GEOMETRIC COMPLEXITY and COMPUTER GRAPHICS -- DOES THEORY APPLY in PRACTICE?

David P. Dobkin

Department of Computer Science

CS-014

July, 1985

GEOMETRIC COMPLEXITY and COMPUTER GRAPHICS -- DOES THEORY APPLY in PRACTICE?

David P. Dobkin+

Department of Computer Science
Princeton University
Princeton, NJ 08544

ABSTRACT

Theoretical work in geometric complexity is often justified by its relevance to key problems of computer graphics, most notably the problems of hidden line and hidden surface removal. We consider a geometric structure - the convex drum - both in the context of a theoretical algorithm for polyhedral intersection and in a practical context giving an algorithm for computing and decomposing unions of polygons. This is used as a model of situations where theoretical ideas can have relevance to actual implementations.

1. INTRODUCTION

Many theoretical papers in computational geometry contain a sentence justifying their existence on the basis of applied problems of computer graphics. Despite the possibility of practical application of theoretical ideas, few authors have actually taken the step of moving ideas from one framework into the other. We do so here by considering a geometric structure -- the 2.5 dimensional drum -- which can be used in solving a theoretical problem and a practical problem. In the former case, an interesting upper bound was obtained. The latter provides the support for an useful piece of software.

It is common in computer graphics for windowing systems to consider images as being two-and-a-half dimensional. That is, a screen consists of a series of windows each specified as a 2 dimensional area (typically an isothetic rectangle), and an ordering on the windows. This ordering insures that z-coordinates can be assigned so that no two windows have overlapping z coordinates. This allows the screen to repainted in a simple manner.

The hidden surface algorithm of Newell, Newell and Sancha[NNS] relies on a similar principle. They consider an image to be specified as a collection of polygons in 3 dimensions and then determine an ordering in which the polygons can be painted to give an accurate rendering of what the eye sees. Since their polygons may overlap in z-extents, subdivision may be necessary in order to generate such an ordering. But, in each case, the 2.5 dimensional output, consisting of a set of polygons in the plane (or more generally, a set of 2 manifolds) and a paint ordering is a structure of value in computer graphics.

Typically such windowing systems or hidden surface algorithms use a painters' algorithm to display their results. The ordering they produce supports this choice. In our recent work, applications have arisen where even the existence of such an ordering does not allow the use of a painters algorithm. Rather, we need a complete description of the output as a collection of visible

+Research of this author supported in part by National Science Foundation Grant MCS83-03926. Portions of this work were done while the author was visiting the Computer Science Laboratory at Xerox PARC. An earlier version of this paper was presented at the SIAM Conference on Geometric Modeling and Robotics, Albany, New York, July, 1985

portions of the input polygons. This problem arises from two considerations -- first, the desire to move images among display devices of varying characteristics requires that we produce images which specify pictures as non-overlapping fragments. Also, our use of anti-aliasing to produce high quality images makes the use of a painter's algorithm impossible since the display of a pixel might involve many polygons. Scan-line solutions to this problem have been discussed in [Car,Cat].

In our application, the inner loop depended upon a procedure which reduces two convex (planar) polygons to a set of convex polygons comprising the different regions of the union of the two polygons. Polygons in the solution correspond to disjoint regions which belong to one or both of the polygons. After considering various algorithms for solving this problem without resorting to analyzing special cases, we chose a solution which conceptually involves considering the problem as a convex hull problem in 2.5 dimensions. The convex hull is now a drum and the added edges lead to the desired decomposition. We present here a practical version of this idea.

We are using a version of this algorithm to combine triangles in our hidden surface algorithm[Cl] from which Figure 1 were produced. A more general form of the algorithm is being used to develop a general polygon combiner at Xerox PARC[Ar].

In earlier work[DK], David Kirkpatrick and I considered the problem of determining whether two 3 dimensional convex polyhedra intersected. The goal of that research was to find a (theoretically) efficient algorithm which would provide a witness to the intersection or separation of two polyhedra. The work was justified on the basis of applications to practical problems. Our hope was to add some geometric intuition to the already existent algorithms for the problem[CD].

The previous algorithms had shown that convexity is a useful property in such problems. A convex polygon can be decomposed into monotone chains which appear to be 1-dimensional. The intersection problem then reduces to one of determining if two piecewise linear chains intersect. Monotonicity and convexity then lead to a logarithmic solution to this problem.

In our attempts to extend and simplify previous results, we sought lower dimensional structures from which to build results for three dimensional polyhedra. The structure which emerged was the drum. We found that polyhedral intersection algorithms could be derived from drum intersection algorithms in a manner which will be made precise below.

In the next section, we describe the drum as a basic 2.5 dimensional object and discuss certain of its properties. Section 3 then describes the details of the Dobkin-Kirkpatrick polyhedron intersection algorithm which build upon our understanding of drums as 2.5 dimensional objects. The fourth section describes the lens decomposition algorithm and it's use in anti-aliased hidden surface procedures. And, we conclude with a discussion of how the methodology of this paper might be used elsewhere to bring about a fruitful exchange between computational geometry and computer graphics.

2. The Geometry of Drums

Intuitively, a drum is a 3-dimensional convex object such that all of its vertices lie on a pair of parallel planes. For example, a cube is a drum as is a ruled cylinder. A pyramid is a partially degenerate drum.

Formally, a drum is specified by a pair of parallel planes which we refer to as *top* and *bottom* , a polygon in each of vertices $t_1,t_2,\cdots,t_n$ and $b_1,b_2,\cdots,b_m$ and a collection of e ($\leq$ n+m) edges pairing top vertices to bottom vertices. Edge i connects top vertex $t_{tv(i)}$ to bottom vertex $b_{bv(i)}$. For all i between 1 and e, with subscripts interpreted modulo e, either tv(i) = tv(i-1) or bv(i) = bv(i-1) or the four point $t_{tv(i)},t_{tv(i-1)},b_{bv(i)},b_{bv(i-1)}$ are coplanar.

A drum can also be viewed as a function f which performs a continuous deformation from bottom into top along the edges. The properties of drums which we will use involve their convexity and separations of pairs of drums. We state them in the following.

Fact:

1) A drum is exactly the convex hull of its top and bottom

2) A positive change of scale in the direction normal to the top and bottom of a drum cannot have an effect on its convexity. That is, if the planes supporting the top and bottom of a drum are translated perpendicular to their normals away from each other, the resulting object is always convex.

3) If two drums are non-intersecting then there either there is a plane supporting a lateral face or edge of one which separates them or if no such plane exists, there is a plane supporting an edge of the top or bottom of one which separates them. We can distinguish among these situations.

In what follows, we shall use convexity in various situations. Fact 2 will be used to aid in the decomposition results stated in section 4. The separation result (Fact 3) forms the basis of the intersection algorithm stated in the next section.

3. Using Drums to Detect Polyhedral Intersections

In this section, we follow the development of [DK] to produce an algorithm for determining if two convex polyhedra intersect. We assume that P and Q are convex polyhedra which have been previously decomposed into drums. We assume further that this decomposition was done independently on the polyhedra. In each case, the decomposition was done by choosing a direction and passing planes parallel to the direction through the vertices of the polyhedron. The resulting cross-sections and adjoining edges then yield a sequence of drums. No assumption is made about the direction of the cross-sections. Thus, cross-sections of one polyhedron will not be parallel to those of another. Thus, drum decomposition can be done once (e.g. as the polyhedron is entered into our system) and remains invariant under translation, scaling and rotation. Note that a polyhedron of n vertices will give rise to a decomposition of no more than n-1 drums.

For the development of our algorithm, we assume the following result:

Lemma [DK]: If D and E be drums involving a total of n vertices, then O(log n) operations suffice to determine either

a) a point in their intersection, if one exists or

b) a plane separating the drums which supports either a face which is not the top or bottom of either, or an edge which joins top and bottom of one, if one exists or

c) a plane separating the drums which supports an edge of the top or bottom of one.

We refer the reader to [DK] for a proof of this Lemma and proceed to apply it to the intersection detection problem.

Our algorithm will follow a merging strategy which relies on convexity to eliminate half the remaining drums of one polyhedron for each application of the above Lemma. Notice that in each test for intersection of two drums, if an intersection is reported (case a of the Lemma), the algorithm terminates having found a witness to the intersection of P and Q. Otherwise, the separation information is used to form the smaller problem from which we proceed.

We begin by considering the middle drum of each polyhedron. Assume that these drums do not intersect. If case b) occurs, it is easy to show that the separating plane also supports a face of the polyhedron and separates that face from the other drum. Let D be a drum of P and E a drum of Q. Assume, wlog, that we have a plane supporting a face of P which separates P from E. Note that this plane can intersect Q above or below E but not both. Convexity allows us to determine which case occurs in logarithmic time, since the drums intersected by the plane must be contiguous. Once one such drum has been determined, half the drums of Q can be eliminated from further consideration.

Case c) is more difficult. Here we assume, wlog, that the top of D separates D from E. The extension to the case where a plane supporting a top edge is separating is straightforward. We now define the *cone* of a drum as the intersection of the planes of support of all of its faces (except its top and bottom). Note that the cone of a drum will contain any convex polyhedron for which this is a drum. Since the only separation of D and E supports a top or bottom face, the

cone of D (which we call co(D)) must intersect E. Let this intersection be the set S. Since the top of D separates D from E, S must intersect co(D) above D. Let **x** be a point of this intersection. This leads to:

Claim: S cannot intersect co(D) below D.

If this claim is true then the bottom drums of P (including D) can be eliminated from further consideration. Hence it remains to prove the claim. This proof follows from the convexity of co(D) $\cap$ S. Let **y** be a point which lies below D and in co(D) $\cap$ S. Now, the segment connecting **x** and **y** must lie within co(D) $\cap$ S. This segment must pass thru D and hence must contain a point of D. But, by the definition of S, this point must also belong to E. Hence D and E intersect, which is a contradiction.

This results in :

Theorem: Let P and Q be polyhedra which have been decomposed into drums. Assume that P and Q involve a total of n vertices, then $O(\log^2 n)$ operations suffice to compute

a) A point of P $\cap$ Q or

b) A proof that P and Q do not intersect.

A theoretical analysis of the drum-drum intersection detector suggests that this algorithm may be reasonably efficient. The program to compute this intersection does, however, involve some complicated case analysis. Using the drum-drum method to detect polyhedral intersections as we've described above is likely to be quite efficient. However, the preprocessing required to realize these algorithms makes their ultimate practicality dubious.

A naive approach to preprocessing a polyhedron of n vertices is to use $O(n^2)$ operations to compute all cross-sections and then $O(n^2)$ storage to store the resulting drum representation. In this case, retrieval in O(1) operation is possible. Unfortunately, the space requirements here are prohibitive except for small values of n. And in these cases, better methods exist.

An alternative to naive preprocessing is to compute a data structure which requires less space (e.g O(n) or O(n log n)) and which stores sufficient detail to enable portions of drums to be reconstructed efficiently. Such a structure was proposed in [DM] and improved in [Ov] resulting in an $O(\log^3 n)$ intersection detection algorithm after polyhedra had been preprocessed in O(n log n) time into a structure requiring linear storage. While this structure is theoretically faster, it adds another layer of overhead which makes its practical utility questionable.

While the results of this section have not yet had an impact on practical graphics software, they do increase our insight into geometric structures which might be computationally useful. In the next section, we see an example of a practical application of drums.

4. Using Drums to Compute and Decompose Polygon Unions

Next we consider the application of the drum structure to the hidden surface elimination problem. We assume that our input consists of a collection of convex polygons* lying in R^3. A viewing position is also given. The goal is to produce a description of the scene as viewed from that position. This output consists of a collection of non-overlapping polygons which represent the visible scene. We choose this model based upon hardware and software considerations. In our computing environment, pictures are transported among devices of various resolutions and characteristics, making it unreasonable to transport bitmaps. Furthermore, some of our devices support firmware smooth-shading of polygons. Our desire for high-quality images causes us to use anti-aliasing rendering methods which make painters algorithms inappropriate.

We consider first the problem of computing the difference of two polygons. It is generally the case that this difference of polygons will involve more than one connected component. For each connected component, our goal is a decomposition into triangles or convex polygons.

---

* Throughout this section, polygon means **convex polygon**.

We now describe a procedure for determining all edges which might be necessary to compute and triangulate the difference between two polygons P and Q, where we assume that P is contained in Q. Our first observation is that all edges that must be added could be found by moving one polygon to a parallel plane, considering the two polygons as the top and bottom of a drum and compute the convex hull of the drum. Note that the relevant portion of the convex hull computation is not the lengths of edges added to connect the polygons, but rather the endpoints of the edges. Hence, the computation easily moves back to the original plane.

We further observe that edges added are exterior to polygon P and interior to Q. This follows from the definition of convexity. In particular, any parallel cross-section of the drum must represent a step in the transformation (which is actually a contraction) of Q into P. It remains to show that this convex hull can be computed in linear time which we do below for the more general case.

Similarly when P and Q intersect, the drum convex hull can be used. Here, those edges which lie outside both P and Q are not relevant to the decomposition of the polygons arising in their union. And, those edges surrounding a point of intersection must be modified to yield the edges of an intersection. This situation is explored in more detail below and shown in Figure 2.

Finally, when P and Q do not intersect, all added edges will be exterior to (at least) one of the polygons. While these edges will be unnecessary for decomposing the polygons in the union of P and Q, they will be useful should the convex hull of P $\cup$ Q be desired.

The details of this polygon decomposition are best considered as three separate cases. In each case, the drum convex hull plays a pivotal role in the final algorithm, but the details work differently in terms of actual implementation. Let the polygons be P and Q and consider the cases where P and Q are disjoint, where P lies within Q (the case where Q lies within P being symmetric) and where P and Q have a boundary intersection. We assume initially that we are able to distinguish among these cases. In the first case, no further computation is necessary since the decomposition of the polygonal union is the two separate polygons. The second case involves computing a drum convex hull directly and the third case involves decomposing lenses (connected regions lying in one polygon and not the other). While these processes are similar, it is worthwhile to consider them separately here for the purposes of exposition.

If P lies wholly within Q, find a point, x, of their intersection (i.e. within P). Next, divide the plane into sectors, where each sector is a wedge anchored at x and containing the region of the plane between adjacent vertices of P (in counterclockwise order). A vertex of Q can now be identified by the sector of the plane in which it lies. If each sector contains vertices of Q, a decomposition is formed involving a polygon for each sector along with a polygon (actually a triangle) filling in between polygons for neighboring sectors. The sector polygons consist of the two vertices (of P) from which the sector is created along with all vertices of Q which lie within the sector. Triangles connecting sector polygons involve two vertices from Q and one from P. The P vertex is the vertex common to adjacent sectors and the Q vertices are the last vertex lying in one sector along with the first vertex lying in the adjoining (in counterclockwise order) sector.

If a sequence of vertices of P give rise to empty sectors, it must be the case that portions of a single edge of Q lie within the individual sectors (though outside of P). Thus a polygon can be created consisting of this edge along with all vertices of P corresponding to empty sectors. It remains to triangulate this non-convex polygon. This is done by finding the vertex of P closest to the edge and connecting that vertex to both endpoints of the edge. Vertices (of P) before that maximal vertex are then connected to the initial endpoint and those after the maximal vertex are connected to the final endpoint. Since the distances of vertices from the edge form a unimodal sequence, this process is easily done while the vertices of P are traced. Since these triangles function as triangles joining (now empty) polygonal sectors, there is no need for additional polygons joining sector polygons in this region of P. Pseudo-code for the situation where P lies within Q is similar for the pseudocode for the situation for lens decomposition which is included below.

There now remains the case where the polygonal boundaries actually intersect. Here the situation is similar to the situation where P lies strictly within Q once lenses of the intersection

have been identified. The picture can be decomposed into a convex polygon representing the polygonal intersection along with a set of lenses. Each lens is bounded by a point common to both boundaries, followed by a sequence (in counterclockwise order) of vertices of one polygon followed by a point common to both boundaries, followed by a (possibly empty) sequence (in clockwise order) of vertices of the other polygon. That is, a lens is the region between two polygonal chains. We refer to the chains as O (for outer polygon) and I (for inner polygon) and label the intersection points as S and E. Thus a lens consists of the vertex chain $S, O_i, O_{i+1}, \ldots, O_j, E, I_l, I_{l-1}, \ldots, I_k$. Note that no vertices of the inner polygon need appear, but at least one vertex of the outer polygon must occur. With the exception of edges involving S, O_i, O_j, E, I_k and I_l, the added edges involved in the decomposition are edges involved in that portion of the drum convex hull involving these vertices of the underlying polygons P and Q.

Our lens decomposition routine requires a subroutine which takes an ordered sequence of three vertices in the plane and determines whether they are clockwise or counterclockwise in orientation. As convex polygons are formed, typically by adding edges, they are output. As above, we assume that arithmetic of polygonal indices is done modulo the number of vertices.

5. Conclusions

Our goal here has been to demonstrate the utility of a theoretical construct in the solution of a practical problem. The drum appears to have potential to extend beyond the decomposition problem considered here and to find application to a wider range of computational problems. Extensions of this work are currently proceeding [Ar] to understand the possible extensions of such a method.

On the theoretical side, there appears to be potential for extending the ideas here to higher dimensions. Just as the ability to lift 2 dimensional problems to a 2.5 dimensional world aided our understanding, the analog of the 2.5 dimensional drum in 3.5 dimensions may prove productive. In particular, this might be the appropriate tool for decomposing the union of two convex 3-polyhedra.

Finally, we hope that this work will encourage further cooperation between those pursuing algorithms for interesting theoretical problems in computational geometry and those seeking solutions to practical problems of computer graphics.

6. Acknowledgements

Numerous conversations with Bill Thurston and Dennis Arnon are acknowledged. In particular, the initial idea of computing polygonal decompositions as drum convex hulls was suggested by Bill. Dennis greatly enhanced an initial implementation eliminating numerous bugs in the process.

The software implementation of the lens decomposition for triangles which produced the images in Figure 1 was done by Matthew Clark.

The readability of this document was improved by helpful comments from Diane Souvaine.

7. References

[Ar] Arnon, D., Lamming, M. , Greene, D. and Dobkin, D., A Methodology for Shape Combination, Internal Memo, Xerox PARC, July, 1985.

[CD] B. Chazelle and D. Dobkin, Detection is easier than computation, ACM Symposium on Theory of Computing, Los Angeles, Ca, May, 1980,146-153, to appear in *JACM* as Intersection of convex objects.

[Car] Carpenter, L., The A-buffer, an antialiased hidden surface method, *Computer Graphics* 18,3, 103-108.

[Cat] Catmull, E., A hidden surface algorithm with antialiasing, *Computer Graphics* 12,3, 6-11.

[Cl] Clark, M., Triangle subdivision algorithm for hidden surface removal of three dimensional objects with shading and shadowing, Senior Thesis, Princeton U., 1985.

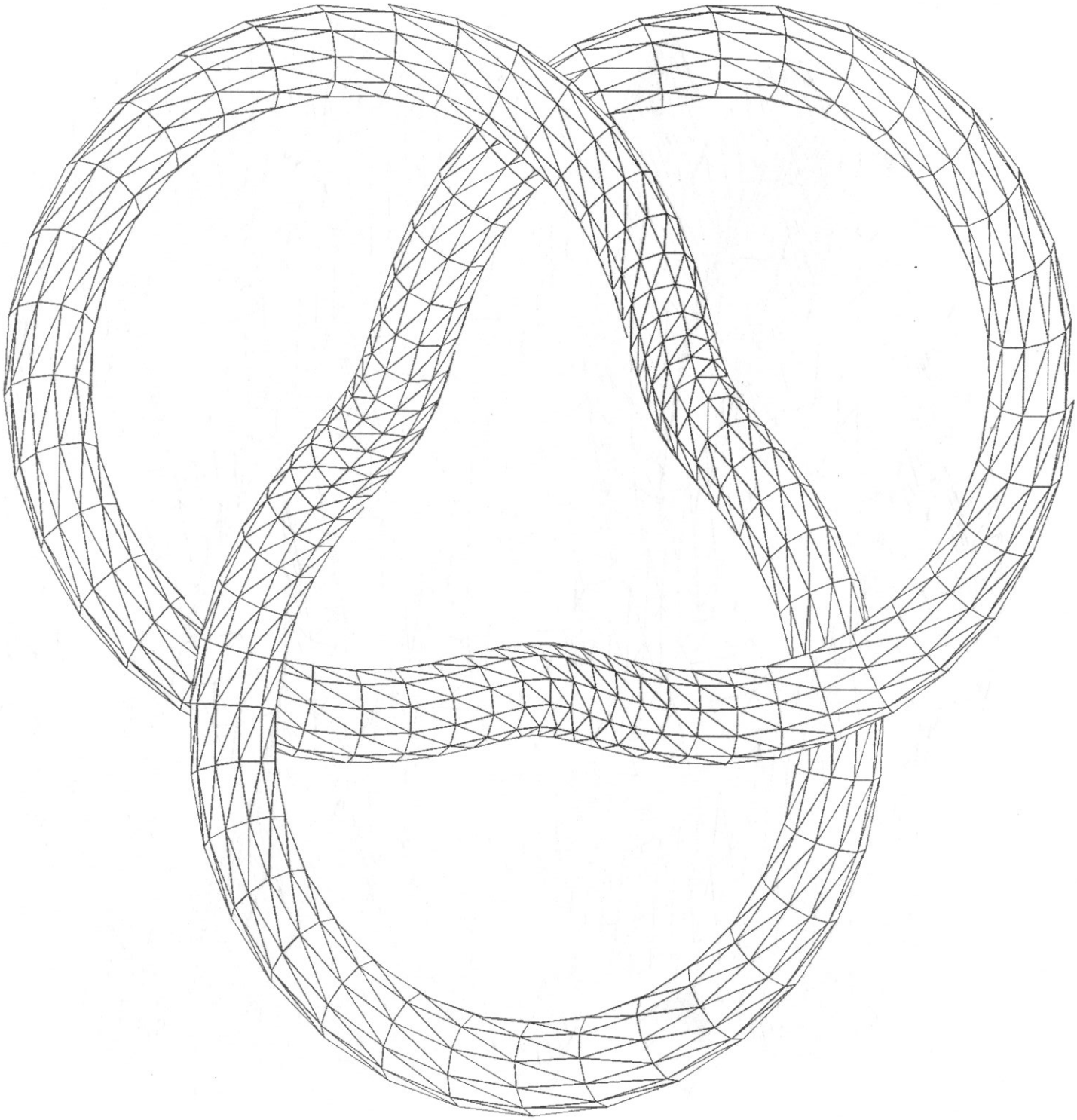
[DK] Dobkin, D.P. and Kirkpatrick, D.G., Fast detection of polyhedral intersections, *Theoretical Computer Science* 27 (1983) 241-253.

[DM] Dobkin, D. and Munro, J.I., Efficient Uses of the Past, *Journal of Algorithms,* to appear.

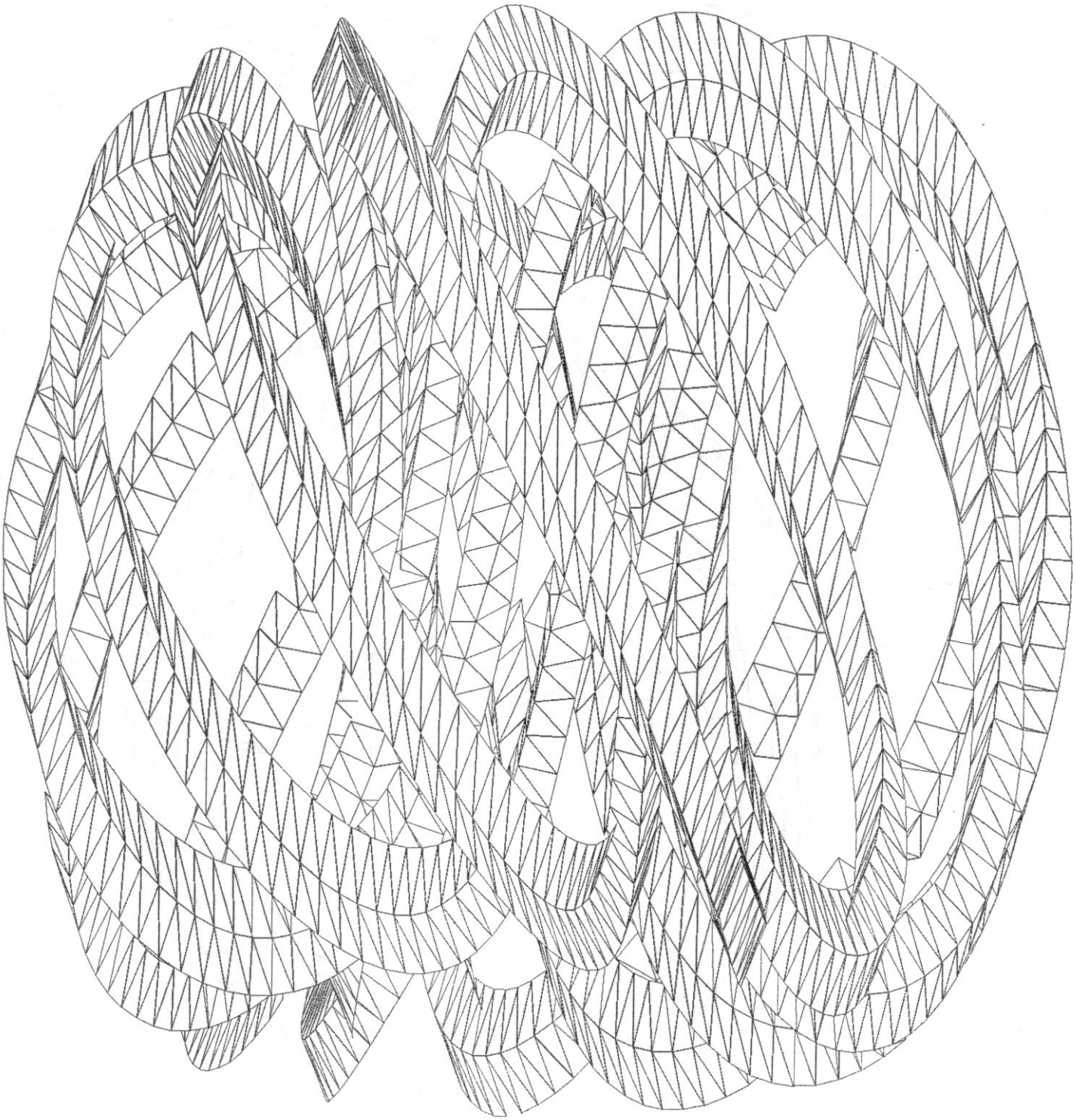
[NNS] Newell, M.E., Newell, R.G., and Sancha, T.L., A new approach to the shaded picture problem, *Proc. ACM National Conference,* 1972, p. 443.

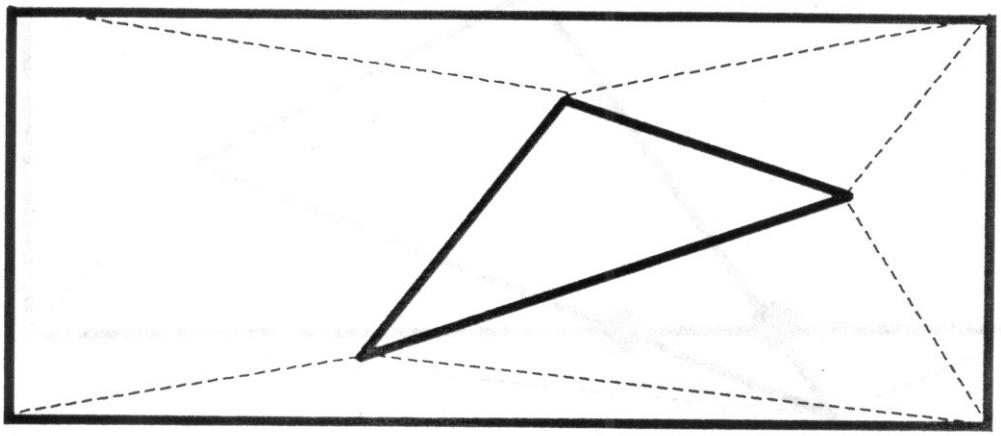
[O] Overmars, M.H., Searching in the past, Parts I and II, University of Utrecht Technical Reports, 1981.

[Sh] M. Shamos, *Computational Geometry,* PhD Thesis, Yale U., May, 1978.

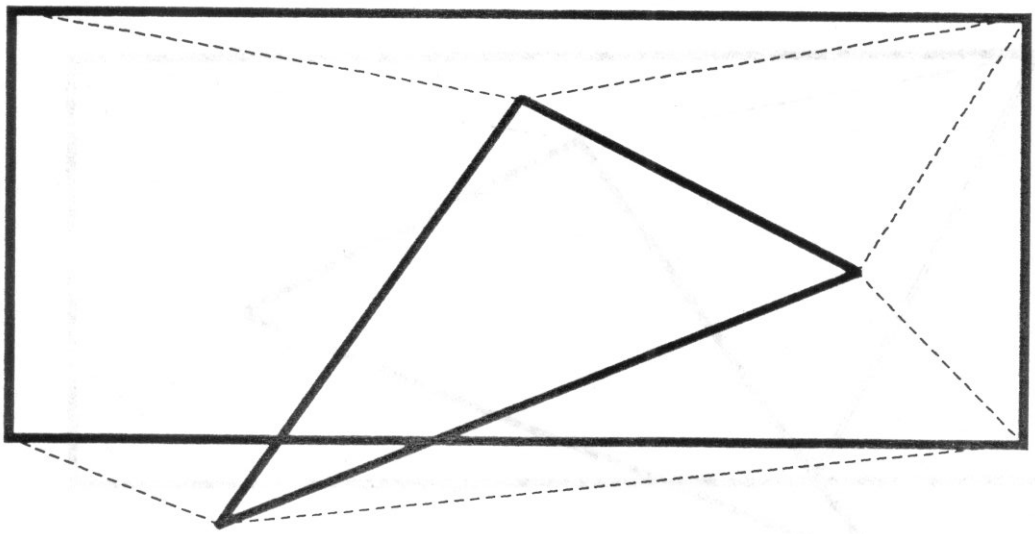


Figures 1 -- on this and the following page are two torus knots which were created as 3 dimensional triangulated polyhedra and rendered through a hidden surface removal based upon the algorithm given in this paper.

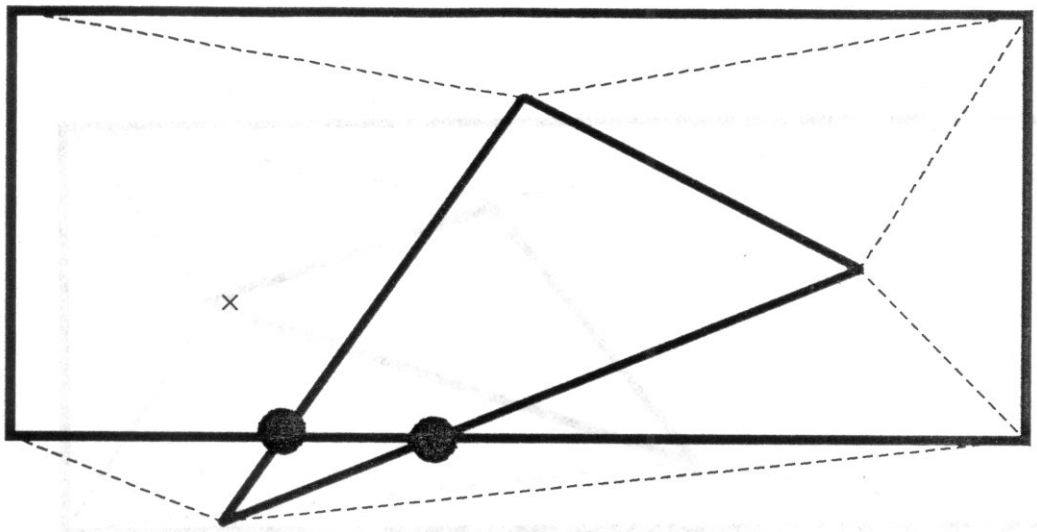




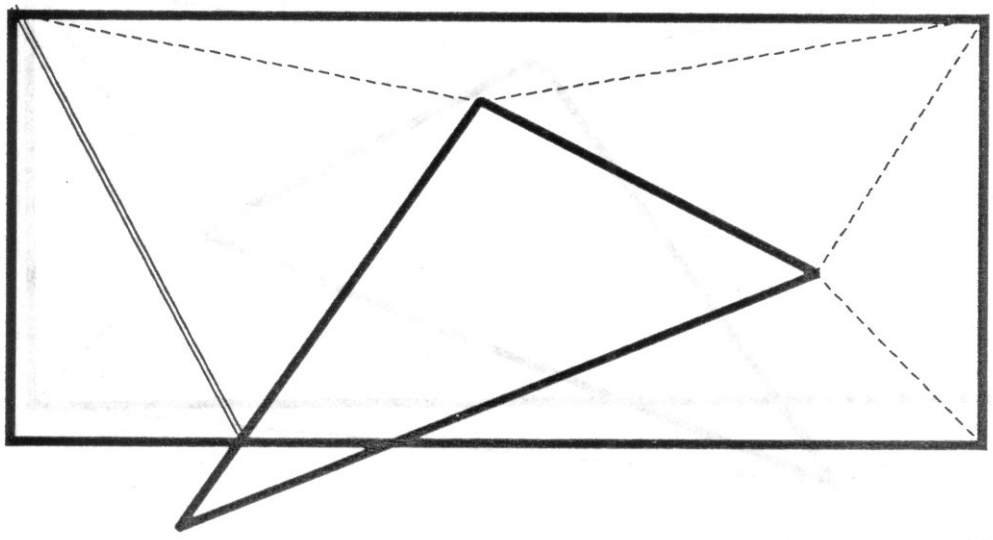
The triangle lies within the rectangle.
Edges of the decomposition are those that would be added to form the drum convex hull.



The triangle intersects the rectangle.
Edges of the decomposition are modifications of those that would be added to form the drum convex hull.



Intersection points are identified and lenses are then decomposed. Resultant edges are derived from the drum hull edges.



The final decomposition. The dotted edges would be added by the algorithm, the doubled line would be added when triangulating the resulting output.

Figure 2 Decomposing the union of two polygons

```
Decomp : --decompose the lens formed from convex chains  I and O.
        --I is the inner polygon with vertex indices ranging from k to l(cw order)
        --O is the outer polygon with vertex indices ranging from i to j(cw order)
        --note that I can be empty but O cannot

        --Edge of I ending at k and edge of O ending at i intersect at point S
        --Edge of I starting at l and edge of O starting at j intersect at point E
        --Subscripts are interpreted modulo the number of vertices in a polygon.
        --Thus, i = i + 1 does not always correspond to adding 1
BEGIN

Oi : PolygonIndex in O --initially i
Ik : PolygonIndex in I --initially k

IF (I is empty) output the polygon S, O[i], O[i+1], ...O[j],E and exit;

WHILE (triangle S,I[Ik],O[Oi+1] is counterclockwise) Oi = Oi + 1;
            -- Oi is incremented since another vertex of O
            -- can be added to the growing polygon while
            -- preserving convexity.

Add Edge from O[Oi] to I[k] to close off polygon (I[k],S,O[i],O[i+1],...,O[Oi]);
            -- the initial opening polygon.
            -- note that this will always be at least a triangle.


FOR Ik FROM k TO l-1 BY 1 DO BEGIN
      OldOi = Oi;
      WHILE (the triangle I[Ik+1],I[Ik],O[Oi+1] is clockwise)
                  Oi = Oi + 1
            -- Oi is incremented since another vertex of O
            -- can be added to the growing polygon while
            -- preserving convexity.

      IF (Oi != OldOi) THEN Add Edge from O[Oi] to I[Ik+1]
            to close off polygon (I[Ik+1],I[Ik],O[OldOi],O[OldOi+1],...,O[Oi]);
            -- the next polygon;
      ELSE -- there were no O vertices in the sector
            WHILE (the triangle I[Ik+1],I[Ik],O[Oi+1] is counterclockwise)
                  (Add Edge from I[Ik+1] to O[Oi] to close off
                        the triangle I[Ik+1],I[Ik],O[Oi]; Ik = Ik+1;)
      Add Edge from O[Oi] to I[Ik] to close off triangle I[Ik],O[Oi],O[Oi+1];
      Oi = Oi + 1;
      END

Output the final polygon which of E,I[l],O[Oi],O[Oi+1],...,O[j];

END ;
```

Figure 3 -- Pseudocode for Lens decomposition