UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

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

Peter Widmayer
Derick Wood

Data Structuring Group

CS-86-39

September, 1986

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

Peter Widmayer * Derick Wood †

September 11, 1986

Abstract

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. Also, the contour of the Boolean combination is computed in the form of contour cycles rather than single contour edges. We also 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.

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

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

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

practitioners and researchers have devoted their attention to the development and implementation of fast algorithms for the Boolean masking problem, see [1,2,3,6,5,7,9,11,12,17].

While the solutions described for the general case of arbitrary polygons seem to be reasonably efficient, the same cannot be said about the 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, 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. In other words, the algorithm for the general case spends computation time on intersection points that don't 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. Therefore, 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 rectangles in the plane, say a layer $\mathcal{R}$ of red ones and a layer $\mathcal{G}$ of green ones. Let $\mathcal{R} = \{\mathcal{R}_i | 1 \leq i \leq n_r\}$ and $\mathcal{G} = \{\mathcal{G}_j | 1 \leq j \leq n_g\}$, where $\mathcal{R}_i$ ($\mathcal{G}_j$) denotes a red (green) rectangle. Each rectangle is given by the four coordinates of its four sides, that is, by a quadruple $(x_{left}, x_{right}, y_{bottom}, y_{top})$. A rectangle represents a set of points in the plane in the usual way: rectangle $(x_{left}, x_{right}, y_{bottom}, y_{top})$ represents the set $\{(x,y) | x_{left} \leq x \leq x_{right},\ y_{bottom} \leq y \leq y_{top}\}$. For given $\mathcal{R}$ and $\mathcal{G}$ a point $p = (x,y)$ in the plane is called *red* (*green*) if there exists a rectangle $\mathcal{R}_i \in \mathcal{R}$ ($\mathcal{G}_j \in \mathcal{G}$) with $p \in \mathcal{R}_i$ ($p \in \mathcal{G}_j$).

The *contour* of $\mathcal{R}$ is a description of $\cup_{i=1}^{n_r} \mathcal{R}_i$, the set of all red points, in the form of an isothetic 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, [15] show 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 $\mathcal{R}$ and $\mathcal{G}$. That is, we show how to compute the contour of the set of points that are both red and green, that is $\bigcup_{i=1}^{n_r} \mathcal{R}_i \cap \bigcup_{j=1}^{n_g} \mathcal{G}_j$. All operators we use are *regularized* in the sense of [14] . This means the result of an operation is taken to be the closure of the interior of the non–regularized ("usual") operation. The effect of regularizing operations is to get rid of 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 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 will be sufficient to stop the sweep line at the vertical edges of rectangles; we call these x–values *sweep points* of the sweep line.

This well–known principle (see e.g. [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

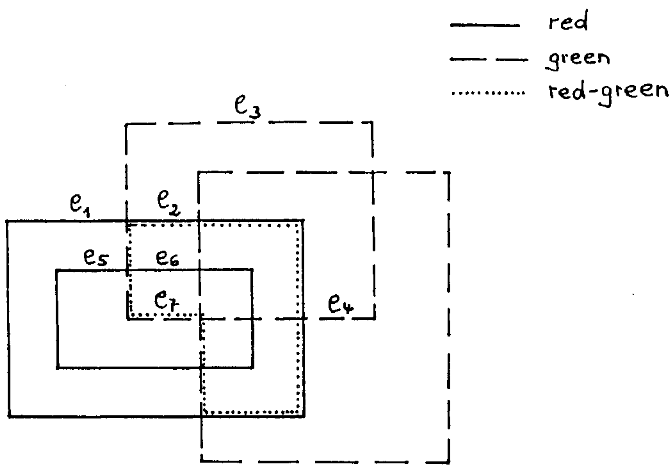

Figure 1: A red–green contour.

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 simply can afford to maintain all of the started but not yet terminated contour cycles without asymptotically extra 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 Figure 1.

A contour edge of the former type is called a *red–green–visible edge*, 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, but not a rg– or 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 Figure 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 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 $[y_{bottom}, y_{top}]$ are clearly untouched. So we restrict our view 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 the left red edge, hence the class of r–edges and the class of rg–edges both become empty. Any edge belonging to the green contour will still belong to the green contour behind the sweep point, but will also be in a red area. Hence, the class of gr–edges is defined as the union of the classes of gr–edges and g–edges; the class of g–edges becomes empty.

Second, consider the case that 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 that red rectangle may (partly) still be covered by some other red rectangle. We again restrict our view 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 Figure 2 at sweep point h of the sweep line, e_1 and e_2 become visible, whereas e_2 and e_3 don't. Of those edges becoming visible, some may become red–visible, others may become red–green visible. In our example in Figure 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.

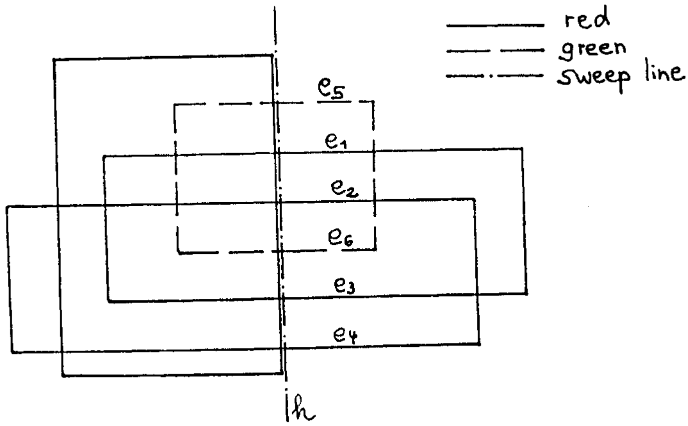

Figure 2: Horizontal edges becoming visible.

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 ist, becomes green–red–visible or becomes red–green–visible, an edge of the desired reg–green contour is started; whenever an edge is removed from those 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 $R = \{\mathcal{R}_i | 1 \leq i \leq n_r\}$ and
$\mathcal{G} = \{\mathcal{G}_j | 1 \leq j \leq n_g\}$ 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 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 edge e do: Let e be a red vertical edge (without loss of generality; for e being 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 bottom and top points. Let e_{bottom}, e_{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_{bottom}, e_{top} to the corresponding class of active horizontal edges, if any; and output start for those of the two edges falling into RG or GR.

   (b) If e is a right edge, then: Let R', RG' be the sets of edges that were not visible and now become red–visible, 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_{bottom}, e_{top} from the corresponding class of active horizontal edges, if any; and output end for those of the two edges 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 will have been removed. We, therefore, use a counter indicating 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 (similar for green):

   (a) If e is a left vertical edge, then: The bottom and top end points 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_{bottom} and y_{top}, increase the red–cover value by 1.

   (b) If e is a right vertical edge, then: The two end points, y_{bottom} and y_{top}, occur as interval boundaries in the set of red–cover intervals. Decrease the red–cover for all intervals between y_{bottom} and y_{top} by 1. Now the two intervals sharing endpoint y_{bottom} have the same red–cover value. Remove y_{bottom} and merge the two intervals into one, inheriting the red–cover from the merged intervals. Do the same for y_{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 iff 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 iff it lies within a green–cover interval with green–cover value > 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 Figure 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) $R := \{e | e \text{ is red active}, RC(e) = \{0,1\}, gc(e) = 0\}$;
(2) $RG := \{e | e \text{ is red active}, RC(e) = \{0,1\}, gc(e) > 0\}$;
(3) $G := \{e | e \text{ is green active}, GC(e) = \{0,1\}, rc(e) = 0\}$;
(4) $GR := \{e | e \text{ is green active}, GC(e) = \{0,1\}, rc(e) > 0\}$.

The changes of 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 line 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 yields $2n$ 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 don't cover $I(parent(u))$; that is, $RC(u) := |\{I_r | I(u) \subseteq I_r, I(parent(u)) \not\subseteq I_r\}|$;

(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 $R(u) := G(u) := RG(u) := GR(u) := \emptyset$. 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 $I(parent(u)) \not\subseteq I$. 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 to 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 $R(u) := R(leftchild(u)) \cup R(rightchild(u))$;
        $G(u) := G(leftchild(u)) \cup G(rightchild(u))$;
        $RG(u) := RG(leftchild(u)) \cup RG(rightchild(u))$;
        $GR(u) := GR(leftchild(u)) \cup GR(rightchild(u))$;
(B) If $RC(u) > 0$ and $GC(u) = 0$
    then $R(u) := RG(u) := G(u) := \emptyset$;
        $GR(u) := G(leftchild(u)) \cup G(rightchild(u)) \cup GR(leftchild(u))$
            $\cup GR(rightchild(u))$;
(C) Similarly for $RC(u) = 0$ and $GC(u) > \emptyset$;
(D) If $RC(u) > 0$ and $GC(u) > 0$
    then $R(u) := G(u) := RG(u) := GR(u) := \emptyset$;

These relationships are crucial in that they allow efficient transitions at all sweep line positions, even though at first glance this might not appear to be the case. We shall 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), whichever 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 also consult [16].

Here, we simply want to mention that we use the technique 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 on 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 following 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 any of the $O(n)$ nodes of the augmented visibility tree, and the fact that there are at most $O(n)$ edges in all intersected contour cycles at any stage in the sweep (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 of 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 of course not matter in which order it has entered them.

In general, we get the following subsets of relevant active horizontal edges for k layers $l_1, l_2, \ldots, l_k$. Similarly 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 doesn't matter, we simply have a set of L_j's, $j \neq i$, following L_i. Let $\mathrm{L} = \{L_i | 1 \leq i \leq k\}$. Then we get all relevant classes of edges as follows:

for all $L_i, 1 \leq i \leq k$, and
  for all subsets $S \subseteq \mathrm{L} - L_i$,
    $L_i S$ 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 $\mathrm{L} - L_i$, for each of the L_i, $1 \leq i \leq k$, totalling to $k \cdot 2^{k-1}$ 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 $k \cdot 2^{k-1}$ 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 the following :

(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 $\forall L_j \in S_0, L_jC(u) = 0$, and $S_1 = \bigcup_{L_jC(u)>0} L_j$, is defined as
$L_iS(u) := \cup_{v\in\{leftchild(u), rightchild(u)\}} L_i(S_0 \cup \bigcup_{S'\in S_1} S')(v)$

(II) For all L_i for which $L_iC(u) > 0$:
$L_iS(u) := \emptyset,\ \forall S \subseteq \mathrm{L} - L_i;$
$L_jS(u) := \emptyset,\ \forall S \subseteq \mathrm{L} - L_i, L_j \neq L_i.$

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 similarly generalized. 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.*

4 Acknowledgement

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

References

[1] C.S. Chang. LSI layout checking using bipolar device recognition technique. In *Proceedings of the 16th Design Automation Conference*, pages 95–101, 1979.

[2] I. Dobes and R. Byrd. The automatic recognition of silicon gate transistor geometries: an LSI design aid program. In *Proceedings of the 13th Design Automation Conference*, pages 327–335, 1976.

[3] W.E. Donath and C.K. Wong. An efficient algorithm for Boolean mask operations. In *IEEE International Conference on Computer Design*, pages 358–360, 1983.

[4] R.H. Güting. An optimal contour algorithm for iso-oriented rectangles. *Journal of Algorithms*, 5:303–326, 1984.

[5] U. Lauther. An $O(N \log N)$ algorithm for Boolean mask operations. In *Proceedings of the 18th Design Automation Conference*, pages 555–562, 1981.

[6] U. Lauther. Simple but fast algorithms for connectivity extraction and comparison in cell based VLSI design. In *Proceedings of ECCTD 80*, pages 508–514, 1980.

[7] B.W. Lindsay and B.T. Preas. Design rule checking and analysis of IC mask designs. In *Proceedings of the 13th Design Automation Conference*, pages 301–308, 1976.

[8] W. Lipski and F.P. Preparata. Finding the contour of the union of a set of iso-oriented rectangles. *Journal of Algorithms*, 1:235–246, 1980.

[9] C.A. Mead and L.A. Conway. *Introduction to VLSI Systems*. Addison-Wesley Publishing, Co., Reading, Massachusetts, 1980.

[10] J. Nievergelt and F.P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25:739–747, 1982.

[11] Th. Ottmann, P. Widmayer, and D. Wood. A fast algorithm for the Boolean masking problem. *Computer Vision, Graphics, and Image Processing*, 30:249–268, 1985.

[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 Transactions on Computers*, C-29:874–883, 1980.

[15] P. Widmayer and D. Wood. *Time– and Space–Optimal Contour Computation for a Set of Rectangles.* Technical Report, University of Waterloo, Department of Computer Science, 1986.

[16] D. Wood. The contour problem for rectilinear polygons. *Information Processing Letters*, 19:229–236, 1984.

[17] C.K. Yap. *Fast Algorithms for Boolean Operations on Rectilinear Regions.* Technical Report , Department of Computer Science, New York University, 1982.