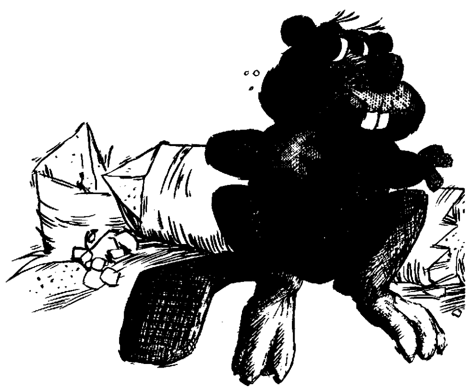
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO



Computing Finitely-Oriented Polygon Intersections

Gregory Rawlins
Derick Wood

Data Structuring Group
CS-84-40

November, 1984

COMPUTING FINITELY-ORIENTED POLYGON INTERSECTIONS[†]

Gregory Rawlins

Derick Wood

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

ABSTRACT

We present an efficient algorithm for computing all pairwise intersections in a given set of n polygons. The polygons have a bounded number e of edges and a bounded number f of edge orientations. The algorithm requires $O(e\,f\,n \log e\,n + e^2k)$ time, where k is the number of intersections, and $O(en)$ space. This improves an earlier algorithm and, moreover, requires $O(n \log n + k)$ time and $O(n)$ space when e and f are treated as constants. In this case the algorithm is time- and space-optimal.

1. The Problem

Given a set S of polygons in the plane and an integer $f \geq 2$, we say S is *f-oriented* if the edges of the polygons in S have at most f distinct orientations. We say a set S is *finitely oriented* if it is f-oriented for some $f \geq 1$. Clearly a set S of polygons is always n-oriented, where n is the number of edges in S. This notion was introduced in [Gü1] and subsequently studied in [Gü2] and [GüO], where the term 'C-oriented' is used.

Rectilinear[‡] or orthogonal polygons, which have been well studied, see the recent survey [Wo], are 2-oriented. The two orientations are orthogonal and

† Work carried out under the Natural Sciences and Engineering Research Council of Canada Grant No. A-5692.

‡ The choice of the adjective rectilinear is unfortunate in that it only means straight, rather than orthogonal or rectanguloid. However it is so well entrenched that we continue to use it.

appear in VLSI design and layout, floor plans, geographic databases, etc. However it is often the case in VLSI artwork that a 45° oriented lines are also used, that is they are 3-oriented. While this was the original motivation for introducing finitely-oriented polygons, we consider finitely-oriented polygons to be of interest in their own right. They bridge the gap between rectilinear polygons and arbitrary polygons.

Similarly, given a set S of polygons and an integer $e \geq 1$, we say S is *e-bounded* if each polygon in S has at most e edges.

We say two polygons *intersect* if they have at least one point in common.

Letting e and f be two integers, we can state the *e,f Polygon Intersection Problem* as:

> *Given a set S of polygons which is e-bounded and f-oriented, report all pairs of intersecting polygons.*

Since the problem has parameters e and f we refer to it as *(e,f)PIP*.

The reason for bounding the size of the polygons is to avoid having output of size proportional to the total number of edges in the polygons of the given set. This restriction is common, see [EKM] and [Gü2].

[Gü2] has obtained a solution to (e,f)PIP which requires $O(e f^2 n \log^2 e n + e^2 k)$ time and $O(e n \log e n)$ space, where n is the number of polygons, k is the number of pairwise intersections and e and f are as above.

In the next section we establish:

Theorem. *Each instance of (e,f)PIP containing n polygons can be solved in $O(e f n \log e n + e^2 k)$ time and $O(en)$ space, where k is the number of pairwise intersections.*

When e and f are constants, for example for sets of orthogonally-oriented rectangles, this yields $O(n \log n + k)$ time and $O(n)$ space, that is it is time- and space-optimal. Optimality follows by the usual reduction to the element uniqueness problem, see [BW] for example.

2. The Solution

To solve (e,f)PIP for a set S of polygons we proceed in two steps. First, we partition each polygon into triangles and rectangles and, second, we solve the simpler problem of detecting intersections amongst these simpler objects.

The method of partitioning each polygon is particularly straightforward. First, as in [Gü2], we vertically cut each polygon through each vertex, see Figure 1. This partitions each polygon into trapeziums with two vertical edges and, possibly, into triangles with one vertical edge. Second, we horizontally cut each trapezium and triangle through each vertex, see Figure 2. This results in orthogonally-oriented rectangles and triangles with one horizontal and one vertical edge. By construction, the hypotenuse of each triangle must be part of an edge of an original polygon and is, therefore, f-oriented. Moreover the number of triangles and rectangles obtained from each polygon is $O(e)$ and, therefore, $O(en)$ over all.

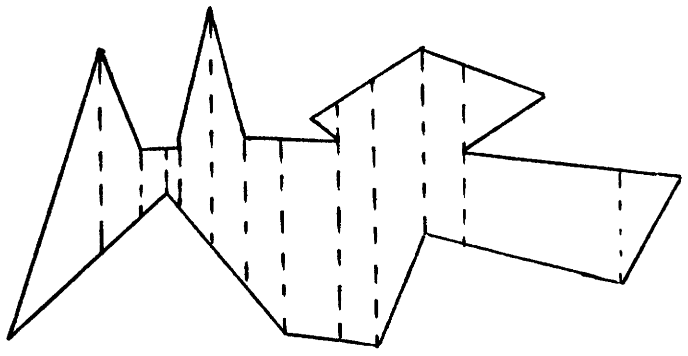


Figure 1

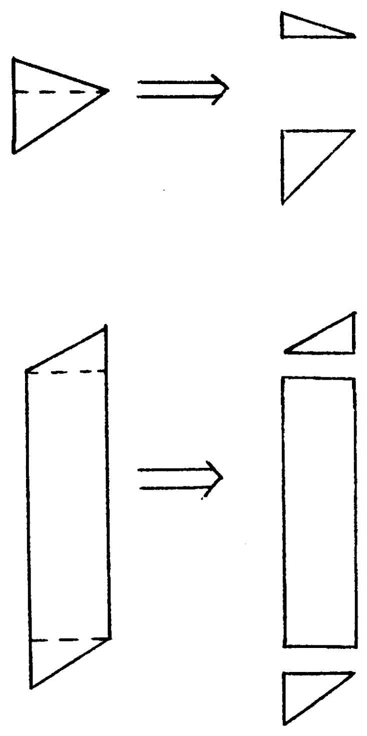


Figure 2

Rectangle-rectangle intersections can be found by any one of the methods in the literature, for example [BW], [E], and [Mc1]. However right-angled triangle intersections appear to be more difficult. We demonstrate that they are no more difficult to handle than rectangles, thus providing a basis for the solution. Consider the right-angled triangle A of Figure 3. The line which is colinear with A's hypotenuse satisfies a linear equation $y = f_A(x)$, say. The triangle determines a vertical interval $[y, f_A(x')]$ at each vertical line x' which cuts the triangle. Therefore to determine if a rectangle or some other triangle, which begins at x', intersects A we only need to test if A's vertical interval overlaps the vertical interval of the other object.

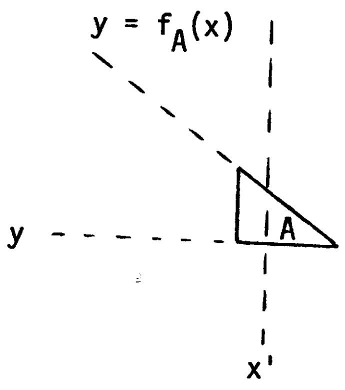


Figure 3

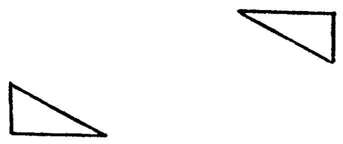
This idea is used as a basis for a plane-sweep algorithm. We sweep a vertical line, the *sweep line*, from left to right through the objects. At each left endpoint of a triangle or rectangle we determine any new vertical overlaps and also add the object to the currently active objects. At a right endpoint of an object we remove the object from the currently active objects. Noting that the vertical interval of a rectangle R is also of the form $[y, f_R(x)]$, we require a data structure that represents intervals of this form, as well as providing for efficient insertion, deletion, and querying for overlaps with such intervals. The priority search tree of [Mc2] *is* such a data structure. It provides for efficient updating with intervals and allows a query of the form $[c,d]$ which returns all intervals $[a,b]$ which satisfy $a \leq d$ and $c \leq b$, that is $[a,b]$ and $[c,d]$ overlap. The time bounds are $O(\log n)$ for insertion and deletion and $O(\log n + k)$ for a query, where k is the number of reported intervals.

It is necessary that the intervals in a priority search tree (or, indeed, in any similar structure), do not change their relative order during the plane sweep. This implies that we keep $2f+1$ priority search trees. For each orientation we have the two kinds of triangles, displayed in Figure 4, resulting in $2f$ priority search trees. The remaining priority search tree is for the rectangles.

On meeting an object for the first time during the plane sweep it causes two actions:

(i) $2f+1$ queries of the $2f+1$ priority search trees

(ii) One insertion into the corresponding priority search tree.



The two kinds of triangles with the same orientation

Figure 4

On meeting an object for the last time during the plane sweep it causes one action:

(i) One deletion from the corresponding priority search tree.

By the properties of the priority search tree we conclude that the plane-sweep algorithm requires $O(e\,f\,n \log n + e^2 k)$ time, where k is the number of pairwise intersections. This is $O(n \log n + k)$ time if e and f are treated as constants. The space requirement is $O(en)$, since while we have $2f+1$ structures, there are only $O(en)$ objects, and each object appears in at most one structure.

3. REFERENCES

[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.

[E] Edelsbrunner, H., A Time- and Space-Optimal Solution for the Planar All Intersecting Rectangles Problem, Technical University of Graz, Institut für Informationsverarbeitung, Report F50, (1980).

[EKM] Edelsbrunner, H., Kirkpatrick, D.G., and Maurer, H.A., Polygonal

Intersection Searching, *Information Processing Letters 14*, (1982), 74-79.

[Gü1] Güting, R.H., Stabbing C-Oriented Polygons, *Information Processing Letters 16*, (1983), 35-40.

[Gü2] Güting, R.H., Dynamic C-Oriented Polygonal Intersection Searching, Universität Dortmund, Lehrstuhl Informatik VI, Report 175, (1984).

[GüO] Güting, R.H., and Ottmann, Th., Efficient Algorithms for C-Oriented Hidden Line Elimination. In preparation, 1983.

[Mc1] McCreight, E.M., Efficient Algorithms for Enumerating Intersecting Intervals and Rectangles, Xerox Palo Alto Research Center, Report CSL-80-9, (1980).

[Mc2] McCreight, E.M., Priority Search Trees, Xerox Palo Alto Research Center, Report CSL-81-5, (1982).

[Wo] Wood, D., An Isothetic View of Computational Geometry, University of Waterloo, Computer Science Department, Technical Report CS-84-01, (1984).