THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 51, No. 7, September, 1972
Printed in U.S.A.

Derivation of All Figures Formed by the Intersection of Generalized Polygons

By MICHAEL YAMIN

(Manuscript received March 7, 1972)

A computer program is described which generates every intersection figure resulting from the superposition of two closed polygon-like plane figures, each consisting of an arbitrary number of line segments or circular arc segments. Each intersection figure is assigned to one of four regions of the plane, representing the union, the intersection, and the two "exclusive-OR's" formed by the pair of input figures. The two input figures may intersect or be tangent at any number of points and may have sections of coincident boundaries. No grid approximation is used. The program operates in two stages: the first stage analytically finds and classifies every point of intersection or tangency of the figures; the second stage regards these points as the nodes of a graph and applies an algorithm which causes each intersection figure to be traced just once.

I. INTRODUCTION

In the course of a project related to computer-aided integrated circuit mask design, it became necessary to describe the configuration which is formed when two polygon-like plane figures are superimposed on one another. Two closed figures in a plane divide the plane into four regions: inside both figures, inside the first but not the second, inside the second but not the first, and inside neither. Each region may consist of one or more figures. Assuming that the original figures consist of an arbitrary number of sides, each of which may be a line or circular arc segment, it was desired to describe every figure resulting from their intersection and to assign each to one of the four regions.

The simple approach of establishing a grid of points and determining which sets of points are included in each region was not considered applicable, because the number of grid points required for sufficient resolution would have led to excessive computation. Instead, the following approach was used. Pairs of sides, one from each input figure,

were investigated analytically for points of intersection or tangency, or colinear sections. From the resulting information, a table of intersection points was developed. Included in this table was information concerning the type of the intersection: crossing, tangency, beginning or end of a colinear section resulting in a crossing, or beginning or end of a colinear tangency. With the intersection table available, the collection of line and arc segments constituting the superimposed figures could be considered as a graph, the intersections being the nodes. An algorithm for tracing the graph so as to generate every intersection figure once and only once was developed. The tracing paths thus determined were used to select appropriate analytical data from the original figure descriptions so as to generate descriptions of the new figures in the same numerical formats as the original figure descriptions. Classification of the figures was an automatic result of the tracing algorithm.

II. NUMERICAL REPRESENTATION OF FIGURES

Both input and output figures are described as sequences of sides in a numerical format designed for efficiency of computation rather than compactness of storage. The rotational tracing sense (clockwise or counterclockwise) of each figure is specified, and thus each side has a direction. A side, which may be a line or circular arc segment, is represented by eight sequential words in memory: the starting coordinate pair, parameters of the analytical equation, and the terminal coordinate pair. Since the terminal point of one side is the starting point of the next, each additional side requires six additional words of storage. Each figure is explicitly closed with a side, which terminates at the starting point of the first side.

The side sequence of each figure is preceded by a header containing information about the figure as a whole: its extreme x and y coordinates, allowing for projecting arcs; its area; the number of sides; and the sense (cw or ccw) of the entire figure. The header also contains pointers which can be used to associate the figure with externally tabulated information or can be used to associate it with other figures in one or more linked lists. Figure descriptions are stacked sequentially in memory; specific figures are located by pointers into this data structure. The exact arrangement of pointers and external tables depends on the application.

Two limitations are imposed on the input figures; they must nowhere intersect themselves; and no side should be colinear with an adjacent

side. It is possible to override the second limitation with a preprocessing program which combines such sides and condenses the figure description appropriately.

III. DEVELOPMENT OF INTERSECTION TABLE

Given two figures described in the above format, it is desired to list all their points of intersection. This list of intersections will include the coordinates of each intersection point, indices assigning it to a specific side of each of the two figures, and a code defining it as a crossing point, a tangency, one end of a colinear section topologically equivalent to a crossing, or one end of a colinear section topologically equivalent to a tangency.

It is typical of this entire analysis that much of the work is involved with the treatment of special cases. When the middle of one side is intersected by the middle of another, it is no trouble to make the appropriate entry in the intersection list. The problems arise when intersections involve more than one side of each of the input figures, as when they occur at corners, or when colinear sections "wrap around" several sides of each figure. To handle these cases, the intersection table must be developed in stages, entries ambiguous at a given stage being flagged as such and clarified in the next stage.

One of the input figures is arbitrarily declared the operating, or A, figure; the other the passive, or B figure. The elementary process of the intersection table development is the solution of the side equation of one side of A with one of B to find any intersections, tangencies, or colinearities within the bounds of both segments. A solution which is detected just at the end of one of the sides can not generally be classified as a crossing or tangency at this stage. The second stage is to repeat the process, with the same side of A, for every side of B, so that a table of points of intersection for this one line or arc segment with B is created. At this stage, while ambiguities resulting from intersection at a corner of B have generally been cleared up, solutions at the ends of the A test side are still not finally classified. The third and last stage is to repeat the above for every side of A, pairing up unclassified solutions which correspond to intersections at corners of the A figure.

Each of these stages of analysis is performed by a subroutine which loops on the previous one. Thus, a subroutine called INSECT returns the point or points of intersection of two line or arc segments (that is, those solutions of their equations which lie within the bounds of both segments) or reports their colinear coincidence. SLASH returns the

points of intersection of a line or arc segment with a closed figure consisting of such segments; classifying as crossings, tangencies, etc., those intersections which do not involve the ends of the segment and returning special codes to provide what information it can about the others. INTLST returns the points of intersection of two closed figures consisting of line and arc segments, each intersection being classified as a crossing, a tangency, or the start or end of a colinear tangency or crossing. This is the "intersection list" previously described; the intersections appear in the order in which they would be encountered when tracing the A figure. Two special situations are recognized: if the figures nowhere intersect, an intersection list of zero length is returned; if the figures are coincident (everywhere colinear), a special code is returned.

In principle, this procedure requires a number of calls to INSECT equal to the product of the number of sides of the two figures. However, each of the subroutines applies a quick preliminary test for overlap of the circumscribed rectangles of the input line segments or figures. In most cases, this avoids a great deal of detailed computation.

INTLST and SLASH use a complicated sieve of analytical tests to determine the nature of intersections which occur at figure corners and involve three, sometimes four, arc or line segments. Typically, these tests must decide whether the intersections represent a crossing or tangency of the two figures, and the geometrical question to be resolved is usually whether the direction of incidence of one line or arc segment on another segment, or on the point of intersection of two other segments, is from the left or from the right. In the case of lines, a vector cross product is used to make this determination; in the case of arcs, which may be tangent at a point of intersection, logic involving the direction of the arc center is required.

All three levels of geometric analysis make use of a quantity called the "error margin," which is necessary to allow for computational inaccuracies. The assumption is made that two points closer together than this small distance are in reality the same point. The error margin differs from a grid approximation in that its magnitude has no influence on computation time.

One more item of information must be added to the intersection list for the purposes of figure tracing. This is the incidence direction of the A figure on the B figure, and of B on A, at every intersection. The incidence is a two-valued parameter; one value corresponds to *entry* of one figure into the other at a crossing intersection or *inside incidence* at a tangency; the other value corresponds to *exit* at a crossing

or *outside incidence* at a tangency. There are obviously four types of intersection, corresponding to the combinations of the two possible incidence values of A-on-B with those of B-on-A. It is convenient to determine these values as part of the program next to be described, rather than as part of INTLST.

IV. GENERATION OF INTERSECTION FIGURES

The development of the intersection table described above has been a geometrical problem in that finding the coordinates and the nature of each intersection has required knowledge of analytical parameters describing the exact position of each geometrical entity in the plane. With the intersection table in hand, the problem of tracing each intersection figure becomes topological in nature. The overlapping polygons may be considered as a graph, that is, a set of intersection points, or nodes, between some of which exist connecting paths, or edges. For the purpose of figure-tracing, it is immaterial whether an edge between two intersections consists of one or many sides, or whether they are lines or arcs. It is enough to know to which figure the edge belongs.

An algorithm has been devised by which each circuit of the graph which corresponds to an intersection figure may be traced once and only once. The sequence of edges constituting such a circuit may then be used to select analytical information from the input figure descriptions, from which a description in similar format of the intersection figure represented by the circuit may be assembled. Since each edge contains no intersections, it must belong to a single figure, which is known. Its start and end points, and the figure sides upon which they lie, are associated with its terminating nodes as listed in the intersection table. The tracing direction of a given edge may be the same as or opposite to the direction of the original figure. The generation of the intersection figures then consists essentially of copying sequences of sides or partial sides, forward or backward, as guided by the edge data and the edge sequence in the specified circuit. The region of the plane to which the intersection figure belongs is determined from the incidence information.

The most direct way to implement such a procedure is based on the fact that each edge is part of two intersection figures. The method is to list, in cyclic order, all the edges radiating from each node. Then, starting from any node along an edge, proceed to the next node and transfer to the adjacent edge in cyclic order, clockwise or counter-

clockwise as appropriate, so that the trace remains within a "cell" of the graph. Continue until the starting node is reached, then repeat with a new starting node. The complete set of intersection figures will be traced when each edge has been traversed just twice (a colinear section being considered one edge). Unfortunately, no explicit numerical description of the edges connecting the nodes is available at this stage in the analysis; the necessary information must be computed from the input figure descriptions, the side blocks of which do not, in general, explicitly represent edges of the graph.

The algorithm actually used, therefore, centers its attention on the nodes (intersection table entries), rather than the edges. As the trace encounters each node, the algorithm specifies the next node which must be encountered to keep the trace within a "cell" of the graph; the numerical description of the connecting edge is only then generated and posted directly to the output figure description. In principle, each node must be encountered four times for completeness, once for each of the figures of which it is a corner. The implementation of this test for completeness is complicated by special considerations which arise at nodes which are tangencies or colinearities; while the verbal description of these special cases in Section 4.1 appears complex, their implementation in computer code requires only minor modifications in the program flow.

The circuit tracing and intersection figure generation operations are embodied in a single subroutine called OVRLAP. This subroutine calls INTLST to establish the intersection table, in which the intersections are listed in A-figure order. For efficiency, it is desirable to have a table of intersections in B-figure order. This could be created by another call to INTLST, but it is much more efficient to produce it by reordering a copy of the A-figure intersection table. The two tables are crossreferenced so that any entry in one can be located in the other. The OVRLAP subroutine also finds, and tabulates, the A-on-B and B-on-A incidence values at each intersection as described previously.

OVRLAP is a modular program, with modules (procedures) which perform elementary functions such as:

(*i*) Extract a side from the A or B figure.
(*ii*) Insert a side in an output figure description.
(*iii*) Replace a side description by a description of the same side, but backwards.
(*iv*) Given a side description and a point on the side, truncate the side to return the first or second segment.

Other modules use these to perform higher-level functions, such as:

(*i*) Insert in an output figure description a sequence of sides from one intersection to another, derived from the A or B figure as required, forward or backward as required.

(*ii*) Generate an output figure by tracing a complete circuit, according to the circuit-tracing algorithm, from a given intersection back to itself.

(*iii*) Copy an input figure bodily as an output figure, making appropriate changes in the figure header.

4.1 *Circuit Tracing Algorithm*

First, for simplicity, consider a pair of intersecting figures in which all the intersections are point crossings. Tangencies and colinearities introduce complications which require modified treatment.

(*i*) Starting at any intersection, trace two circuits. One is initiated by tracing forward on the B figure, the other by tracing backward on the B figure. Before starting these traces, compute a circuit incidence code for each. This code has four possible values, one for each of the possible combinations of A-on-B and B-on-A incidence values. For the B-forward trace, it combines the two incidence values (B-on-A and A-on-B) stored for the starting intersection; for the B-backward trace, it combines the A-on-B value with the opposite of the B-on-A value.

The procedure of "tracing" consists of going to the next intersection on the appropriate figure in the chosen direction. The "next intersection" is an adjacent item in the A-figure or B-figure intersection table, as appropriate. The last item on each list is considered "adjacent" to the first.

(*ii*) At the next intersection:

If tracing on B, switch over to A backwards.

If tracing on A, switch back to B in the starting direction.

Label every intersection encountered in this way with the circuit incidence code for the current trace. Each item block in the B-figure intersection table has space to store four such code labels.

Repeat step (*ii*) until the starting intersection is encountered, completing the circuit.

(*iii*) On completing both circuits from the current starting intersection, go to the next intersection on the A figure and proceed as under (*i*) above. However, do not perform any trace for which the starting intersection is found already to be labelled with its circuit incidence code.

(*iv*) Continue this procedure, making each intersection along the A figure the starting intersection in turn, until every intersection has been the starting intersection.

On termination of this procedure, every circuit on the graph which corresponds to an intersection figure will have been traced just once. Moreover, the circuit incidence code associated with each figure directly assigns the figure to one of the four regions into which the two intersecting figures divide the plane, thus:

Circuit Incidence Code Means	*Region of Plane*
A enters B, B enters A	Inside A, inside B (A and B)
A enters B, B leaves A	Outside A, inside B (B and not-A)
A leaves B, B enters A	Inside A, outside B (A and not-B)
A leaves B, B leaves A	Outside A, outside B (not-A and not-B) (A or B)

(The last category represents the logical union of the two figures. Such a figure may be a "hole" in the union, or the outer boundary of both figures. In the latter case, the inside of the figure is logically the complement of not-A and not-B, namely, A or B).

This algorithm follows from the fact that each crossing intersection forms a corner of four figures, each in a different region of the plane as described above. Step (*i*) "stakes a claim" on two of these figures. (The other two will be or have been claimed by the previous A intersection.) Step (*ii*) assures that the trace stays within the "claimed" figure. Step (*iii*) moves the procedure forward, assuring that no figure previously "claimed" is retraced. Since every intersection figure is "claimed" by at least one intersection, step (*iv*) assures that every figure is eventually described.

A point of tangency between two figures differs from a crossing intersection in that two of the four figures which impinge on it belong to the same region of the plane. The following special considerations apply to tangencies:

(*i*) On encountering a tangency when tracing on the A figure, switch over to the B figure only if the B-on-A incidence value at the tangency is the same as at the starting intersection for a B-forward trace or opposite for a B-reverse trace. When tracing on the B figure, switch to the A figure only if the A-on-B incidence is the same as at the starting intersection. If this condition is not met, go on to the next intersection.

(*ii*) When switchover from B to A is made at a tangency, do not label the tangency with the circuit incidence code.

(*iii*) When it is the turn of a tangency to be the starting intersection, only one tracing circuit, at most, rather than two, is originated. The permissible tracing direction of B is determined by the following "inside-outside" rule: If the rotation senses of the two input figures are the same (both cw or both ccw), trace backward on B if the B on A and A on B incidence values at the tangency are the same, and forward on B if they are opposite. If the rotation senses are opposite, the rule is reversed. Of course, just as with crossing intersections, no trace is performed if the tangency is already labelled with the appropriate starting incidence code.

Either a tangency or a crossing of the two input figures may occur as a colinear section. Such a colinear section appears in the intersection tables as two adjacent intersections, one labelled as the "start" and the other as the "end" of the colinearity. "Start" and "end" refer to the direction sense of the A figure. The following special considerations apply to colinearities.

(*i*) On encountering either end of a colinearity while tracing a circuit, take the appropriate switchover action immediately, following the rules described above for point crossings or point tangencies as appropriate.

(*ii*) Never begin tracing a circuit at the "start" of a colinearity; go to the "end" of the colinear section and make that the starting intersection as described above.

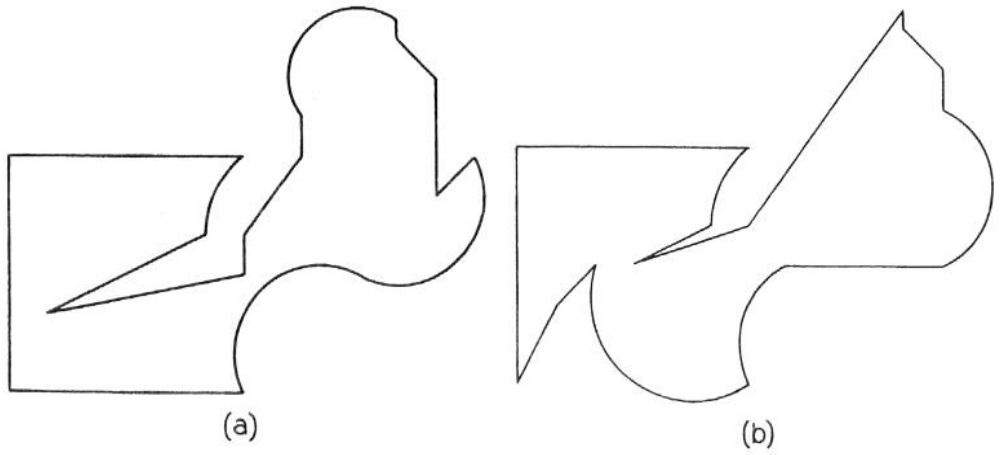


Fig. 1—Two input figures: (a) composed of 16 sides, 12 of them lines and 4 arc segments; (b) composed of 15 sides, 11 of them lines and 4 arcs.

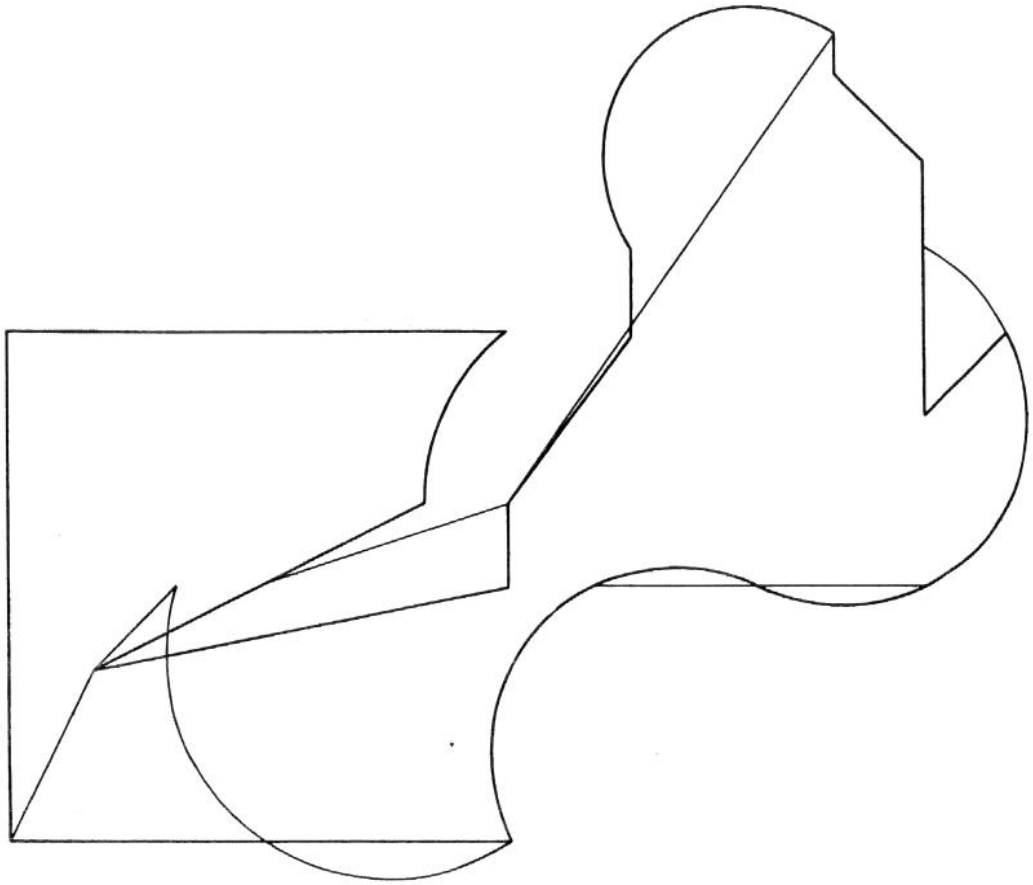
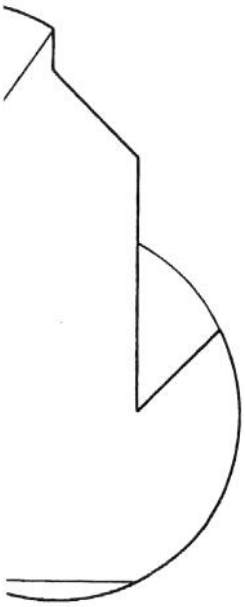


Fig. 2—Graph resulting from superposition of Figs. 1a and 1b.

(*iii*) If a colinear region is being traced, having been entered at the "end" point (whether by switchover or by initiation of circuit tracing), the next intersection encountered will be the "start" point. Do not switch figures at this point; just go on to the next intersection. Any switchover required will already have been done at the "end" intersection.

4.2 *Generation of Output Figure Description*

The algorithm just described sets out a procedure for stepping from node to node in the intersection lists in such a way as to describe certain unique circuits. In essence, it tells one to choose an appropriate item in the A-figure list; find the corresponding entry in the B-figure list; follow the B list item by item, in a specified direction, until certain conditions are met; switch to the corresponding entry in the A-figure list; follow this list backward, item by item, until certain conditions are met; switch to the corresponding entry in the B-figure list; and



continue switching back and forth in this way until the starting point is attained.

The numerical description of the output figure which is the objective of this program is generated simultaneously with the execution of this algorithm. As was mentioned earlier, the program contains a module which, given two specified intersection nodes, will generate a sequence of side descriptions representing a path from the first intersection to the second, on the A or B figure as specified, and forward or backward as specified. This module is invoked every time a switch from one intersection list to the other is made. The two intersections specified are that at which the list was entered, and that from which the switch is being made (if there are no tangencies or colinearities, these are adjacent intersections). The figure is that represented by the intersection list, and the direction is that in which the list was followed. The resulting side sequences are stacked one after another in memory,

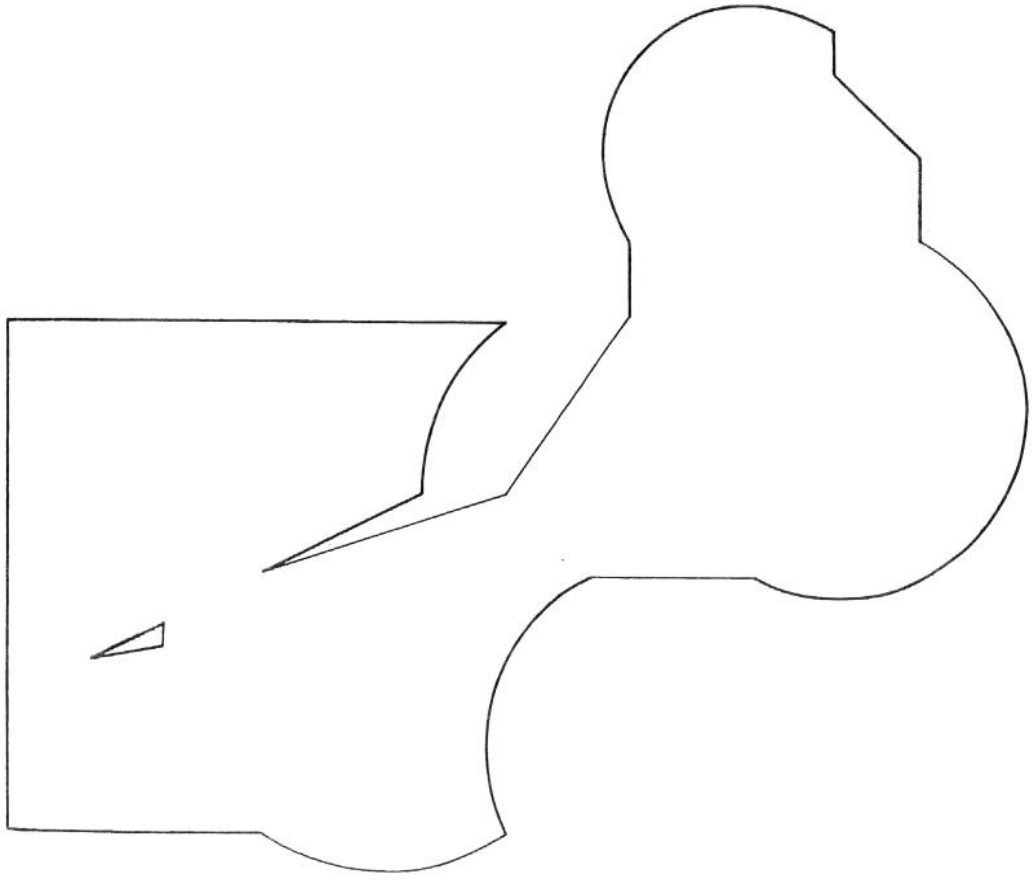


Fig. 3—Figures classified as "A or B"–the union of two input figures.

and 1b.

been entered at
ation of circuit
e "start" point.
ext intersection.
e at the "end"

r stepping from
describe certain
ppropriate item
ne B-figure list;
n, until certain
in the A-figure
tain conditions
figure list; and

and when the circuit trace is complete, so is the intersection figure description.

An anomaly can occur as a result of the circuit-tracing algorithm: two successive sides of an intersection figure may be returned which are colinear extensions of one another. It is easy to recognize this situation, and the program module which transfers individual sides to the output figure simply combines any two such sides.

Before a circuit trace is initiated, the intersection figure description must be initialized by allocation of space for its header. After the circuit trace is complete, certain header items must be filled in, particularly extreme coordinate values and the area. The area of a figure is found by a sum-of-trapezoids technique, with corrections for arcs. The area returns as a signed number, the sign denoting the rotational sense (cw or ccw) of the figure. The header includes a code which specifies to which of the four regions of the plane the figure belongs.

It can occur that two input figures have no crossings at all, but only tangencies. One may be inside the other, or they may be external to each other. In this case, the input figures are themselves intersection figures, but they will not be traced by the above algorithm. Therefore, they must be copied bodily as output figures and assigned by simple logic to an appropriate region of the plane. The same is true for figures which do not touch each other at all.

Another problem arises when two figures have one point tangency as their sole intersection. A figure will be generated which is wasp-waisted or self-tangent, either of which violates the rule against self-intersection. The program returns these descriptions regardless, and the calling program which uses OVRLAP must watch out for this situation. Finally, when one figure lies within another without any contact, one region of the plane (inside one figure but outside the other) cannot be described as a simply connected figure, though its area is easily calculated.

4.3 *Organization of Output Data Structure*

Intersection figures generated as described in the previous section are stored sequentially in a region of memory allocated for output. To be useful, this data structure must be organized and indexed in some way. The following system has been used:

(*i*) All the figures of a given type (that is, belonging to the same one of the four regions of the plane) are chained together in a linked list by a pointer in the figure headers.

(*ii*) A summary table is provided. This table has four blocks, each corresponding to one of the four regions of the plane. Each block contains: the number of figures of this type, the total area of these figures, and a pointer to the first figure in the corresponding linked list.

(*iii*) Each figure header has a pointer to the appropriate block of the summary table, and one to the next sequential figure description in the data structure.

V. DETAILS OF IMPLEMENTATION

The program described in this paper has been coded in FORTRAN IV and compiled and executed on the General Electric 635 computer. It has been coded as a subroutine, called OVRLAP, with three arguments: The locations of the two input figure descriptions and the starting location of a region of storage available for output. In addition,

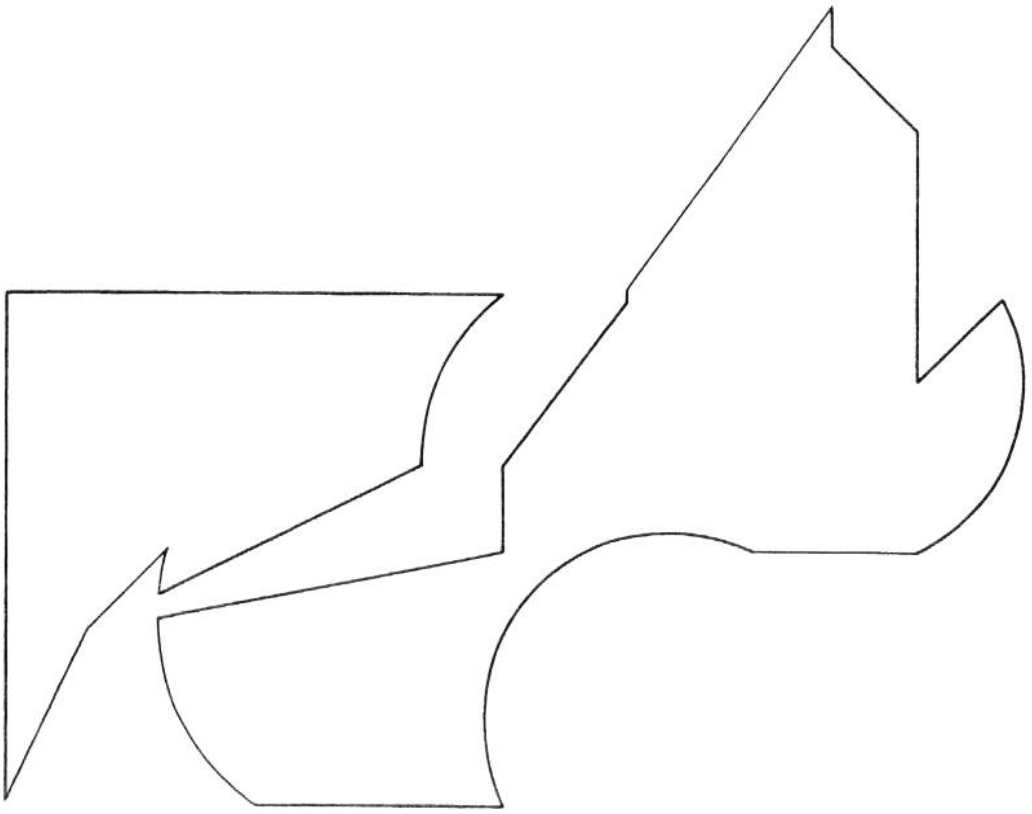


Fig. 4—Two regions of overlap, "A and B."

common blocks must be provided for the summary table described above, for a summary table (of different form) which controls the data structure in which the input figures are located, and for the error margin quantity discussed in Section III. The common block containing the summary table has three additional locations. One reports back the total number of figures generated; another carries a code for the mode of intersection (no contact, tangencies only, coincident figures, or normal intersection). The third is set on input as the total number of locations available for output figure descriptions and returns as a pointer to the first unfilled location. An error return occurs if the output data structure tries to overflow the space allocated for it.

Figures 1 through 6 are examples of the operation of this program. Figure 1 shows the two input figures. The first has 16 sides, 12 of them lines and the other 4 arc segments; the second has 15 sides, 11 of them lines and the other 4 arcs. Figure 2 shows the graph resulting from their superposition. There are 15 points of intersection, and every

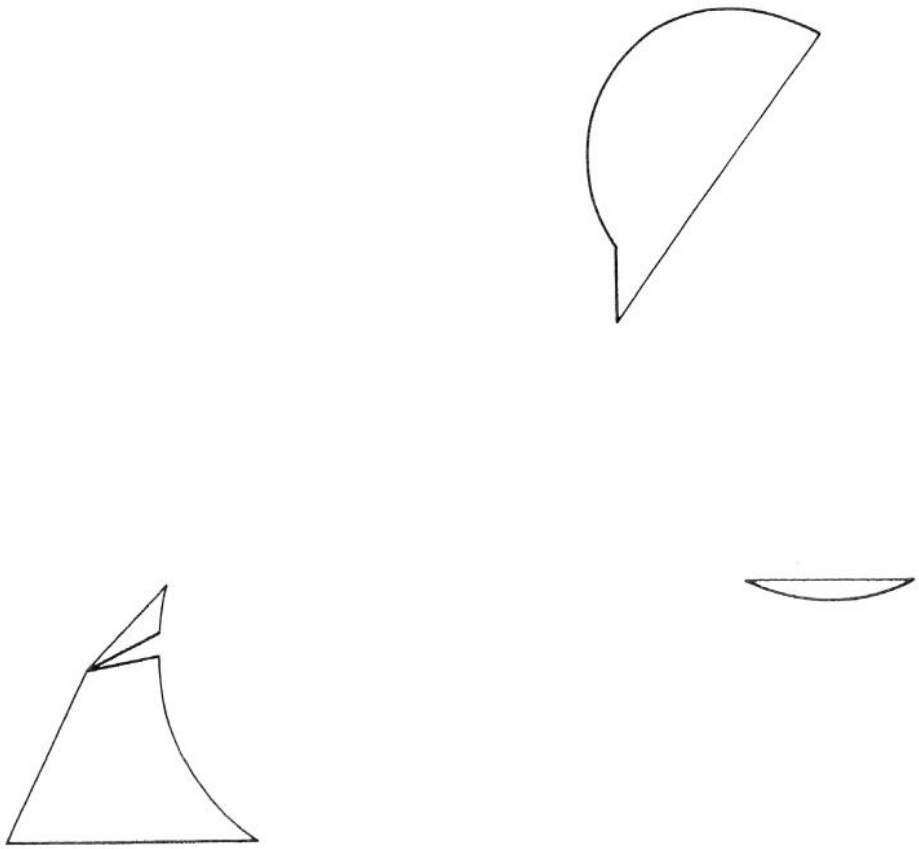


Fig. 5—Four regions of A not covered by B—A and not-B.

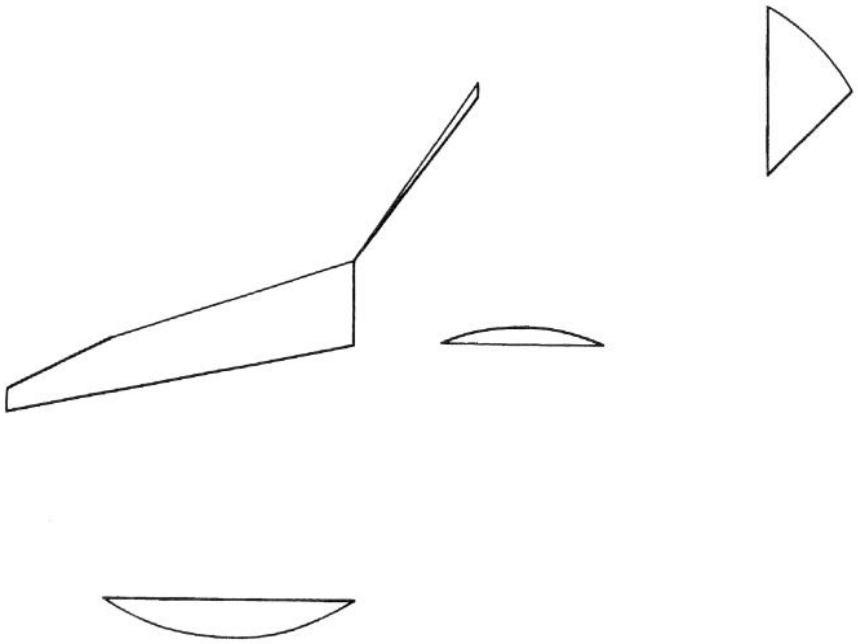


Fig. 6—Five regions of B not covered by A—B and not-A.

type is represented: point crossings, point tangencies, colinear crossings, and a colinear tangency. Thirteen intersection figures, including the outer boundary of the pair, are generated. Figure 3 shows the figures classified as "A or B" (the union of the two input figures). Note that there is an inner figure, or hole, which can also be described as bounding a region "not-A and not-B". Figure 4 shows two regions of overlap, "A and B" (the intersection, in the logical sense, of the two figures). Figure 5 shows four regions of A not covered by B (A and not-B) and Fig. 6 shows five regions of B not covered by A (B and not-A).

Execution of this analysis by the OVRLAP subroutine required 0.75 second of processor time on the GE 635, the memory cycle of which is one microsecond. This does not include time spent on physical input, generation of the input data structure, or physical output.

OVRLAP and all its required subroutines and common blocks, exclusive of system library subroutines, occupies about 11000 (decimal) words of storage.

VI. SUMMARY

A computer program has been developed which, given two polygon-like figures each consisting of an arbitrary number of line segments or circular arc segments, generates every intersection figure resulting

from their superposition and identifies its logical relationship with the input figures. The two input figures may intersect or be tangent at any number of points and may have sections of coincident boundaries. Each input figure is described as a sequence of bounded analytical expressions and the output figures are generated in the same numerical format. No grid approximation is used; the program operates by analytically finding every point of intersection or tangency of the figures; these points are the nodes of a graph through which the program traces according to an algorithm which causes every intersection figure be traced just once. A FORTRAN IV implementation of this program took 0.75 second on a GE 635 computer to resolve the superposition of two figures, of 15 and 16 sides respectively, into 13 resultant figures.