UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

A Fast Algorithm for Boolean Mask Operations

Thomas Ottmann
Peter Widmayer
Derick Wood

Data Structuring Group
CS-82-37

November, 1982

UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT

A FAST ALGORITHM FOR BOOLEAN MASK OPERATIONS†

Thomas Ottmann
Peter Widmayer

Institut für angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe
West Germany

Derick Wood

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

ABSTRACT

We present an algorithm for the calculation of Boolean combinations between layers of a VLSI circuit layout. Each layer is assumed to contain only polygons, which are specified by their edges; the output is also polygonal. The algorithm makes heavy use of some results in computational geometry and is conjectured to be space and time optimal. It runs in $O((n+k)(r+\log n))$ time and the $O(n \cdot r)$ space, where n is the maximum number of edges on any layer, k is the number of edge intersections, and r is the number of layers. We prove that when the polygons are presented using a hierarchical description language the problem becomes NP-hard. Finally we discuss how this approach can be used to solve the i-contour-problem of computational geometry and the hidden line problem of computer graphics.

1. INTRODUCTION

On the one hand calculating Boolean mask combinations, for example **AND, OR, XOR, ANDNOT**, between different layers of VLSI circuit designs is basic to design rule checking, connectivity checking and device recognition, for

† This work was carried out under NATO grant No. RG 155.81 and the work of the third author was additionally supported by a Natural Sciences and Engineering Research Council of Canada grant No. A-7700.

example see [C], [DB], [L1], [L2] and [LP] and [MC]. Thus it is important that such computations be carried out efficiently. On the other hand, the investigation of such problems is an important aspect of the rapidly blossoming area of computational geometry. A number of problems motivated by VLSI design rule checking have already been studied, see [BO1], [BW], [LiPr], [NP] and [OW], for example. However, in each of these studies it is assumed that a single set of objects is given, that is a single layer. Lauther [L2] is the first author to study collections of sets of objects using present state-or-the-art results in computational geometry. In the present paper we provide an algorithm which improves on that of [L2].

We assume that each set of objects, that is each layer, is in fact a set of polygons. Each polygon is specified by its edges, where the edges have an orientation such that the region inside the polygon is to the right of each edge. Furthermore each edge has a colour indicating which layer it belongs to. In the detailed discussion of Section 2 we consider two layers, red and green, together with the Boolean operation **AND**, in order to provide a clear presentation of the basic ideas. In Section 3 we explain how these ideas can be extended for r layers and arbitrary "regularized" Boolean mask combinations, while in Section 4 we consider the effect of describing the polygons hierarchically, see [MC], [BO2] and [OW]. Finally, in Section 5 we discuss how these ideas can be applied to the hidden-line problem of computer graphics [NS], and to the i-contour problem.

2. THE INTERSECTION ALGORITHM

In our description of an algorithm for determining the contour of the intersection of two sets of simple polygons we assume that the reader is familiar with the Bentley-Ottmann algorithm for reporting the intersecting pairs of line segments in the plane [BO1].

We assume furthermore that the set of simple polygons is given as a set of oriented edges of the polygons, where each edge is described by its two end-points, the orientation, and the colour. The interior of the polygon lies to the right of the oriented edge. Assume - for simplicity only - that no vertical edges occur (this can always be achieved by a rotation of the coordinate system), we will talk about left - and right-oriented edges. An edge is left-oriented if the x-value of the starting point is greater than the x-value of the terminating point of the edge. One set of polygons is assumed to be colored red and the other green.

The task of the algorithm is to produce as output a set of oriented edges, describing the intersection of the set of red polygons with the set of green polygons; we will consider what Tilove [T] calls regularized intersection.

We will first give a general description of the algorithm, and later look at it in detail and discuss its complexity.

2.1. The Outline

An edge or a part of it (no matter which colour, but with the original orientation) is output by the algorithm, if it forms the boundary between a region covered by at least one red and at least one green polygon and a region for which this is not the case.

To determine all boundary-edges we use the sweeping line paradigm, see [BO1], [NP] for example. We sweep a vertical line in the horizontal direction through the plane, keeping track of the covering of all regions under consideration. At each scan-line position, the active edges are totally y-ordered, and so are the intervals between them. A change in the covering of a region can only occur when a change in the total order of all active edges occurs. This is possible exactly at the starting, terminating and intersection points of edges. Consequently, the crucial positions of the sweeping line (the sweeping points) are exactly those of the Bentley-Ottmann algorithm. The region between two adjacent scan line positions and two adjacent active edges is called an elementary region, since for all points in such a region, the covering with red and green polygons is the same. For a given scan line position, we associate the elementary regions with the edges currently active, speaking of active regions below, above and between edges, according to the y-order of edges (and active elementary regions). Besides carrying out the actions and maintaining the information of the Bentley-Ottmann algorithm we also keep track of the covering of each elementary region by means of a counter for the number of green and a counter for the number of red polygons covering the region. At each step of the sweep, a scan line data structure stores all currently active edges in y-order, together with these two counters for each active elementary region. A move of the scanline from one position to the next consists of the update of the active edges, the update of the counters for the active elementary regions, and the output of (parts

of) edges. The nature of polygons and polygonal intersections ensures that only local changes of the boundary occur together with local changes of edges at the sweeping points. We show in detail in the next subsection that such changes and the necessary housekeeping are also local. We must, however, be careful to output the correct starting and terminating points of parts of edges. We do this by using an output buffer for each edge. Whenever an edge becomes a boundary edge, it is put into its output buffer, together with the current sweeping-point, indicating the starting point of the output edge; whenever an edge ceases to be a boundary edge, it is output from the buffer, together with the current sweeping point, indicating the terminating point of the output edge.

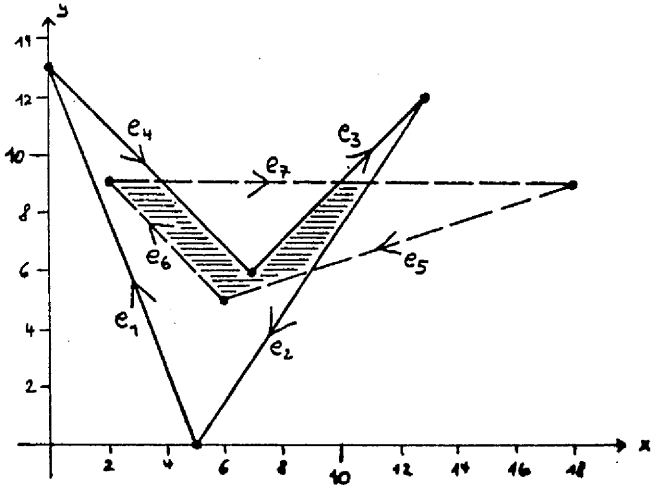
2.2. The Details

For the sake of simplicity we assume that at each sweeping point only one change occurs; in the case that several changes occur, they must simply be carried out in the appropriate order. The sweeping points of the scan-line are determined in the same way as in [BO1], and all actions carried out there are also carried out in our algorithm, without further notice. However, the data structure maintained for the scan-line is slightly different: in addition to the actual edges in ascending y-order we store the two counters RED and GREEN for each pair of adjacent actual edges, representing the number of red and green polygons covering the elementary region between the edges, respectively. The additional actions to be carried out at a sweeping point are the updating of counters, the preparation of edges for output by means of the output buffer, and the output of a buffer.

The following example gives an input and the desired output of the algorithm, and provides instances of most of the different situations which may occur in the algorithm.

INPUT: [$e_i = ((x, y), (x', y')$, orientation, colour)]
$e_1 = ((5,0), (0,13)$, left, green),
$e_2 = ((5,0), (13,12)$, left, green),
$e_3 = ((7,6), (13,12)$, right, green),
$e_4 = ((0,13), (7,6)$, right, green),
$e_5 = ((6,5), (18,9)$, left, red),
$e_6 = ((6,5), (2,9)$, left, red),
$e_7 = ((2,9), (18,9)$, right, red).

Graphic Representation:

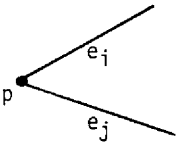


OUTPUT: $E_1 = ((2,9), (4,9), \text{right})$,
$E_2 = ((4,9), (7,6), \text{right})$,
$E_3 = ((7,6), (10,9), \text{right})$,
$E_4 = ((10,9), (11,9), \text{right})$,
$E_5 = ((11,9), (9,6), \text{left})$,
$E_6 = ((9,6), (6,5), \text{left})$,
$E_7 = ((6,5), (2,9), \text{left})$.

Our use of positive integers for the input and output points is for the sake of simplicity of presentation only; in general, rational numbers may occur.

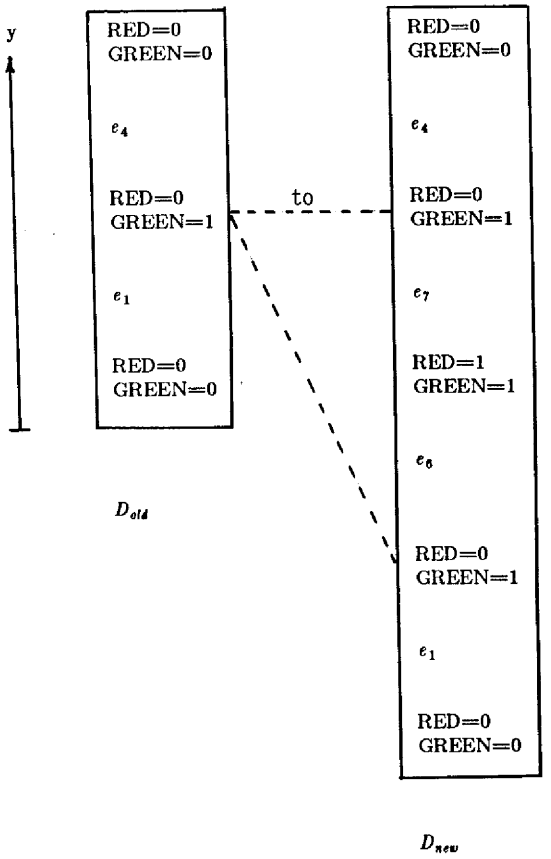
We describe the actions to be carried out by the algorithm for each of the four possible cases.

(1) a polygon point with two edges lying to the right is encountered:



Both edges must be of the same colour and must have different orientations. Both edges are inserted into the data structure D representing the scan line at the position determined by the y-value of p. The edges divide the region, in which they fall, into two regions with unchanged counter values, and additionally squeeze in a new region between e_i and e_j, where the counter of the colour of the edges is incremented by 1 if e_i is oriented right, and otherwise decremented by 1. The counter of the other colour remains unchanged.

This is in our example the case for point (2, 9) and edges e_7, e_6. The scan line changes at that position from



The only edges that may become boundary edges (or cease to be boundary edges) are e_i and e_j : they are both (partially) boundary edges if and only if

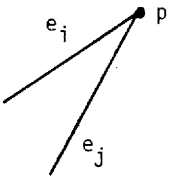
((RED > 0) **AND** (GREEN > 0) above e_i and below e_j)
AND (NOT((RED > 0) **AND** (GREEN > 0)) between e_i and e_j)
OR ((RED > 0) **AND** (GREEN > 0) between e_i and e_j)
AND (NOT((RED > 0) **AND** (GREEN > 0)) above e_i and below e_j)

In our example, both e_7 and e_6 are boundary-edges (at least partially, as will be seen later).

If both e_i and e_j become boundary edges, they are put in their output buffers, together with their starting point p .

We will work out the other cases in descending degree of detailed presentation:

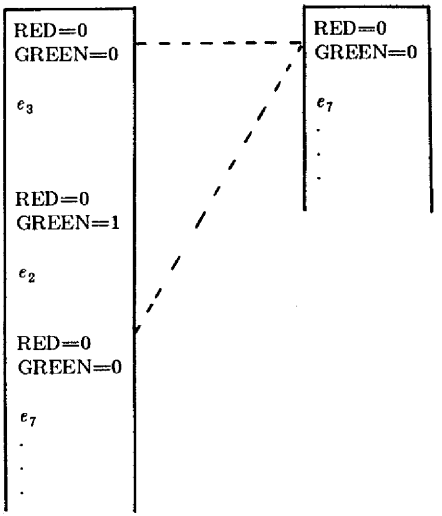
(2) a polygon point with two edges lying to the left is encountered:



In our example, one such point is (13, 12), where e_3 and e_2 meet.

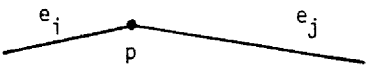
Both edges again have to be of the same colour and must have different orientations. They must be adjacent in the scan-line data structure D. Both edges and their counters in between are removed from D . The counters above e_i and below e_j must have the same values; the two regions are just merged into one with the old counters.

In our example:



If both edges were boundary edges, their buffers, containing the starting points, are output, together with terminating point p.

(3) a polygon point with one edge to the left and one edge to the right is encountered:



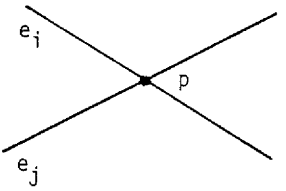
In our example, this holds for point (7, 6) and edges e_4, e_3.

Again, both edges are of the same colour. They also must have the same

orientation. In D, e_j replaces e_i; all counters remain unchanged.

If e_i was a boundary edge, then its buffer is output, together with endpoint p, and e_j is put into its buffer, together with starting point p.

(4) a (formerly detected) intersection point of two edges is encountered:



In advance, nothing can be said about the colours and orientations. Let us distinguish between the different cases:

(a) **Both edges have the same colour**:
(Our example provides no instance of this case.)

(i) Both edges have the same orientation:
We exchange edges e_i and e_j in D. All counters remain the same. If one of e_i, e_j was a boundary edge, then its buffer is output, together with the terminating point p, and the other edge of e_i, e_j becomes the boundary edge and is thus inserted in the buffer with starting point p.

(ii) Both edges have different orientations:
We exchange edges e_i and e_j in D, and update the counter for the region between e_i and e_j: The counter for the colour of the edges is incremented by 2, if e_i is oriented left, and is decremented by 2 otherwise. If both edges were boundary edges, their buffers are output, together with terminating point p; if none of them was a boundary edge, they may both become boundary-edges. They are checked for this property, and they become boundary edges they are put in their buffers, together with their starting point p.

(b) **Both edges have different colours**:

(i) Both edges have the same orientation:
In our example, point (4, 9) and edges e_4, e_7 provide an instance of this case.

We exchange edges e_i and e_j in D, and update the counter for the region between e_i and e_j. The counter for one of the colours is incremented by 1, the colour for the other is decremented by 1:
if ((e_i is green) **AND** (e_i is oriented right)) **OR** ((e_i is red) **AND** (e_i is oriented left))
then $\text{RED}_{NEW} := \text{RED}_{OLD} + 1$ and
$\text{GREEN}_{NEW} := \text{GREEN}_{OLD} - 1$,
else $\text{RED}_{NEW} := \text{RED}_{OLD} - 1$ and
$\text{GREEN}_{NEW} := \text{GREEN}_{OLD} + 1$.

In our example, e_i is green (e_i is e_4), and is oriented right, so the **then**-clause applies.

The counters for regions above and below p remain unchanged, but are necessary to determine whether any of the edges changed its boundary edge status. We leave the straightforward but tedious task of examining at all possible cases to the interested reader.

(ii) Both edges have different orientations:
In our example, point (11, 9) and edges e_7, e_2 provide an instance of this case.

We exchange e_i and e_j in D, and update the counter for the region between e_i and e_j. If e_i is oriented left, then both counter values are incremented by 1, otherwise both counter values are decremented by 1.

Again, the counters for the regions above and below e_i and e_j remain unchanged, but are necessary to determine which of e_i, e_j change their boundary edge status. If only one edge of e_i, e_j was a boundary edge, then this edge remains as a boundary edge, and if both were boundary edges, both cease to be boundary edges. If, however, none of them were boundary edges, then either none of them are or both become boundary edges. We leave the details again to the reader.

2.3. The Complexity

The number of sweeping points of the scan-line is, as in [BO1], $O(n+k)$, where n is the number of edges given in the input and k is the number of intersection points found. Like Bentley-Ottmann, we can accomplish every manipulation of the scan line data structure D in time $O(\log n)$, since all actions are local and counters always refer to the region between edges. A balanced binary search tree might be used for this purpose, for example, where edges are stored in the leaves, and between any two adjacent leaves the counters are stored with double links. As any intersection point creates at most two additional boundary edges, the number p of output edges (edge parts) is bounded by $O(k)$. The output buffer itself is bounded in size by $O(n)$, as only one part of an input edge can be prepared for output and thus stored in the

buffer at any given time. The access to the output buffer elements is possible in constant time, for example, by use of an array. Applying the economic storage technique described in Brown [B], we conclude:

Theorem 1: The problem of reporting the intersection of two given sets of simple polygons can be solved in time $O((n+k)\log n)$ and with space $O(n)$, where n is the number of input edges and k the number of edge intersections.

This complexity is the best we can hope for with the present state of the art, as the Bentley-Ottmann algorithm itself has the same complexity.

3. GENERALIZATIONS

The efficiency of the algorithm presented in Section 2 depends neither on the fact that the intersection (the Boolean **AND**) is the only boolean operation involved nor on the restriction to two colours, red and green. On the contrary, only the nature of polygons is crucial in order to maintain all necessary information with only local changes in the scan-line data structure D.

We may therefore generalize our algorithm to apply to any partition of a set P of polygons (into different colours, layers, etc.) and the application of any boolean operators which are reasonable for sets of polygons, that is under which the set of simple polygons is closed. As Tilove [T] points out, we should deal with regularized Boolean operators to be mathematically precise, rather than with set operators. A regularized operator yields as a result of the operation a regularized set of polygons, that is, a set of polygons without isolated points and dangling lines (a polygon is regularized by taking its topological interior and applying the topological closure operation to it, for details see [T]).

Let us assume that the set of polygons P, is partitioned into r (disjoint) subsets $P_1, P_2, ..., P_r$, representing the different colours, layers, etc.

We define a Boolean function, $CLASS(P_i, S)$, to denote for each set S of points in the plane the property of belonging to the subset P_i of polygons: $CLASS(P_i, S)$ is true if and only if for all points $p \in S$, p belongs to P_i. In other words, $CLASS(P_i, S) = (S \subseteq \bigcup_{p \in P_i} \{p\})$.

A Boolean function, $OUTPUT(P_1, ..., P_r, S)$, characterizes for a partition of the set of polygons and a set S of points whether S belongs to the output of the algorithm. $OUTPUT$ is restricted to be a function over the functions $CLASS(P_1, S), ..., CLASS(P_r, S)$, using only regularized Boolean operators under which the set of simple polygons is closed, for example the operators **AND, OR, ANDNOT**, and **XOR**. For example, partitioning the set of polygons into subsets of red, green and blue polygons, one possibility for an $OUTPUT$ function is

$$OUTPUT(P_{red}, P_{green}, P_{blue}, S) = CLASS(P_{red}, S) \textbf{ AND } CLASS(P_{green}, S) \textbf{ OR } CLASS(P_{red}, S) \textbf{ ANDNOT } CLASS(P_{blue}, S).$$

Although the expression used to describe the output may deal with any number of operators and any number of subsets of P, it can be handled in exactly the same way as the simple intersection operation is handled by our algorithm. We need only convince ourselves that the classification of (a part of) an edge as (a part of) an output edge is possible with only local operations in the scan-line data structure D. For this purpose, we just maintain a counter $c(P_i, S')$ for each of the r properties $CLASS(P_i, S')$ of the set S' of all points in the region between any two adjacent edges in D. At each step of the computation, $c(P_i, S')$ denotes the number of polygons from P_i covering region S'. Obviously, $CLASS(P_i, S')$ for such a region S' can be computed from the counter information: $CLASS(P_i, S') = (c(P_i, S') > 0)$ for any region S' between adjacent edges. We call such a region S' an output region, if and

only if $OUTPUT(P_1, ..., P_r, S')$ is true. Hence, (a part of) an edge e is (a part of) a boundary edge, if and only if the region above e is an output region, and the region below e is not, or vice versa. Hence, being a boundary edge or not can be determined for any given edge e in D by just evaluating the $OUTPUT$ function for the regions above and below e and comparing the results. Accounting for the function evaluation with time complexity $O(r)$ and for the space needed to store the r properties of the regions between any two adjacent edges in the scan-line data structure D with space complexity $O(n \cdot r)$, we conclude:

Theorem 2: The problem of reporting the contour of the set of polygons determined by the application of some regularized Boolean operators to r sets of polygons can be solved in time $O((n+k)(r+\log n))$ and with space $O(n \cdot r)$, with n is the total number of input edges, and k the number of intersections between edges.

In many applications, including VLSI design, r is a small constant. In this case, our algorithm has time complexity $O((n+k)\log n)$ and needs $O(n)$ space, which is again the best we can expect with the present state of the art. For some computing environments it is a considerable advantage that the algorithm needs only one sweep of the scan-line through the plane, for example if the input polygon edges are stored in sorted order on an external device with sequential access, and central memory is rather limited.

4. Boolean Mask Operations for Hierarchical Input

It is well known that for many applications of geometric algorithms the input data are not as the collection of objects, which most algorithms require. For example in VLSI design and in computer graphics (see for example [MC], [NS]) hierarchical descriptions of geometric objects are used. Bentley and Ottmann [BO2] were the first to investigate the theoretical implications of hierarchical input for the complexity of geometric algorithms. We assume the reader to be familiar with [BO2], and we present a hierarchical language for the specification of polygons in the plane in a similar way to that chosen for rectangles in [BO2].

A simple regular polygon is described by denoting the end points of its edge, in clockwise order. Each end point is denoted only once, and between any two adjacent end points p_i , and p_j it is understood that an edge leads from p_i to p_j ; additionally, an edge leads from the last point to the first (in the given order). We use the notation

$$\text{POLYGON}(\{<\text{point}>\}_3^*)^\dagger$$

where each point is given by its (x, y)-coordinates. For example

$$\text{POLYGON}((6,1),(3,1),(1,4),(6,4),(2,3))$$

denotes the polygon depicted in Figure 4.1.

† The meta-notation $\{x\}_i^*$ denotes an arbitrary large sequence of x's containing a least three x's.

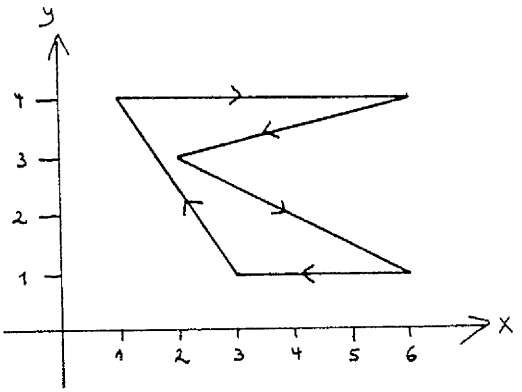


Figure 4.1

A description of a polygon in this form is called a polygon command:

polygon command: POLYGON($\{<\text{point}>\}_3^*$)

A set of polygons is described by a sequence of polygon commands, augmented with an identifying number for the set. The description of a set of polygons, together with its number, is called a symbol:

symbol: <symbol number> : $\{<command>\}_1^*$

For example

1 : POLYGON ((1, 1), (1, 2), (2, 1))
POLYGON ((0, 0), (0, 3), (3, 0))

is a description of a set of two polygons and is referred to as symbol 1.

Once a set of polygons is defined, it may be used as part of another set of polygons. With its use, a translation of the set is allowed: the "old" origin may be positioned relatively to a given "new" origin. Notationally, this is done by a call of the corresponding symbol (by its number) in the form of a draw command:

draw command: DRAW <symbol number> AT <point>

For example,

DRAW 1 AT (0, 5)

refers to the set of polygons denoted by symbol number 1, where the polygons are positioned so that the "old" origin of the coordinate system lies at point (0, 5).

Draw commands may also be used as parts of symbol definitions, provided that the called symbol has already been defined:

command: polygon command | draw command

Hence, calls of symbols defined later than the calling symbol and recursive calls are not allowed. To control the calling structure easily, we associate ascending natural numbers with the symbols and allow calls to symbols with smaller numbers only.

A collection of symbol definitions in the defined hierarchical input language (HIL) is called an HIL description:

HIL description: $\{<\text{symbol}>\}_1^*$.

An HIL description denotes the set of polygons denoted by the symbol with the highest symbol number. This allows us to "forget" about intermediate steps in the description.

For example,

1: POLYGON ((0, 0), (0, 1), (1, 0))
2: DRAW 1 AT (0, 0)
  DRAW 1 AT (1, 1)
3: DRAW 2 AT (0, 1)
  DRAW 2 AT (2, 0) .

denotes the set of polygons depicted in Figure 4.2.

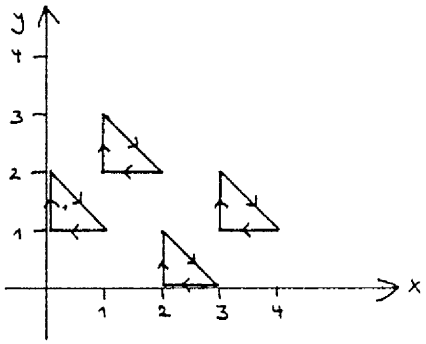


Figure 4.2

Note that the origin HIL description (see [BO2]) deals with rectangles rather than polygons and is thus a special case of the HIL description given here.

Let $L(D)$ be the set of polygons described by an HIL description D. The Boolean Mask Problem in terms of HIL is the problem of constructing an HIL description of a set P of polygons denoted by a Boolean expression using operators **AND, OR, ANDNOT, XOR** on some sets $P_1, ..., P_k$ of polygons. Each set of polygons $P_i\,(i = 1, ..., k)$ is given by an HIL description D_i :

Boolean Mask Problem:

Input: $k \in \boldsymbol{N}$; k HIL descriptions $D_1, ..., D_k$ of sets of polygons $L(D_1), ..., L(D_k)$, and an expression $\mathrm{expr}(L(D_1), ..., L(D_k))$ on the $L(D_i)\,(i \in \{1, ..., k\})$ using only regularized Boolean operators from {**AND**, **ANDNOT**, **OR**, **XOR**}.

Output: An HIL description D such that $L(D) = \mathrm{expr}(L(D_1), ..., L(D_k))$.

As this problem is quite complex because many HIL descriptions are involved as input, and an arbitrary complex expression may be given, we consider a simpler problem, whose complexity provides a lower bound for the Boolean Mask Problem. We restrict our attention to two given HIL descriptions and an expression with one of the regularized Boolean operations:

Restricted Boolean Mask Problem:

Input: Two HIL descriptions D_1, and D_2 of sets of polygons $L(D_1)$, and $L(D_2)$, and a Boolean operator $op \in \{\textbf{AND}, \textbf{ANDNOT}, \textbf{OR}, \textbf{XOR}\}$.

Output: An HIL description D such that $L(D) = L(D_1)\ op\ L(D_2)$.

The Restricted Boolean Mask Problem is at least as difficult to solve as any one of the four subproblems, where each subproblem involves exactly one of the four different Boolean operators. That is, each subproblem's complexity provides a lower bound for the original problem. To simplify the analysis, we restrict the problem further: instead of searching for a description D, we restrict our attention to whether or not $L(D)$ is empty. For an HIL description D, the emptiness of $L(D)$ (and also the non-emptiness) is of course decidable in polynomial time. To see this, observe that a syntactically correct HIL description consists of at least one symbol, and each symbol consists of at least one command. Draw commands may only refer to symbols with smaller number. This implies that at least the symbol with smallest number contains only polygon commands. Furthermore, each of the sequences of symbols starting with the symbol with highest number and continuing with a symbol referenced by a draw command must end at a symbol containing only polygon commands. Hence, any syntactically correct HIL-description describes a nonempty set of polygons. We conclude that the complexity of the non-emptiness and emptiness problems for $L(D)$ is a lower bound for the complexity of our original problem. We shall prove that the nonemptiness question for the operator **AND** to be NP-complete, thus showing that both the Restricted Boolean Mask Problem and the Boolean Mask Problem are NP-hard.

Let us look at the

Conjunction Nonemptiness Question:

Input: Two HIL descriptions D_1 and D_2.

Output: Yes, if and only if $(L(D_1)\ \textbf{AND}\ L(D_2)) \neq 0$.

Complexity: NP-complete.

Proof:

(1) The problem is in NP.
Guess two polygons, $p_1 \in L(D_1)$ and $p_2 \in L(D_2)$, and a point p interior, (that is not on the boundary, to both p_1 and p_2. Guess the polygons in the form of a sequence of commands in D_1 and D_2, respectively. The the verification that $p_1 \in L(D_1)$ and $p_2 \in L(D_2)$ is easy: just follow the commands in the given sequence and calculate the position of the respective polygon. Testing whether or not point p lies inside polygon p_i $(i \in \{1,2\})$ consists of checking for each edge of p_i whether p lies on the its side; this can of course be done in time proportional to the number of edges of p_i.

(2) The problem is NP-hard.
A lower bound for the complexity of our problem can be obtained by looking at a special case. Let the description deal with rectangles only (instead of polygons in general) and let D_2 describe one single rectangle. Substituting $\{x\}$ for $L(D_2)$ we obtain the answer yes if and only if $L(D_1) \cap \{x\} \neq 0$. This is exactly the problem of the non-empty intersection of $L(D_1)$ with a query rectangle x which [BO2] have shown to be NP-complete.

Though the observations show that the Reskricted Boolean Mast Problem is NP-hard, we are interested in the question's complexity for any of the operators considered. We now know that **AND**- operations are NP-hard, but what about the others? Let's treat them, one by one:

ANDNOT:

The emptiness question for $(L(D_1)\ \mathbf{ANDNOT}\ L(D_2))$ reduces to the construction of a description D such that $L(D) = (L(D_1)\ \mathbf{ANDNOT}\ L(D_2))$: once we know D, we can decide the emptiness of $L(D)$ easily. We prove the NP hardness of the construction by proving that the emptiness-question is NP-hard:

ANDNOT-Emptiness Question:

Input: Two HIL descriptions D_1 and D_2.

Output: Yes, if and only if $(L(D_1)\ \mathbf{ANDNOT}\ L(D_2)) = 0$.

Complexity: NP-hard.

Proof: We prove this by reducing the well known (cf. for example [GJ]) NP-complete subset sum problem to the conjunction nonemptiness question:

Subset Sum Problem:

Input: A (multi-)set W of n positive integers, $W = \{w_1, ..., w_n\}$, and a positive integer T.

Output: Yes, if and only if a subset $W' \subseteq W$ such that $\sum_{w \in W'} w = T$.

We give an HIL description D_2, where each possible subset's sum is represented by some polygon (we use a small triangle, as in the previous example) along the y-axis at the y-position representing the subset sum. The other HIL description D_1, denotes a single polygon of the same kind on the y-axis at position T. Then there is a solution to the subset sum problem if and only if in $L(D_2)$ a polygon occurs at height T. This is the case if and only if $(L(D_1))$ **ANDNOT** $L(D_2))$ is empty. The demonstration that a description such as D_2 can be given with polynomial length (in the number n of integers in W) is given in detail in [BO2] and [BOW] for unit squares. We apply exactly the same technique yielding D_2 to consist of symbols

0: POLYGON ((0, 0), (0, 1), (1, 0))

and for each $i \in \{1, ..., n\}$

i: DRAW i–1 AT (0, 0)
DRAW i–1 AT (0, w_i)

OR

The disjunction of two sets of polygons is described easily, if we don't impose restrictions such as intersection-freeness on the polygons within an HIL description or some sort of minimal representation of covered area. In this case, the **OR**-operation is just a concatenation of two descriptions. Let's assume for simplicity that two descriptions D_1 and D_2 are given with disjoint symbol numbers, say numbers 1 up to n for D_1 and numbers $n+1$ up to n' for D_2. Then we construct a description D such that $L(D) = L(D_1)$ **OR** $L(D_2)$ simply by concatenating D_1 and D_2 and adding the additional symbol

$n'+1$: DRAW n AT (0, 0)
DRAW n' AT (0, 0).

Hence, the disjunction construction problem is solvable in constant time.

Note that if we have to make the symbol numbers disjoint, that is we have to choose new numbers for the symbols, we can still solve the problem in linear time.

If, however, we deal with HIL descriptions D, where polygons within $L(D)$ are not allowed to intersect (having some "minimal" representation of covered area in mind), the disjunction construction problem becomes NP-hard,

too. The intersection-freeness requirement forces us to find a representation D_3 for $L(D_3) = (L(D_1)$ **OR** $L(D_2))$ such that polygons within $L(D_3)$ don't intersect. "Intersection" is understood in the usual way. A polygon denoted a multiple times by an HIL description D counts only once in $L(D)$, in accordance with the usual semantic of sets, and therefore does not constitute an intersection. For example, the following description D' fulfills the intersection-freeness restriction:

```
D'   0:   POLYGON ((0, 0), (0, 1), (1, 0))
     1:   DRAW 0 AT (0, 0)
          DRAW 0 AT (0, 0) .
```

The disjunction construction problem is shown to be NP-hard by a reduction of the problem of whether the disjunction of two HIL descriptions describes a set of exactly one polygon. To convince the reader that for a given description D it is decidable in polynomial time whether or not $|L(D)| = 1$, let us assume that no unreferenced symbols occur in D_2 (if they do occur they may be removed by a linear scan, starting at the symbol with highest number). Let us further assume that no symbols consisting of a single draw command occur (if they occur, they may be removed, and every reference to them changed to refer to the appropriate symbol with the appropriate translation of coordinates). Let us call "bottom symbols" all symbols containing only polygon commands (they form the bottom of the call structure). To test whether $L(D)$ describes a set of exactly one polygon, observe that if any of the symbols in D' describes a set of more than one polygon, then $|L(D)| > 1$. We therefore organize the test in a bottom-to-top manner in the call structure: starting with the bottom symbols, we decide whether the set of polygons denoted by a symbol has cardinality more than one. Associating with each symbol the polygon it denotes, we can answer the question of a calling symbol s by combining the polygon information of the symbols called from s : if the polygons of the called symbols, together with their translations, are all the same, then s denotes the same polygon, too. We continue the procedure only as long as no symbol denoting more than one polygon occurs. Either we find such a symbol, then $|L(D)| > 1$, or we don't, in which case $|L(D)| = 1$. In either case, we obtain the answer to our original question after at most one simple test for each of the symbols, that is in polynomial time.

We call the problem of whether or not $|L(D)| = 1$ the singleton problem, and we conclude that if the singleton problem for the disjunction of two HIL descriptions is NP-hard, then the construction of the description of the disjunction is also NP-hard. We demonstrate the former as follows:

Disjunction Singleton Problem;

Input: Two HIL descriptions D_1 and D_2.

Output: Yes, if and only if $|(L(D_1)\ \mathbf{OR}\ L(D_2))| = 1$.

Complexity: NP-hard.

Proof: Again, we reduce the subset sum problem to our problem. Let $W = \{w_1, \ldots, w_n\}$ be the set of n positive integers, and let T be the positive integer from the subset sum problem's input. Then we construct D_1 to denote a set of unit squares along the y-axis with y-values corresponding to all subset sums of W. We construct D_2 to denote the two rectangles of unit length and height from y-value $T+1$ to y-value $\sum_{i=1}^{n} w_i + 1$. This yields the following two descriptions:

D_1:

0: POLYGON ((0, 0), (0, 1), (1, 1), (1, 0))
1: DRAW 0 AT (0, 0)
DRAW 0 AT (0, w_1)
2: DRAW 1 AT (0, 0)
DRAW 1 AT (0, w_2)
.
.
.
i: DRAW i-1 AT (0, 0)
DRAW i-1 AT (0, w_i)
.
.
.
n: DRAW n-1 AT (0, 0)
DRAW n-1 AT (0, w_n).

and

D_2: 1: POLYGON ((0, 0), (0, T), (1, T), (1, 0))
POLYGON ((0, $T+1$), (0, $\sum_{i=1}^{n} w_i$), (1, $\sum_{i=1}^{n} w_i$), (1, $T+1$)).

Then we have the following reduction of the subset sum problem to the disjunction singleton problem:

The subset sum problem has a solution
if and only if there is a unit square at position (0, T) in $L(D_1)$
if and only if $|(L(D_1)\ \mathbf{OR}\ L(D_2))| = 1$, because the unit square at (0, T) fills exactly the "hole" between the two polygons from $L(D_2)$.

XOR:

The construction of a description D such that $L(D) = L(D_1)$ **XOR** $L(D_2)$ is shown to be an NP-hard problem by a reduction of the emptiness problem for $L(D_1)$ **XOR** $L(D_2)$:

> to decide the emptiness of $L(D_1)$ **XOR** $L(D_2)$ we construct D as above and decide D's emptiness straightforwardly. The **XOR** emptiness problem remains to be shown NP-hard:

XOR-emptiness question:

Input: Two HIL-descriptions D_1 and D_2 .

Output: Yes, if and only if $(L(D_1)$ **XOR** $L(D_2)) = 0$.

Complexity: NP-hard.

Proof: We give a reduction of the subset sum problem to our problem in a way similar to that for OR:

Let D_1 be the description of unit squares along the y-axis at the heights of all subsums, i.e. chose D_1 exactly as described for the disjunction singleton question.

Let D_2 be a call of the symbol with the highest number of D_1 , plus an additional unit square at height T:

D_2:

0: ⎫
. ⎪
. ⎬ as for D_1
. ⎪
n: ⎭

$n+1$: DRAW n AT (0, 0)
POLYGON ((0, T), (0, $T+1$), (1, $T+1$), (1, T)) .

The only possible difference between $L(D_1)$ and $L(D_2)$ is the unit square at height T: this is really a difference if and only if no subset sum yields T. Hence, $L(D_1)$ **XOR** $L(D_2)$ describes the empty set of polygons if and only if the subset sum problem has a solution. This reduction of the subset sum problem to the **XOR** emptiness problem proves the latter to be NP-hard.

Thus we have proved.

Theorem 3: The Boolean Mask Problem for hierarchical input is NP-hard for each of the operators **AND, ANDNOT, OR,** and, **XOR**.

Proof: By the previous considerations.

5. FURTHER APPLICATIONS

5.1. The i-Contour Problem

The i-contour of a set of polygons is the contour of the region covered by at least i polygons. The description of the contour is a description of non-intersecting polygons, having at most a finite number of points in common. This implies that for every two polygons no common boundary of positive length may exist. The contour is thus a "minimal" representation of the covered area.

Given a set of polygons, we can computer the 1-contour easily by applying the given boolean mask algorithm, where we have only one relevant layer (a second one is considered empty) and the operation **OR** is to be performed.

In order to be able to solve the general problem we just slightly modify the test of whether an edge is an output edge: (a part of) an edge is (a part of) an output edge if and only if exactly one of both adjacent counters (we deal with one layer only) has a value greater than or equal to i. Obviously, this algorithm runs in time $O((n+k)\log n)$ and needs $O(n)$ storage.

The fact that the counter for each elementary region provides all necessary information enables us to solve all related problems within the same bounds. For example, looking for the contour of all regions covered by exactly i polygons or for the contour of all regions covered by at most i polygons, simply change the test for output edges accordingly.

5.2. The Hidden Line Elimination Problem

Given a set of non-intersecting solids in 3D-space, bounded by plane polygonal faces, we want to produce a 2-dimensional, for example on a screen of what an observer sees, looking from a specified position in some specified direction.

In order to determine all visible parts of lines, we deal with the 2-dimensional projection of the faces onto a "viewing plane" through the observer's position, normal to the viewing direction (for a detailed presentation of the problem see [SSS] or [OWW]). A scan of the viewing plane is accomplished in a fashion similar to that applied for the Boolean mask operations. However, instead of a constant number of layers we have faces of solids at arbitrary distances from the observer. Instead of a counter for each layer within each elementary region we maintain information about the distances of all faces covering the region. A dynamic version of a segment-tree-like data structure allows the necessary operations to be carried out efficiently; for details see [OWW].

6. REFERENCES

[B] Brown, K.Q.: Comments on 'Algorithms for Reporting and Counting Geometric Intersections'; *IEEE Transactions on Computers* C-30, (1981), 147-148.

[BO1] Bentley, J.L. and Ottmann, Th.: Algorithms for Reporting and Counting Geometric Intersections; *IEEE Transactions on Computers*, C-28, (1979), 643-647.

[BO2] Bentley, J.L. and Ottmann, Th.: The Complexity of Manipulating Hierarchically defined sets of Rectangles; *Proc. 10th Symposium on MFCS*, Štrbské Pleso, *Springer-Verlag Lecture Notes in Computer Science*, (1981), 1-15.

[BOW] Bentley, J.L., Ottmann, Th. and Widmayer, P.: The Complexity of Manipulating Hierarchically Defined Sets of Rectangles; to appear in *Advances in Computing Research*, edited by F.P. Preparata, 1983.

[BW] Bentley, J.L. and Wood, D.: An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles; *IEEE Transactions on Computers*, C-29, (1980), 571-577.

[C] Chang, C.S.: LSI Layout Checking Using Bipolar Device Recognition Technique; *Proceedings of the 16th Design Automation Conference*, San Francisco, (1979), 95-101.

[DB] Dobes, I. and Byrd, R.: The Automatic Recognition of Silicon Gate Transistor Geometries: An LSI Design Aid Program; *Proceedings of the 13th Design Automation Conference*, San Francisco, (1976), 327-335.

[GJ] Garey, M.R. and Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W.H. Freeman and Company, San Francisco, 1979.

[L1] Lauther, U.: Simple but Fast Algorithms for Connectivity Extraction and Comparison in Cell Based VLSI Designs; *Proceedings of the ECCTD 80*, Warsaw, 1980, 508-514.

[L2] Lauther, U.: An $O(N \log N)$ Algorithm for Boolean Mask Operations; *Proceedings of the 18th Design Automation Conferencef*, Nashville, (1981), 555-562.

[LP] Lindsay, B.W. and Preas, B.T.: Design Rule Checking and Analysis of IC Mask Designs; *Proceedings of the 13th Design Automation Conference*, San Francisco, (1976), 301-308.

[LiPr] Lipski, W. and Preparata, F.P.: Finding the Contour of a Union of Iso-Oriented Rectangles; *Journal of Algorithms*, 1, (1980), 235-246.

[MC] Mead, C.A. and Conway, L.A.: *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass. 1980.

[NP] Nievergelt, J. and Preparata, F.P.: Plane-Sweep Algorithms for Intersecting Geometric Figures; *Communications of the ACM*, 25, (1982), 739-747.

[NS] Newman, W. and Sproull, R.: *Principles of Interactive Computer Graphics*, McGraw-Hill, 2nd edition, New York, 1979.

[SSS] Sutherland, I.E., Sproull, R.F. and Schumacker, R.A.: A Characterization of Ten Hidden-Surface Algorithms; *Computing Surveys*, 6, (1974), 1-55.

[T] Tilove, R.B.: Set Membership Classification: A Unified Approach to Geometric Intersection Problems; *IEEE Transactions on Computers*, C-29, (1980), 874-883.

[OW] Ottmann, Th. and Widmayer, P.: Reasonable Encodings Make Rectangle Problems Hard; Report 106, Universität Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Karlsruhe, 1981.

[OWW] Ottmann, Th., Widmayer, P. and Wood, D.: A Worst-Case Efficient Algorithm for Hidden-Line Elimination; Computer Science Technical Report CS-82-33, University of Waterloo, Waterloo.