THE INSIDE STORY ON SELF-INTERSECTING POLYGONS

Martin E. Newell, Xerox Palo Alto Research Center **Carlo H. Sequin,** University of California, Berkeley

A convention for interpreting general polygons is proposed, together with algorithms for processing such polygons. This convention, as well as the algorithms for dealing with it, has been designed to fulfill the requirements found in processing the geometry of IC masks.

The Case for Polygons in IC Mask Description

Several geometric primitives have been developed for use in describing the geometry of IC masks. These primitives are oriented towards efficient description of the types of geometric constructs typically encountered. They include *rectangles*, *wires*, *flashes*, and *polygons*. For example, Caltech Intermediate Form, CIF (Mead and Conway 1980), a mask description language now in widespread use in the university community, makes use of all of these constructs.

The term "primitive" is not strictly applicable, since the first three of these can be generated as polygons, unless circular flashes and circular ends and corners of wires are required; even then only circular flashes and polygons are needed. Since polygons can be used to represent most constructs, there is a need to be able to handle polygons in the most general way. Stated another way, a system that can handle general polygons is able to deal with most constructs met in IC mask specification. Other primitives may be added for efficiency reasons, but a basic polygon capability is essential. Unfortunately, most specifications of mask geometry either prohibit general polygons or do not adequately specify how they should be interpreted. The latter is true of the definition of CIF (Mead and Conway 1980).

We will define a *simple polygon* to mean a polygon having a simple, (i.e., non-self-intersecting), boundary. The processing of convex, simple polygons is straightforward. The interpretation of *inside* and *outside* may be made in the obvious way, and scan conversion algorithms can rely on there being only one pair of edges to consider at a time. However, such polygons are not sufficiently general. They do not allow the representation of such common shapes as an L. Furthermore, a restriction against self-intersection prohibits the simplest specification of some shapes such as the outline of a self-intersecting wire, or certain types of alignment marks, Figure 1. Also, shapes with holes have to be specified as a combination of several polygons that overlap each other. Even if an initial specification of such self-intersecting polygons is prohibited, they can arise from the *expansion* of simple polygons (Sequin and Newell) for the purposes of compensating for photolithographic distortions and for accommodating fabrication design rules, and so techniques for handling the self-intersection case are necessary.

This paper presents a convention for the determination of the inside of polygons that suits the requirements for processing the geometry of IC masks. Algorithms for processing such polygons in various ways are also presented.

Inside and Outside—Existing Conventions

Several conventions are in common use for determining whether a given point is to be considered inside or outside of a polygon. These include the *Surrounded*, the *Parity*, and the *Oriented Multiply-Covered* conventions. Each of these is briefly examined below.

Surrounded Convention

A conceptually straightforward approach would be to declare that all points within the outermost boundary of a polygon are inside the polygon. While this convention works well for simple polygons, and even for some self-intersecting ones, as in Figure 1, it has the unfortunate consequence that it is not possible, using a single polygon, to define any structure having a hole; the inner area would always be filled. This convention has the advantage that it is easy to implement on a raster scanned device.

Parity Convention

A common convention used in many computer graphics packages determines the state of a point by the parity of the number of intersections between edges of the polygon and a straight line drawn from the point to infinity in any direction. Again, this convention works well for simple polygons, and for some self-intersecting polygons. Moreover, it allows the definition of polygons including holes, and is straightforward to implement. The main problem with this convention is that while it provides a perfectly consistent definition of inside and outside in the abstract, in the domain of IC mask specification it is counterintuitive. For example, if a polygon is being used to represent the outline of a wire that crosses itself, then the area of self-intersection is not filled in, as in Figure 2; yet if the wire crosses this region a third time, it is filled in. Material does not behave like that. This is more than an idealistic objection; as was mentioned above, a simple polygon can become self-intersecting as the result of being expanded. In such a case, it would be wrong to leave a hole where the two expanded regions overlap.

Oriented Multiply-Covered Convention

Another convention in use in some circles involves

defining an orientation for the boundary of a polygon such that, for example, counterclockwise boundaries define material and clockwise boundaries define holes, as in Figure 3. Material that overlaps material is simply material. Each hole is able to annihilate exactly one layer of material. Moreover, holes in space are ignored. This convention, like the others, works well for simple polygons, except that now care must be taken to ensure that the orientation of polygons is counterclockwise. Polygons with holes can be accommodated without awkward self-intersecting boundaries—the hole may be defined simply as another polygon defined in a clockwise direction. Among the disadvantages of this convention is that the polygon of Figure 1 will have only one loop filled in, the one whose boundary is counterclockwise. Furthermore, whenever a mirror operation is performed, care must be taken to reverse the order of definition of every polygon (or to reverse the clockwise/counterclockwise convention). In summary, the main objection to this convention is the asymmetry it exhibits. There is much to be said for a convention under which it is possible to determine inside and outside from a picture of the edges, without knowing in what order the vertices were defined and how many times the polygon has been mirrored.

From the above discussion desirable properties for an inside~outside convention can be identified:

- It should be possible to define polygons involving concavities and holes.
- Multiply-covered regions should be treated intuitively; for example, if a polygon is used to describe the outline of a wire, then regions where the wire crosses itself should be filled in.
- It should be possible to determine inside and outside from a picture of the edges of a polygon, without knowing in what order the vertices were defined or how many times the polygon has been mirrored.

The Nonzero Winding Number Convention

Another way of describing the *Oriented Multiply-Covered* convention above is to say that a point is considered to be inside the polygon if the *winding number* of the boundary with respect to that point is strictly positive (not zero or negative). The winding number, sometimes called the *wrap number*, of a boundary with respect to a given point is defined as the net number of times that a point on the boundary wraps around the given point while the boundary point makes one complete traversal of the boundary. The asymmetry of the *Oriented Multiply-Covered* convention results directly from the asymmetry of the condition on the winding number. To overcome this, the proposed *Nonzero Winding Number* convention specifies that:

A point is considered to be inside the polygon if the winding number of the boundary with respect to that point is nonzero.

This convention comes close to satisfying all of the criteria given in the previous section. Both loops of the polygon in Figure 1 are filled in. Reversal of the order of definition of the vertices has no effect on which regions are considered inside. Multiply covered regions are filled in unless, by self-intersection, the orientation of part of the polygon is reversed, in which case it cancels material. Note that this behavior is very similar to

FIGURE 1. Self-intersecting polygon

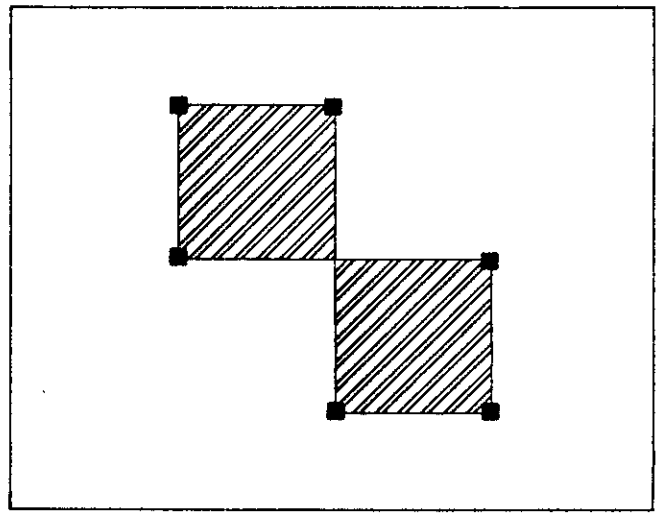


FIGURE 2. Parity convention

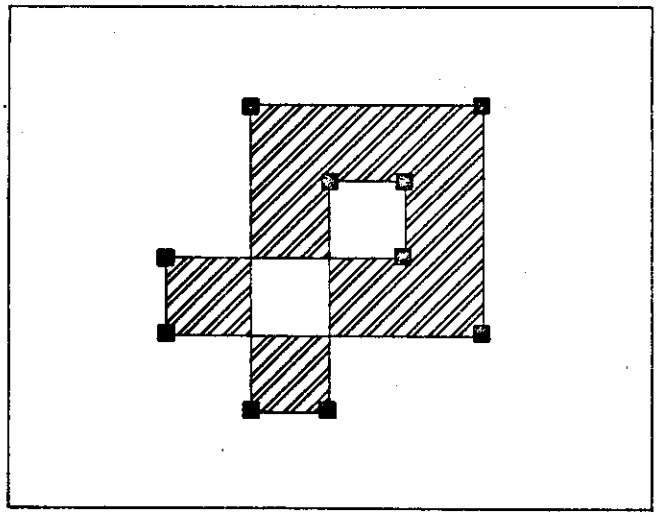
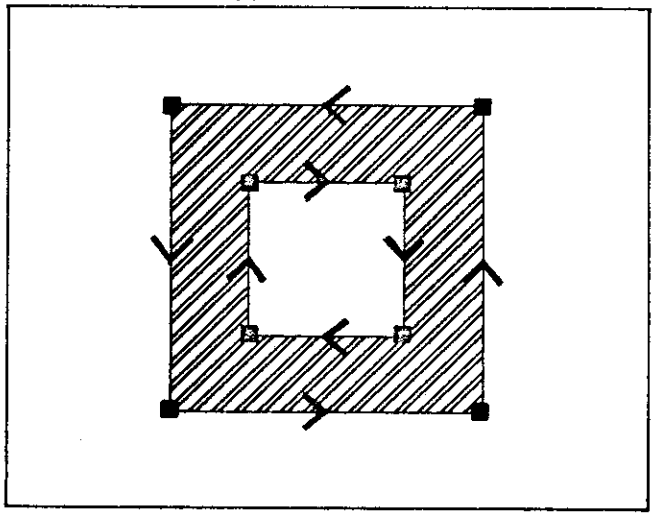


FIGURE 3. Oriented multiply-covered



that of the *Oriented Multiply-Covered* convention, except that no a priori specification is made concerning which direction defines material and which defines holes.

The following interpretation of winding number may help to clarify the convention. Consider representing the boundary of the polygon using a closed loop of string on a board. To determine whether a given point is outside the polygon, stick a pin into the board at that point and attempt to pull the string from around the pin. If you succeed then the winding number of the string with respect to the pin was zero, and the point was outside the polygon.

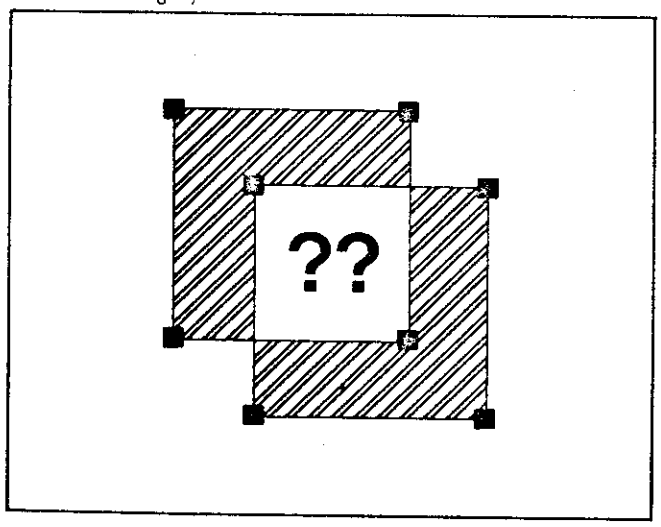
The only case in which this convention does not satisfy all the criteria given is that of a polygon with multiple disjoint boundaries, where two or more of the component polygons overlap. Such polygons are useful for defining holes as separate boundaries of the same polygon, or when the separate disjoint pieces are most conveniently thought of as parts of a single object. For consistent interpretation it is necessary to know the relative direction of definition of the multiple boundaries, even though reversing all directions will still not affect which regions are considered to be inside. For example, consider a polygon that has two disjoint boundaries, each of which is simple, as in Figure 4. If the two regions thus defined should overlap, then the region of overlap will be considered either inside or outside depending on whether the two boundaries were defined in the same or different orientations, respectively.

An interesting, if unusual, example of the application of the *Nonzero Winding Number* convention is shown in Figure 5-b. The result may be justified as follows: the hole, B, in Figure 5-b is to be expected on the basis that it is topologically similar to the hole, A, in Figure 5-a. The hole, C, in Figure 5-b is to be expected from the symmetry of the figure about the line SS.

Algorithm for Direct Raster Conversion

Implementation of the *Nonzero Winding Number* convention for output on raster scanned devices is reasonably straightforward. We shall define the x and y directions to be those of faster and slower scan, respectively. The boundary of the polygon is stored as an *edge list* of *oriented* edges, each edge being tagged with a direction (+1 or -1) to indicate whether it was defined in increasing or decreasing y value. These edges are sorted on their lowermost (minimum y) end. An *active list* is maintained to hold the subset of edges that intersect the current scan line. The current x coordinate of the intersection with the current scan line of each edge in the active list is maintained with each edge, and the edges in the active list are kept sorted by this x coordinate. The change, *dx*, in x for a unit change in y is also held with the edge. The basic loop for stepping from one scan line to the next, and for generating dots for output is as follows (in a language of dubious lineage):

FIGURE 4. Ambiguity

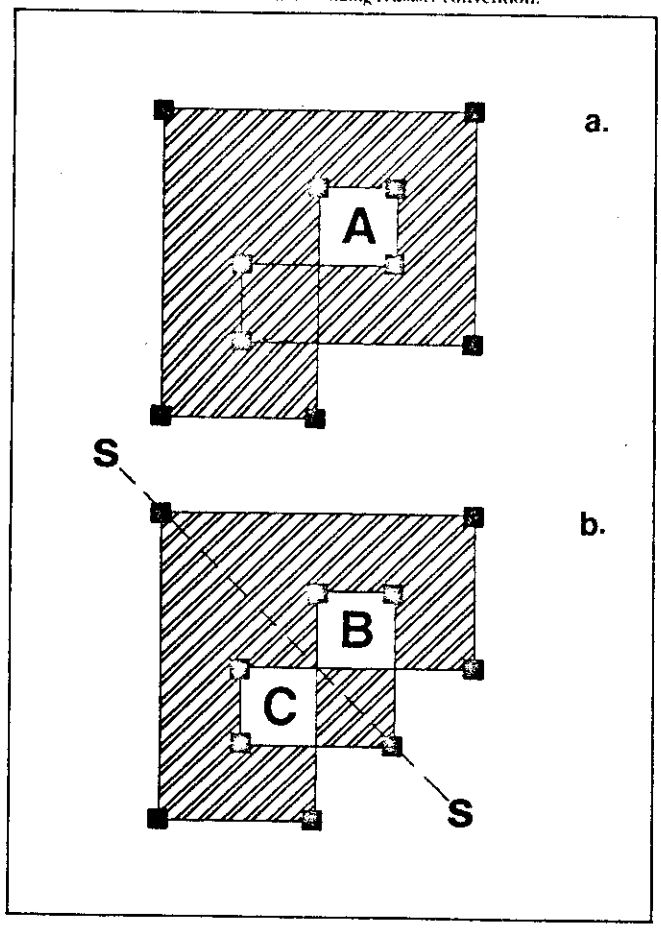



```
until active-list empty and edge-list empty do
  begin
{deal with initial entry or disjoint boundaries:}
  if active-list empty then ycurr ← ymin(first edge in
                                          edge-list)
{introduce edges from edge list into active list:}
  foreach edge in edge-list while ymin(edge)=ycurr do
    begin
    transfer edge from edge-list to active-list
    xcurr(edge) ← x(lower-end(edge))
    end
  sort active-list on xcurr
{generate regions to be filled in on current scan line:}
  wrap ← 0
  left ← first edge in active-list
  foreach edge in active-list do
    begin
    wrap ← wrap + direction(edge)
    if wrap=0 then
        begin
        FillInLineSegment(xcurr(left),xcurr(edge))
        left ← next edge in active-list
        end
    end
  ycurr ← ycurr + 1
{update edges in active list:}
  foreach edge in active-list do
    begin
    if y(upper-end(edge)) = ycurr
        then remove-edge-from-active-list
        else xcurr(edge) ← xcurr(edge) + dx(edge)
    end
  end
```

Further Polygon Processing Requirements

The direct scan conversion of polygons, while useful for such applications as plotting, does not provide a basis for the many other operations that often have to be carried out on polygons. These operation include such processes as *clipping*, for removing parts of the image that lie outside of the region of interest in display and plotting applications; *expanding* and *shrinking*, for the

FIGURE 5. A result of the *Nonzero Winding Number* convention.



purposes of compensating for photolithographic distortions and for accommodating fabrication design rules; *design rule checking*, that requires a knowledge of inside and outside, and may use expansion or contraction techniques to check for adequate clearance or overlap; *fast display*, in which area filling hardware cannot be used if polygons are first scan converted to separate scan lines; *conversion to input for mask-making machines*, which requires either rectangles or trapezoids, the latter becoming far more widespread with the increasing use of electron beam machines.

All of these operations can benefit from techniques for dealing with general polygons at a level higher than a raster converted image. For most of the above processes, algorithms exist for handling mask specifications involving only simple, convex polygons. To use these algorithms for general polygons, the given polygons must first be reduced to a set of convex polygons, preferably nonoverlapping.

Algorithm for Conversion to Trapezoids

Some of the processes listed in the previous section can benefit greatly from the reduction of a given polygon into a particular set of convex polygons, namely, nonoverlapping convex trapezoids having top and bottom edges parallel to the fast scan, or x, direction. An algorithm for generating a minimum set of such trapezoids from a general polygon is presented here. The input and data structures are the same as for the scan conversion algorithm above. The main difference is that instead of stepping from each scanline to the next one, larger steps are made to the next y value on which something "interesting" happens, and then trapezoids are output only where relevant, not necessarily on every scan line, and not necessarily even for every interesting y value. The basic loop of the algorithm is:

```
until active-list empty and edge-list empty do
    begin
    if active-list empty then ycurr ← ymin(first  edge in
                                          edge-list)
    introduce-edges-from-edge-list-into-active-list
    find-ynext
    generate-trapezoids-between-ycurr-and-ynext
    remove-terminating-edges-from-active-list
    ycurr ← ynext
    end
```

where the functions of the called procedures are as follows:

introduce-edges-from-edge-list-into-active-list:
transfer edges that intersect ycurr, as in the raster conversion algorithm. The edges in the active list are then sorted in x.

find-ynext:
this is the minimum of 3 values:

1. The lowest upper y in active-list;
2. The y coordinate of the lower end of the next edge in edge-list;
3. The lowest intersection between edges in active-list. Edges that intersect can be detected by checking if their x values at the currently proposed ynext are in sort. If a pair of edges is found out of sort then their intersection is computed and the current proposal for ynext is backed down to the y value of the intersection, and the check for edges being in sort at the new ynext is started again from the beginning. Note that if an intersection defines ynext, it is necessary to create two new edges between the intersection point and the upper ends of the intersecting pair, and to merge these back into the edge-list; it is also necessary to cut back the intersecting pair to the intersection point.

generate-trapezoids-between-ycurr-and-ynext:
create a trapezoid for each pair of edges that border on a region of zero wrap number; this is analogous to the part labeled *"generate regions to be filled in on current scan line"* in the scan conversion algorithm.

remove-terminating-edges-from-active-list:
these are edges that do not extend above ynext.

This algorithm, as it stands, does not generate a minimum set of trapezoids, since an entire *swath* of trapezoids is produced for every ycurr~ynext pair of horizontal lines. To overcome this shortcoming, instead of simply outputting all the trapezoids created in one swath, they are saved. However, when an edge of a newly created trapezoid is found to be already in use in

the definition of an existing trapezoid from an earlier swath, this existing trapezoid is output. A trapezoid is also output if one of its edges is about to terminate at ynext.

This scheme can be implemented in various ways. For example, the edges that define trapezoids (i.e., that bound regions having zero wrap number), are each assigned a *mate*, that is, the other edge with which the given edge has been paired to define a trapezoid. Trapezoids are output whenever the pairing of mates changes from one swath to the next. Zero, one, or two trapezoids may be output, one for each of the newly paired edges that already had a mate. The mates are marked as single after their trapezoids have been output, to avoid generating the same trapezoid twice.

An example is shown in Figure 6. In swath 1 edges H and G are paired, but no trapezoid is output. In swath 2 edges F and E are newly paired, but still no output is generated. In swath 3 it is found necessary to pair F with D. Since F's existing mate is not D the trapezoid between F and its existing mate, E, is output. F is then paired with D. In swath 4 it is found necessary to pair H with B, and C with G. Since H's existing mate is not B, the trapezoid between H and its existing mate, G, is output, and both H and G are marked single. H is then paired with B. C will also be paired with G, but no trapezoid will be generated because by now G has no existing mate. In swath 5 it is found necessary to pair edges C and D. This will cause two trapezoids to be generated since C and D are both changing mates. These two trapezoids are those between C and G and between F and D. Also in swath 5 the trapezoid between H and B is output because at least one of those edges (in this case, H) terminates at the top of the swath. In swath 6 the trapezoid between A and B is output because both of these edges terminate at the top of the swath. Finally the trapezoid between C and D is output in swath 7 for the same reason.

A serious precision difficulty exists in this algorithm in the computation of intersections. This arises because intersections have to be computed in two different ways: as x(y) to obtain the x coordinate of an edge where it crosses a horizontal line, and as Intersection(e1,e2) to find the intersection of two edges. Great care must be taken to ensure that subsequent comparisons of values resulting from these computations are free from inconsistencies due to rounding errors. Incidentally, a convenient formulation for computing the intersection, (x, y), of two edges, $\mathbf{v}_1,\mathbf{v}_2$ and $\mathbf{v}_3,\mathbf{v}_4$, that are known to intersect is:

$$\mathbf{p} \leftarrow (\mathbf{v}_1 \times \mathbf{v}_2) \times (\mathbf{v}_3 \times \mathbf{v}_4)$$
$$x \leftarrow p(1)/p(3), \quad y \leftarrow p(2)/p(3)$$

where $\mathbf{p}$ is a 3-vector, $\mathbf{v}_i$ are the end vertices expressed as homogeneous 3-vectors $(x_i, y_i, 1)$, and the operator $\times$ is the cross product.

Conclusions

A convention for interpreting general polygons has been proposed, together with algorithms for processing such polygons. This convention, as well as the algorithms for dealing with it, has been designed to fulfill the requirements found in processing the geometry of IC masks. A proposal for the adoption of this convention has been made in Chapter 7 of the new edition of *A Guide to LSI Implementation* (Hon and Sequin 1980). The convention, and algorithms presented here, were used for preparing the mask specifications for the recent MPC79 multi-university chip set. This article explains and, it is hoped, justifies the reasons for choosing this convention.

References

Hon, R. and Sequin, C.H. 1980. *A Guide to LSI Implementation*, Xerox PARC, Palo Alto California.

Mead, C.A. and Conway, L.A. 1980. *Introduction to VLSI Systems*, Boston: Addison-Wesley.

Sequin, C.H. and Newell, M.E., to be published. *"Cutting Corners when Constructing CIF Wires and Offset Boundaries around Polygons"*.

FIGURE 6. An example of polygon with seven swaths.