally in $T(u)$, $I(u)$ becomes red covered}
       maintain $R(u)$, $G(u)$, $RG(u)$, and $GR(u)$ according to $(B)$ or $(D)$, which-
         ever applies
     else {$I(u)$ was already covered red in $T(u)$: do nothing}.
```

```
{right vertical red edge}
1. $RC(u) := RC(u) - 1$;
2. If $RC(u) = 0$
     then {locally in $T(u)$, $I(u)$ is no longer covered red}
       maintain $R(u)$, $G(u)$, $RG(u)$, $GR(u)$ according to $(A)$ or $(C)$, whichever
       applies;
     else {$I(u)$ remains covered red: do nothing};
3. {Update output}
   If $RC(v) = 0$ for all nodes $v$ in $T$ on the path from the root to $u$,
   including $u$
     then {$I(u)$ is no longer covered red}
       for all $e \in RG(u)$ do
         start output of edge $e$;
       if $GC(v) = 0$ for all nodes $v$ in $T$ on the path from the root to $u$, including
       $u$,
         then {$I(u)$ is not covered green}
           for all $e \in G(u)$ do:
             terminate output of edge $e$
       else {$I(u)$ is covered green: do nothing}
   else {$I(u)$ is covered red: do nothing}.
```

It should be clear that the manipulation of the augmented visibility tree T is just an implementation of *Red-and-Green Contour*. Because there are at most $O(\log n)$ nodes in the union of all nodes on the paths from the root of T to all nodes u in the canonical covering of I_r, we can find in $O(\log n)$ steps the cover-values of the nodes on these paths, as required in the above description.

In addition to changing the status of active horizontal edges, we also have to insert (delete) active horizontal edges when the sweep line meets a left (right) vertical edge. The output of a new (old) edge can be properly started (terminated) by taking into account the cover values of nodes on the path from the root of T to the leaf, in the obvious way. Also, the classes to which a new (old) horizontal edge belongs for the nodes on this path can be easily updated by a bottom-up maintenance procedure, a straightforward generalization of the description in [16] for the contour case. For further (conceptually minor) details of the inner workings of the visibility tree consult [16].

Here, we simply want to mention that we use the techniques from [16] to form the unions of sets $R(u), G(u), RG(u), GR(u)$, for nodes u in T. Each of these sets is represented by two pointers to the two ends of a doubly linked list of points, representing intersections of horizontal edges on the sweeping line. With invariants (A), (B), (C), and (D) fulfilled initially and maintained throughout the algorithm, the union of two sets can be realized simply as the concatenation of two lists. Because sets $R(u)$, $G(u)$, $RG(u)$, and $GR(u)$ are disjoint and can be defined as unions of disjoint sets at u's children, the arguments of [16] apply.

2.1.4. *Time and Space Requirements*

As has been shown in the previous subsection, the augmented visibility tree supports the following operations within the stated time bounds:

- insertion/deletion of a horizontal edge: $O(\log n)$

• a single union operation for two disjoint sets as required by the algorithm: $O(1)$

and therefore

• maintenance of sets $R(u), G(u), RG(u), GR(u)$, for all nodes u in the canonical covering of a vertical edge, in total: $O(\log n)$

• additional cost for starting/terminating output for an edge of the red–green-contour (performed when a vertical edge is met): $O(1)$.

For the plane-sweep algorithm, sorting the edges with respect to x, and for the augmented visibility tree, sorting the edges with respect to y, can all be done in time $O(n \log n)$. As we have $O(n)$ insertion/deletion/maintenance operations, apart from the output, the algorithm runs in time $O(n \log n)$. For p output edges in total, we get the following theorem.

THEOREM 2.1. *The contour, in the form of contour cycles, of the intersection of two layers of rectangles (the Boolean AND operation) can be computed in time $O(n \log n + p)$, where n is the number of rectangles, and p is the number of edges in the contour, and with space $O(n)$. This is asymptotically optimal with respect to time and space.*

Proof. The time bound follows from the previous considerations. The space bound follows from the constant space used for each of the $O(n)$ nodes of the augmented visibility tree and because there are at most $O(n)$ edges in all intersected contour cycles at each sweep point (see [15]). Optimality follows from the optimality of the given time and space bounds for the contour problem (see [15]): Compute the contour for a set of rectangles by considering all of them to belong to one layer, and by computing the Boolean AND of that layer with one big rectangle enclosing all others. □

2.2. The Other Boolean Operators

The computation of Boolean operations on two layers of isothetic rectangles other than AND can be accomplished with essentially the algorithm described in Section 2.1, with only minor modifications on the output edges. Recall that for the Boolean AND, the output consists of exactly the edges in $RG \cup GR$. Now consider other Boolean operations and the output edges corresponding to them:

- red OR green has output edges $R \cup G$;
- red ANDNOT green has output edges $R \cup GR$;
- green ANDNOT red has output edges $G \cup RG$.

The important observation is that sets R, G, RG, and GR are just sufficient to describe the result of any Boolean combination of two layers, where we just refer to the covering of areas with layers, disregarding how often some area is covered by rectangles in a layer. Therefore we get the following.

THEOREM 2.2. *The contour, in the form of contour cycles, of the result of any Boolean operation, applied to two layers of rectangles, can be computed in time $O(n \log n + p)$, with space $O(n)$, where n is the number of rectangles and p is the number of contour edges. This is asymptotically optimal with respect to time and space.*

In fact, Boolean operations of the abovementioned types are the practically relevant ones, but not the only ones that can be computed. We can in the same way compute contours that can be specified by requirements on the number of times areas must be covered for each of the layers together with Boolean operators. Examples are the i-contour of a set of rectangles, or expressions like $(2 * red)$ANDNOT$(\geq 3 * green)$ with the obvious meaning. We do not, however, claim that these types of Boolean combinations are of any practical interest.

3. GENERALIZATIONS

Of the many conceivable generalizations, we simply want to mention briefly two fairly straightforward ones with perhaps some practical interest, namely, how to treat more than two layers, and how to deal with isothetic polygons.

3.1. More than Two Layers

When we are dealing with more than two layers, say k layers for $k > 2$, and we want to compute an arbitrary Boolean function of the rectangles in these layers, we simply need to consider more combinations of layers explicitly; that is, we need to maintain more disjoint subsets of active horizontal edges. To see this, consider a third layer, blue, in addition to red and green. A red horizontal edge can now enter a green area, as before, but it can also enter a blue area, or it can enter both. If it enters both, it does not matter, of course, in which order it has entered them.

In general, we get the following subsets of relevant active horizontal edges for k layers $l_1, l_2, \ldots, l_k$. Similar to the red-and-green layer case, we denote the class of active horizontal edges that forms part of the contour of layer l_i by L_i. Whenever an L_i-edge lies in an area covered by l_j, we let L_j follow L_i. Because the order of several layers in which an L_i-edge lies does not matter, we simply have a set of L_j's, $j \neq i$, following L_i. Let $\mathscr{L} = \{L_i | 1 \leq i \leq k\}$. Then we get all relevant classes of edges as follows:

```
for all $L_i$, $1 \leq i \leq k$, and
    for all subsets $S \subseteq \mathscr{L} - L_i$,
        $L_i S$ is a relevant class.
```

For example, returning to the two layer case with $L_1 = R$ and $L_2 = G$, we get all relevant classes in this way as:

$L_1 \emptyset$	corresponding to R;
$L_1\{L_2\}$,	corresponding to RG;
$L_2 \emptyset$	corresponding to G;
$L_2\{L_1\}$,	corresponding to GR.

The number of classes of edges we get in this way is 2^{k-1}, the number of subsets of $\mathscr{L} - L_i$, for each of the L_i, $1 \leq i \leq k$, totalling $k \cdot 2^{k-1}$ classes.

We again use the augmented visibility tree to store classes of active horizontal edges during the sweep. With each node u in the augmented visibility tree, we store a counter for the cover of each layer, denoted by L_iC for $1 \leq i \leq k$. In addition, we store two pointers to each of the $k \cdot 2^{2k-1}$ classes of active horizontal edges, where each class is represented by a doubly linked list. To see that the algorithm works in essentially the same way as before, let us describe the invariant for the classes stored at each node, denoted by (A), (B), (C), (D) in the two layer case. For node u, $L_iS(u)$ denotes class L_iS, as represented local to $T(u)$, that is, if we disregard the rest of T. Then the invariant is

(I) For all L_i for which $L_iC(u) = 0$: $L_iS(u)$ for $S = S_0 \cup S_1$, with S_0 such that, for all $L_j \in S_0$, $L_jC(u) = 0$, and $S_1 = \bigcup_{L_jC(u)>0} L_j$, is defined as

$$L_iS(u) := \bigcup_{v \in \{\text{leftchild}(u), \text{rightchild}(u)\}} L_i\left(S_0 \cup \bigcup_{S' \in S_1} S'\right)(v).$$

(II) For all L_i for which $L_iC(u) > 0$:

$$L_iS(u) := \varnothing, \text{ for all } S \subseteq \mathscr{L} - L_i;$$
$$L_jS(u) := \varnothing, \text{ for all } S \subseteq \mathscr{L} - L_i,\ L_j \neq L_i.$$

Obviously, invariant (A), (B), (C), (D) is a special case of (I), (II). The reader is invited to check how (A) is derivable from (I), (B) and (C) are derivable from (I) for GR and from (II) for R, RG, and G, and (D) is derivable from (II).

The operations of starting and terminating output edges can be generalized similarly. The time spent on processing a node in the visibility tree is, however, no longer a constant, but is, instead, proportional to the number of classes, $k \cdot 2^{k-1}$. We conclude with the following.

THEOREM 3.1. *The contour, in the form of contour cycles, of the result of Boolean operations applied to k layers of rectangles, can be computed in time* $O(k \cdot 2^k \cdot n \log n + p)$ *with* $O(k \cdot 2^k \cdot n)$ *space, where n is the number of rectangles, and p is the number of contour edges.*

Note that optimality in time and space can be claimed here only for a constant number of layers, but not in general, that is, not when k is an input parameter.

3.2. *Isothetic Polygons*

We just want to mention briefly that isothetic polygons can be handled by the visibility tree in basically the same way as rectangles. To see how this is done in the contour computation, consult [16]. In the same way, we get the following.

THEOREM 3.2. *The contour, in the form of contour cycles, of the result of Boolean operations, applied to k layers of isothetic polygons, can be computed in time* $O(k \cdot 2^k \cdot n \log n + p)$ *with* $O(k \cdot 2^k \cdot n)$ *space, where n is the number of contour edges.*

ACKNOWLEDGMENT

We thank Mrs. B. Beck for typesetting this paper.

REFERENCES

1. C. S. Chang, LSI layout checking using bipolar device recognition technique, in *Proceedings, 16th Design Automation Conference, 1979*, 95–101.
2. I. Dobes and R. Byrd, The automatic recognition of silicon gate transistor geometries: An LSI design aid program, in *Proceedings, 13th Design Automation Conference, 1976*, 327–335.
3. W. E. Donath and C. K. Wong, An efficient algorithm for Boolean mask operations, in *IEEE International Conference on Computer Design, 1983*, 358–360.

4. R. H. Güting, An optimal contour algorithm for iso-oriented rectangles, *J. Algorithms* **5**, 1984, 303–326.
5. U. Lauther, An $O(N \log N)$ algorithm for Boolean mask operations, in *Proceedings, 18th Design Automation Conference, 1981*, 555–562.
6. U. Lauther, Simple but fast algorithms for connectivity extraction and comparison in cell based VLSI design, in *Proceedings, ECCTD 80, 1980*, 508–514.
7. B. W. Lindsay and B. T. Preas, Design rule checking and analysis of IC mask designs, in *Proceedings, 13th Design Automation Conference, 1976*, 301–308.
8. W. Lipski and F. P. Preparata, Finding the contour of the union of a set of iso-oriented rectangles, *J. Algorithms* **1**, 1980, 235–246.
9. C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison–Wesley, Reading, MA, 1980.
10. J. Nievergelt and F.P. Preparata, Plane-sweep algorithms for intersecting geometric figures, *Commun. ACM* **25**, 1982, 739–747.
11. Th. Ottmann, P. Widmayer, and D. Wood, A fast algorithm for the Boolean masking problem, *Comput. Vision Graphics Image Process.* **30**, 1985, 249–268.
12. M. Pracchi, *Boolean Expressions of Rectilinear Polygons with VLSI Applications*, Technical Report R-978 UILU-ENG 82-2244 (ACT 36), University of Illinois-Urbana, 1982.
13. F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.
14. R. B. Tilove, Set membership classification: A unified approach to geometric intersection problems, *IEEE Trans. Comput.* **C-29**, 1980, 874–883.
15. P. Widmayer and D. Wood, Time- and Space-Optimal Contour Computation for a Set of Rectangles, *Inform. Process. Lett.* **24**, 1987, 335–338.
16. D. Wood, The contour problem for rectilinear polygons, *Inform. Process. Lett.* **19**, 1984, 229–236.
17. C. K. Yap, *Fast Algorithms for Boolean Operations on Rectilinear Regions*, Technical Report, Department of Computer Science, New York University, 1982.