Toward Reliable Polygon Set Operations

Mark Friedell and Sandeep Kochhar

Aiken Computation Laboratory
Harvard University
Cambridge, Massachusetts 02138

Abstract

Polygon intersection (clipping) and difference are among the most fundamental operations in computer graphics. To the uninitiated, these problems appear trivial; in fact, they are extremely difficult to perform reliably by computer. Although the graphics literature already provides algorithms for polygon set operations, they have two significant weaknesses: (1) they may fail because they are specified ambiguously for some configurations of subject polygons, and (2) small arithmetic errors, an unavoidable artifact of floating-point calculations, can cause significant aberrations in the result.

This paper presents and provides the rationale for a new polygon-intersection algorithm whose input polygons may have holes and, recursively, fillers within holes, holes within fillers within holes, etc. The logical underpinnings of the algorithm enable it to tolerate arithmetic errors due to finite-precision floating-point arithmetic. Polygon set difference is shown to be a small modification of the intersection algorithm.

Researchers in computational geometry consider these and similar problems with the intention of formulating algorithms whose correctness can be assured through formal proofs. The goal of this paper, however, is practical guidance for implementation based on accessible, intuitive arguments.

1. Introduction

This paper revisits the problem of polygon intersection, one of the most fundamental operations in computer graphics. The intersection of polygons A and B is the polygonal boundary of the region(s) common to both A and B. Polygon difference, $A-B$, is the polygonal boundary of the region(s) of A not also contained in B. To the uninitiated, these problems appear trivial; in fact, they are very difficult to solve reliably by computer. When tested carefully, most polygon intersection routines based on the right-turn rule fail almost immediately.

Researchers in computational geometry are working to develop algorithms for these set operations whose correctness is assured through formal proofs, e.g., [HOFFMAN & HOPCROFT], [HOPCROFT & KAHN], [MILENKOVIC A], [MILENKOVIC B], [SEGAL & SEQUIN]. The goal of this paper, however, is practical guidance for implementing reliable polygon set operations based on accessible, intuitive arguments.

In practice, polygon intersection is usually computed with some variant of the "right-turn" rule (e.g., [WEILER & ATHERTON]). In essence, this rule states that the intersection of two polygons is computed by tracing the contours of the polygons in clockwise fashion, switching from one polygon to the other where the contours intersect. This process begins by tracing one contour starting at an intersection where the contour crosses into the interior of the other polygon, as shown in Figure 1.1.

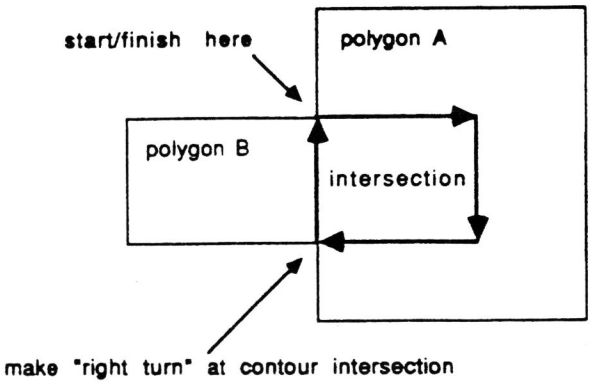


Figure 1.1 Intersection via the Right Turn Rule

There are two problems with the right-turn rule. First,

the rule is frequently ambiguous when a vertex of one polygon lies on the contour of the other, as shown in Figure 1.2. Although an intersector may be patched to process these configurations as special cases, confidence in the underlying rationale for the intersection procedure -- turn right at an intersection -- diminishes as each new problem configuration is discovered and another patch is added to the algorithm.

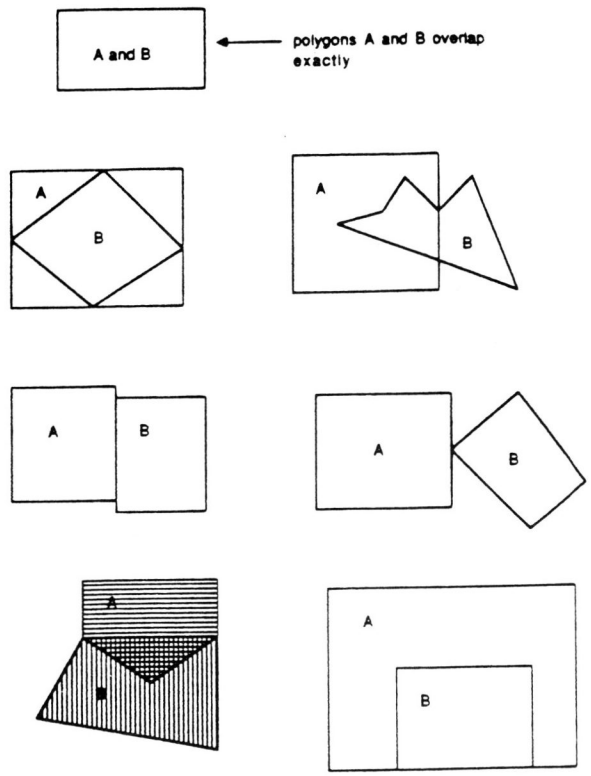


Figure 1.2 Difficult Configurations for the Right-Turn Rule

The second and more insidious problem with the right-turn rule is that it depends on perfectly correct detection of contour intersections, which cannot be guaranteed with floating-point arithmetic. The difficulty is that some contour intersections may not be detected, while some false intersections may be "found." The fallibility of contour-intersection calculations is analogous to the fallibility of the human visual system; as relationships between contours become more difficult to see, the risk of error increases. For example, the intersection of two nearly perpendicular contour edges near their midpoints, shown in Figure 1.3, is more likely to be detected correctly that the intersection (or lack of intersection) between the two edges in Figure 1.4. To overcome this problem, some implementations attempt to identify geometric configurations that may lead to computational inaccuracies and distort the subject polygons as needed to avoid the difficulties. Unfortunately, a distortion intended to avoid one inaccuracy may lead to the potential for another, requiring yet further distortion. Ultimately, the effects of the distortions may become visually perceptible.

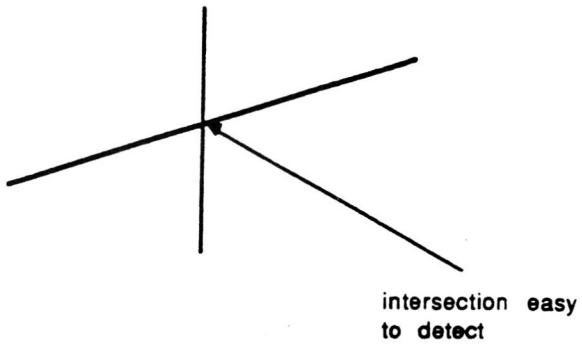


Figure 1.3 An Obvious Intersection

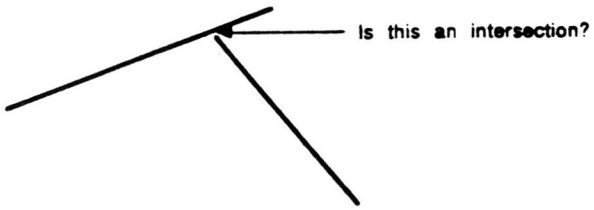


Figure 1.4 A Difficult Intersection to Detect

Section 2 of this paper describes a new polygon-intersection algorithm which accommodates directly the inaccuracies inherent in finite-precision floating-point arithmetic. The presentation is informal, in an effort to be readily accessible, but it is sufficiently detailed to allow implementation directly from this paper.

Making the algorithm immune to floating-point errors is the most difficult challenge. The approach here is to reason directly about the possible effects of floating-point errors, accept that no algorithm can be perfect in the presence of such errors, and develop an algorithm whose output error will be bounded in a satisfactory way. Specifically the error will be sufficiently small that its effects on a rendered image will be visually imperceptible.

Section 3 presents a small modification of the polygon-intersection algorithm which yields an algorithm for polygon difference.

2. Reliable Polygon Intersection

Polygon intersection is constructed from three basic geometric procedures: *Inside*, *Intersect*, and *Surround*. These procedures are unusual in that floating-point error affects their output in a predictable way. In the descriptions below, A and B are polygons, E_1 and E_2 are polygon edges, ε and $\varepsilon_{Surround}$ are error quantities, P is a point, and τ is a tolerance. *Inside* (P, A, τ) is a predicate that determines if P lies within A. More exactly, if *Inside* (P, A, τ) returns **true**, then P lies within A and is more than distance τ from the border of A; if the procedure returns **false**, P lies more than distance τ outside of A. *Inside* (P, A, τ) may also return **close**, which implies that P is less than distance $\tau+\varepsilon$ from the contour of A. Note that *Inside* is allowed to return either **true** or **close** for any point inside A that is more than distance τ but less than distance $\tau+\varepsilon$ from the contour of A. The procedure is allowed to return either **false** or **close** for any point more than distance τ but less than distance $\tau+\varepsilon$ beyond A. The behavior of *Inside* (P, A, τ) is illustrated in Figure 2.1.

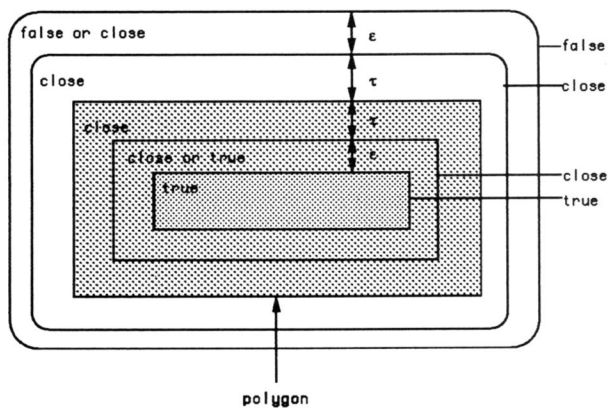


Figure 2.1 The *Inside* Predicate

Intersect (E_1, E_2) determines the point of intersection of two polygon edges. (Edge E extends from vertex E^s to vertex E^t.) The procedure returns either **false** or a parameter α in the range 0 to 1. If *Intersect* (E_1, E_2) returns **false**, then E_1 and E_2 do not intersect. If the procedure returns a value for α, then E_2 comes within distance ε of the point $(1-\alpha)E_1^s+\alpha E_1^t$. Note that in some circumstances, the returned values of **false** or a parameter value are both correct. Figure 2.2 illustrates the behavior of *Intersect*.

Surround (A, B, τ) determines whether polygon A surrounds polygon B. If *Surround* (A, B, τ) returns **true**, then all points in B are less than distance $\tau+\varepsilon_{Surround}$ beyond the boundary of A. If the predicate returns **false**, then some point of B lies more than distance τ beyond A. The behavior of *Surround* (A, B, τ) is illustrated in Figure 2.3. As this figure shows, both **true** and **false** maybe returned correctly by *Surround* for certain arguments. Appendix A outlines how to implement *Surround* in terms of *Inside* and *Intersect*.

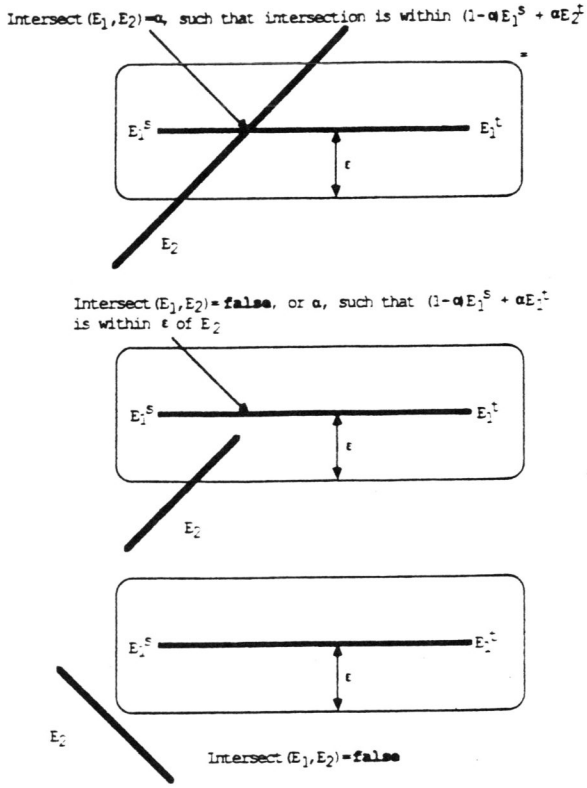


Figure 2.2 The *Intersect* Predicate

In the intersection algorithms presented below, ε and $\varepsilon_{Surround}$ are very small actual distances that cannot be perceived in a rendered image. A value for ε is determined by assuming that all data to be processed are transformed into some normalized coordinate system, say (-1,-1) to (+1,+1), and then taking ε to be the smallest value which can be assured by the implementations of *Intersect* and *Inside*. Ideally, an exhaustive analysis of the implementations of *Intersect*, *Inside* and *Surround* would be performed to calculate the correct value for ε. In most circumstances, however, this is not practicable. The pragmatic approach is to perform a conservative, approximate analysis and then increase the estimate of ε by a decimal order of magnitude. Happily, this approach yields values for the error bounds -- typically 0.001 in a normalized coordinate system ranging from (-1,-1) to (+1,+1) – that are comfortably under the threshold of perceptibility for high-resolution graphics. Note that $\varepsilon_{Surround}$ is a similar error bound for *Surround*, and it depends on ε as described in Appendix A.

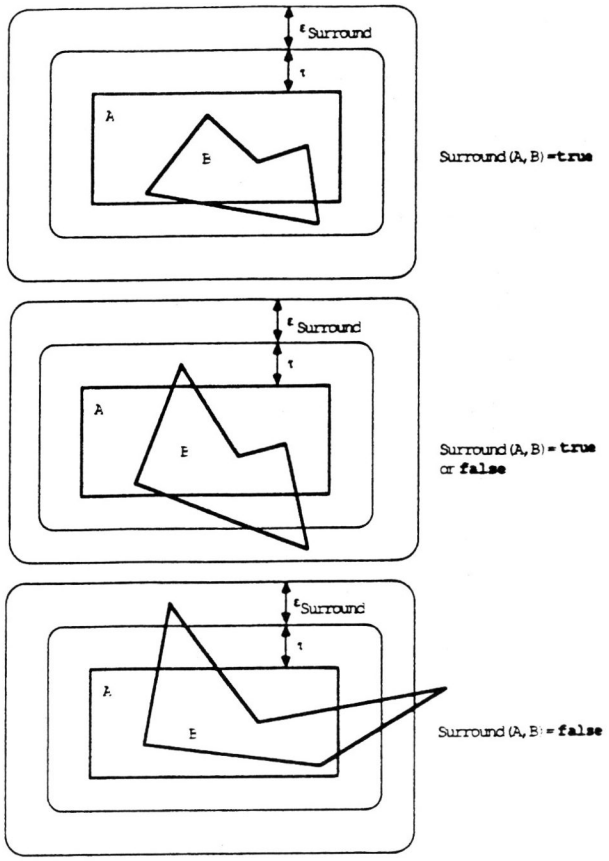


Figure 2.3 The *Surround* Predicate

2.1 Polygon-Edge Intersection

As a preliminary step to polygon intersection, consider the problem of finding the intersection of a polygon and an edge. In the general case of a polygon with concave vertices, the intersection will comprise 0 or more segments of the edge, as illustrated in Figure 2.4. Each endpoint of these segments is either an endpoint of the edge or a point of intersection between the edge and the contour of the polygon.

The first step toward computing the required intersecting segments is to compile a superset of their endpoints. The *Inside* procedure, with τ=0, is used to test the edge endpoints for inclusion in the polygon. If *Inside* reports **true** or **close** for either edge endpoint, it is included in the superset. Next, *Intersect* is used to compare the subject edge with each contour edge of the polygon and any intersection points are added to the superset. The problem now is to identify, in the superset, pairs of endpoints that describe the intersecting segments.

To find the intersecting segments, the superset of endpoints is sorted along the original subject edge. This is done by using the parameter returned by *Intersect* as a sort key. (As a result of imperfect arithmetic, the sorted list may not be in strictly correct order; however, any errors will be imperceptible and will not affect the behavior of any algorithms developed below.) Next, beginning at one end of the sorted list, each sequential pair of endpoints is examined (see Figure 2.5). To determine if a segment lies within the polygon, hence is part of the polygon-edge intersection, *Inside* is used, with $\tau=2\varepsilon+\mu$, to examine the midpoint between the endpoints. (Here $\tau=2\varepsilon+\mu$, because the locations of the endpoints are known only to within ε, the process of computing the midpoint may introduce further error, taken to be very much less than ε, and evaluating the expression for τ may introduce a very small error, bounded by μ.) If *Inside* returns **true**, the segment is part of the intersection. If *Inside* returns **false**, then the segment is not part of the intersection.

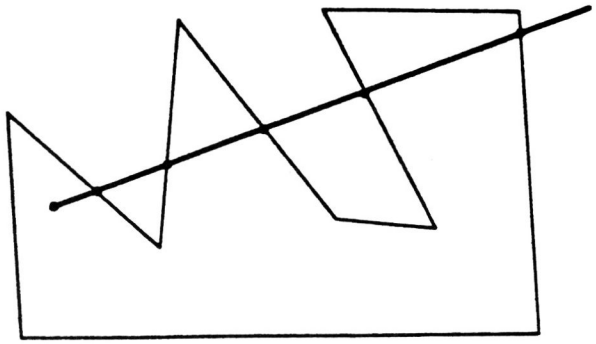


Figure 2.4 Intersection of Polygon and an Edge

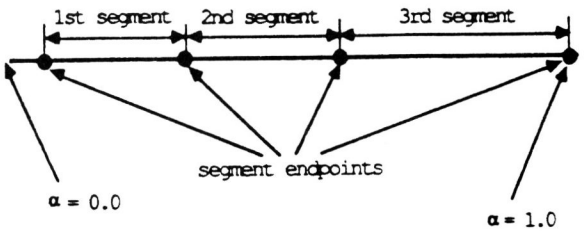


Figure 2.5 Contiguous Segments Along an Edge

Of course, *Inside* might return **close**, and the status of the span would be undecided. If this happens, one or more additional locations between the endpoints must be tested until *Inside* provides an unambiguous result or it is determined that the segment is everywhere very close to the contour of the polygon. The latter happens when *Inside* returns **close** for all locations sampled at intervals of length $\delta_{polygon-edge}$ between the endpoints. In case the segment is everywhere very close to the polygon contour, it is included as part of the intersection. The rationale for this conclusion is that by making $\delta_{polygon-edge}$ sufficiently small, no part of the segment will be perceptibly beyond the contour of the polygon.

This algorithm for polygon-edge intersection is part of the polygon-intersection algorithm presented below, in Section 2.2. In that algorithm, polygon-edge intersection may be performed less exactly: only a superset of the intersecting segments is required. Therefore, any candidate segment whose midpoint is either inside or close to the subject polygon is treated as part of the intersecting-segment superset -- there is no need to test multiple points along any candidate. While there is a performance penalty associated with extraneous intersecting segments in the superset, the cost of eliminating them by testing multiple sample points may be prohibitively high.

2.2 Ordinary Polygon Intersection

Consider the intersection of two ordinary polygons, i.e., polygons without holes. The contour(s) of the intersection region(s) can be composed from components of the contours of the two polygons, as shown in Figure 2.6. For any two polygons, A and B, the required components of the contours can be constructed from the segments of the edges of A that intersect B and the segments of the edges of B that intersect A. The polygon-edge intersection technique from Section 2.1 can be used to find a superset of these segments.

To construct the contour(s) of the intersection of A and B, the superset of intersecting edge segments is treated as the edges of a specialized geometric multigraph, referred to as an *edge-segment graph* (see Figure 2.7). Unlike an ordinary geometric graph, two edges in an edge-segment graph, X and Y, are connected if one endpoint of X is within distance $2\varepsilon+\mu$ of an endpoint of Y. This aggressive connection rule is needed because, as a result of floating-point error, the calculations of the endpoints of the intersecting edge segments are guaranteed only to be within ε of their true location, and the process of measuring the distance between endpoints may be in error by less than μ. Note that in an edge-segment graph, edges X and Z might not be connected, even if X is connected to edge Y and Y is connected to Z, as shown in Figure 2.8.

By interpreting each non-crossing edge cycle of length greater than 2 in the edge-segment graph as a polygonal contour, a superset of the contours required to describe the intersection of polygons A and B is found. This superset may contain, in addition to the required contours, some incorrect contours that do not lie within the intersection of A and B as well as some contours that are redundant (see Figure 2.7). Any incorrect and redundant contours must be identified and eliminated.

To filter out incorrect contours, each constructed contour is compared to both A and B, using the *Surround* predicate with $\tau=\varepsilon$. Here $\tau=\varepsilon$, since the vertices of the constructed contours are known only to within ε of their correct positions. Only those constructed contours that are surrounded by both subject polygons are in the intersection. Redundancies are eliminated by comparing each pair of intersection contours with $\tau=2\varepsilon+\mu$. Any contour that is surrounded by another contour may be eliminated. When eliminating redundancies, $\tau=2\varepsilon+\mu$ since corresponding vertices of "identical" regions may deviate by ε in opposite directions from their true location.

2.2.1 Rationale

The logical basis for this intersection algorithm can be established by working backwards from the desired result. Assume a process that generates a superset of the contours of the intersection of polygons A and B. To identify only the contours that lie within the intersection, compare each generated contour to both A and B. If a generated contour lies within both polygons, it lies within their intersection, but some of these contours might be redundant, i.e., they might lie within other generated contours that also lie within the intersection. To remove the redundant contours, compare pairs of intersection contours and remove any contour that lies within another contour.

The required superset of intersection contours can be generated from a collection of edges containing at least each segment of each edge of polygon A that intersects polygon B and each segment of each edge of polygon B that intersects polygon A. If all elements of this collection are combined in all possible ways to create polygonal contours, the contours of the intersection of A and B certainly will be generated. In addition, some contours that do not lie in the intersection, and some some that are in the intersection but are redundant, may be generated as well.

The set of edges required to create the intersection contours can be generated with the polygon-edge intersection algorithm described in Section 2.1 to find a superset of the intersections of all edges of polygon A with polygon B and the intersections of all edges of polygon B with polygon A.

2.3 Polygon-Polygon Intersection with Holes

Consider now the more complicated case in which two polygons, A and B, have holes. A polygon's outer contour will be referred to as a + contour and a hole will be referred to as a – contour.

The intersection technique for polygons with holes is similar to that for ordinary polygons. One difference is how the edges in the edge-segment graph are generated. For polygons with holes, the intersection of every hole edge, as well as every outer-contour edge, of polygon A with polygon B, and vice versa, is needed.

The polygon-edge intersection procedure with holes is also slightly different from that presented in Section 2.1. When computing the superset of edge-segment endpoints, *Intersect* is used to find the intersection of the subject edge

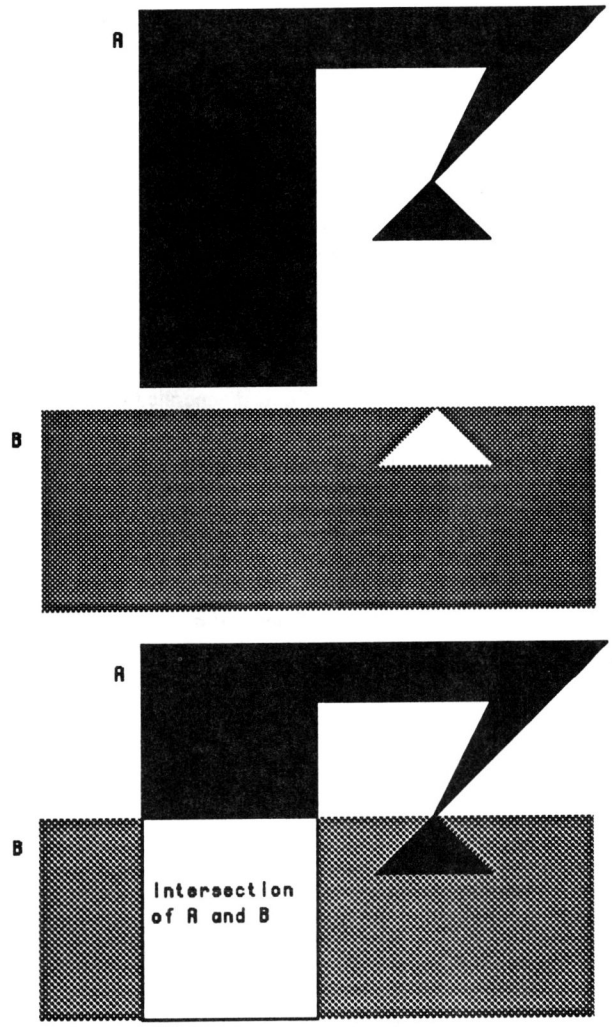


Figure 2.6 Intersection of two Ordinary Polygons

with every hole edge, as well as with every outer-contour edge, of the subject polygon. The *Inside* procedure, which is used to determine whether an edge endpoint or segment lies within a subject polygon, is applied in the ordinary way, i.e., it considers only the outer contour of the subject polygon. Figure 2.9 shows the intersection of two polygons with holes, complete with intersecting edge segments and edge-segment endpoints.

As in the simpler case for ordinary polygons described in Section 2.2, the contour-construction technique applied to the edge-segment graph will certainly produce all the contours, both + and −, required to describe the intersection region(s), but it may also produce some incorrect or redundant contours. As before, the correctness of constructed contours needs to be verified and any redundant contours need to be removed. In addition, the sign (+ or −) of a constructed contour must be determined.

Only constructed contours that lie within the outer contours of both subject polygons are correct and considered further. If a correct contour is surrounded by any hole of either subject polygon, it is a − contour; otherwise, it is a + contour. Among the correct contours, any + contour that lies within another + contour or any − contour that lies within

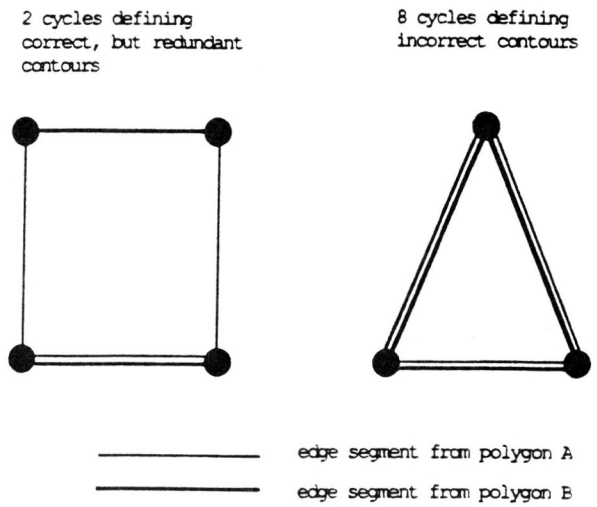


Figure 2.7 Edge-Segment Graph for Figure 2.6

another − contour is redundant and should be removed.

2.3.1 Rationale

The logical basis for this algorithm, beyond that presented in Section 2.2.1., relies on semantics of + and − as used to characterize contours. For polygon X, each point within its + contour is within polygon X unless the point also lies within a − contour of X, while each point within a − contour of X is unequivocally not within polygon X. The − contour is the more definitive characterization.

Any contour, + or −, used to describe the intersection of A and B must necessarily be surrounded by the + contours of A and B. Hence, the algorithm discards as incorrect any generated contours that do not meet this requirement.

Each correct contour is characterized as + or − with respect to both subject polygons. To characterize a contour, C, with respect to a subject polygon, X, the relationship of C to every contour of X is examined. If C is surrounded by a − contour, it is a − contour with respect to X, otherwise, it is a + contour with respect to X. Recall the semantics of + and − and consider what + or − with respect to X means. If C is + with respect to X, each point in C is within polygon X unless it also lies within some − contour of X. If C is − with respect to X, each point in C is absolutely not within X.

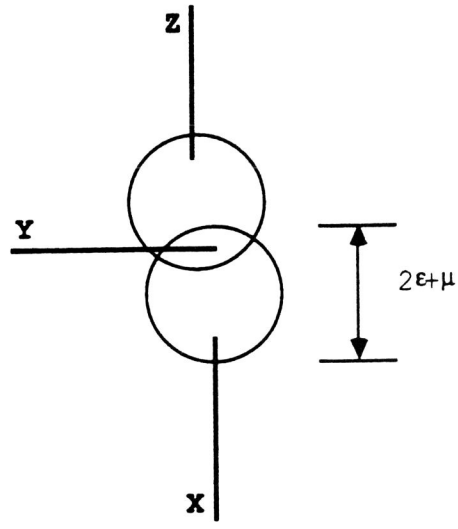


Figure 2.8 Connection of Edges in Edge-Segment Graph

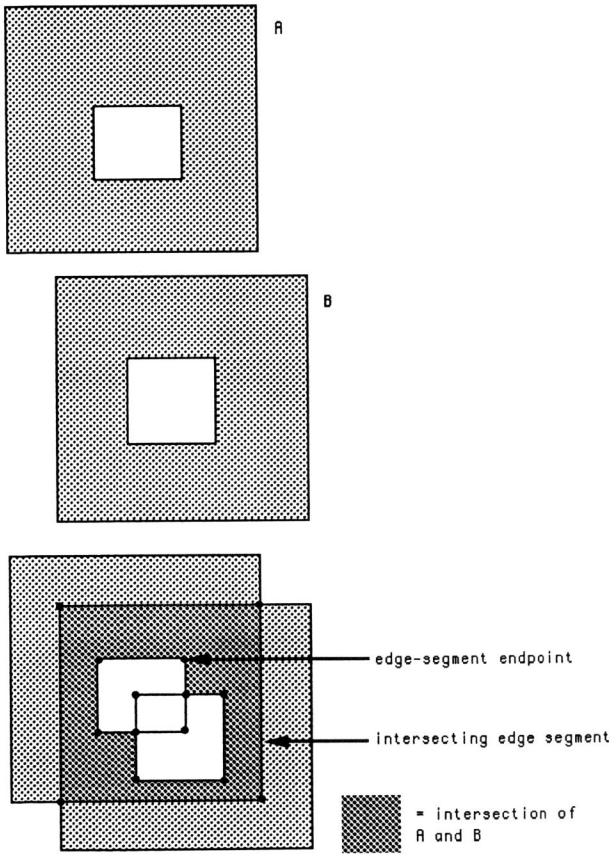


Figure 2.9 Intersection of two Polygons with Holes

If C is – with respect to either subject polygon then C is – with respect to the intersection of the subject polygons, i.e., each point in C is not within at least one of the subject polygons, hence not within their intersection. If C is + with respect to both subject polygons, it is + with respect to their intersection, i.e., every point in C is within the intersection of the subject polygons unless it also lies within some – contour of the intersection.

Now consider more carefully a + contour, C_{I+}, of an intersection. If P is a point within C_{I+} that also lies within some – contour, C_{X-}, of a subject polygon, X, then the intersection of the subject polygons is required to also have a – contour containing P. What guarantee is there that this – contour will be present in the intersection? Since P lies within both subject polygons and P is contained in C_{X-}, the contour-generation process will generate a fragment of C_{X-}, C_{I-}, that contains P. Since C_{X-} surrounds C_{I-}, C_{I-} will be classified – with respect to X, and hence – with respect to the intersection.

2.4 Intersection with Holes and Fillers

The final treatment of polygon-polygon intersection concerns polygons that may have holes and, recursively, fillers within holes, holes within fillers, etc. Such polygons can be described with a tree, the root of which is the polygon's outer contour. The contours of the holes and fillers are arranged at alternating levels in the tree, as shown in Figure 2.10.

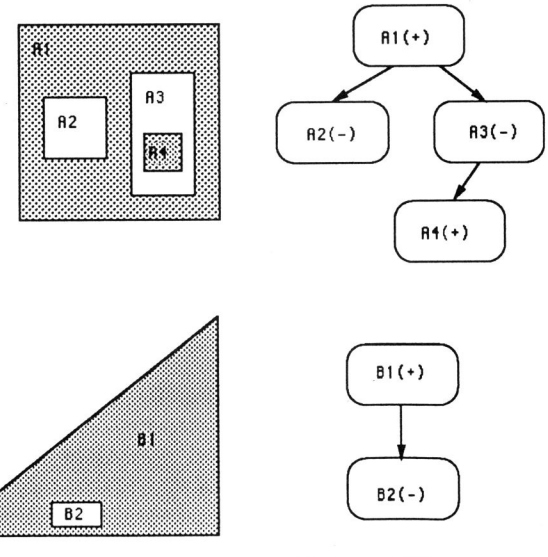
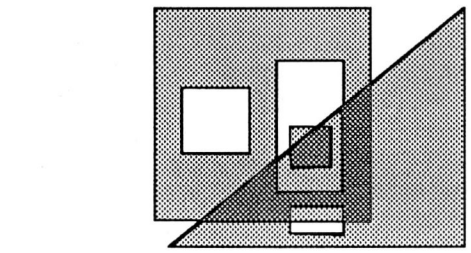


Figure 2.10 Polygons with Holes and Fillers

The intersection procedure is an extension of that described in Section 2.3 for polygons with holes only. Once again, the contours generated from the edge-segment graph may include, in addition to the needed contours, some contours that are incorrect or redundant. Further, the sign of each contour must be decided.

For a constructed contour to be correct, it must lie within the outer contours of both subject polygons. The sign of a constructed contour is decided by finding the smallest contours (outer, hole, or filler) of both subject polygons in which the constructed contour lies. If both surrounding contours are +, the constructed contour is a + contour; otherwise (at least one surrounding contour is a − contour), the constructed contour is a − contour.



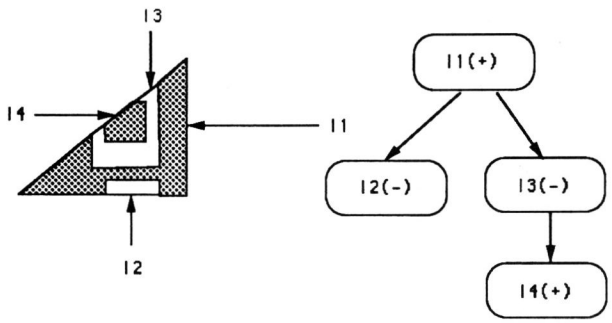


Figure 2.11 The Intersection of the Polygons in Figure 2.10

Redundant contours are eliminated as part of assembling them into a tree-structured description of the intersection, as in Figure 2.11. A directed graph is constructed in which nodes represent generated contours, and an edge goes from X to Y if and only if X surrounds Y, $X \neq Y$, and there is no contour Z such that X surrounds Z and Z surrounds Y. Next, for each outer contour (a node in the digraph to which no edge leads), the contours that it immediately surrounds (nodes accessible via a path of length 1) are identified. If any of these surrounded contours has the same sign (+ or −) as the surrounding contour, then the surrounded contour is redundant and should be eliminated. This process is applied recursively to eliminate all redundant contours. The final result is one or more trees in which the root is the outer contour of a region of intersection. Every path from an outer contour alternately visits − contours (holes) and + contours (fillers). This graph is the required description of the intersection of A and B. The logical basis for this algorithm is essentially the same as that for the algorithm in Section 2.3.

3. Polygon Set Difference

Polygon set difference can be computed by a simple variant of the procedure in Section 2.4 for finding the intersection of two polygons with holes and fillers. To find the area of polygon A not also contained in polygon B, first construct a new polygon, $B_{inverse}$, from the rectangle and B. This rectangle is the outer contour of $B_{inverse}$ and the outer contour of B and the fillers of B become the holes of $B_{inverse}$, while the holes of B become the fillers of $B_{inverse}$, i.e., the signs of the contours of B are changed. The intersection of A and $B_{inverse}$, found using the procedure described in Section 2.4, is the set difference of A and B.

4. Conclusions

Reliable geometric procedures can be developed only if the inherent limitations of finite-precision floating-point arithmetic are accommodated. This paper addresses directly these limitations in presenting accessibly new algorithms for polygon intersection (clipping) and difference which operate on input containing holes, and recursively, fillers within holes, holes within fillers, etc.

5. References

[HOFFMAN & HOPCROFT]
Hoffman, C. and Hopcroft, J. "Towards Implementing Robust Geometric Computations." *Proceedings of the Symposium on Computational Geometry*, ACM, 1988.

[MILENKOVIC A]
Milenkovic V. "Verifiable Implementations of Geometric Algorithms Using Finite-Precision Arithmetic." *Artificial Intelligence*, 37:377-401, 1988.

[MILENKOVIC B]
Milenkovic V. "Double Precision Geometry: A General Technique for Calculating Line and Segment Intersections Using Rounded Arithmetic." *Proceedings of the 30th Annual Symposium of the Foundations of Computer Science*, IEEE, 1989.

[SEGAL & SEQUIN]
Segal, M. and Sequin, C. H. "Consistent Calculations for Solids Modeling." *Proceedings of the Symposium on Computational Geometry*, ACM, 1985.

[WEILER & ATHERTON]
Weiler, K. and Atherton, P. "Hidden Surface Removal Using Polygon Area sorting." *Computer Graphics*, 11, 3.