DETECTING THE INTERSECTION OF CONVEX OBJECTS IN THE PLANE

David P. Dobkin
Diane L. Souvaine

CS-TR-231-89

October 1989

Detecting the Intersection of Convex Objects in the Plane*

David P. Dobkin

Department of Computer Science
Princeton University
Princeton, NJ 08544

Diane L. Souvaine

Department of Computer Science
Rutgers University
New Brunswick, NJ 08903

ABSTRACT

Numerous applications require intersection detection. Many algorithms have been developed for telling whether two polygons intersect. We have extended one such algorithm to allow us to determine in $C \log n$ operations whether two convex planar regions intersect. Our algorithm is significant because it can be presented as a combination of two ideas. First, there is a revision of previous algorithms for detecting whether two convex polygons intersect. Second, there is a general method for transforming algorithms which work for polygons to make them work for piecewise curved boundaries. The constant C depends strictly upon the complexity of the piecewise curves. The algorithms presented here have been implemented and details of their implementation are included.

1. Introduction

The numerical operation at the core of graphics and geometric systems almost always involves determining the intersection of 2 objects. This is the case in all of the diverse contexts one might consider within interactive graphics, CAD/CAM, and computational geometry. For example, when designing a user interface, the central operation involves hit detection. And, a window manager must manage motion of bitmaps by determining intersections of rectangles[GY]. Similarly, ray tracers rely upon procedures for determining the first intersection of a ray with a collection of objects [Wh]. In the context of solid modeling, intersection operations are pervasive in both the definition of

* This research supported in part by the National Science Foundation under Grants CCR87-00917 and CCR-88-03549, a Henry Rutgers Research Fellowship and a Guggenheim Fellowship.

objects (e.g. CSG [Br] or B-rep [Mo]) and in their maintenance. Because of this, intersection has been widely studied within the graphics literature.

Unfortunately, the wide variety of situations in which intersection computations are required have made it difficult for any single algorithm to be deemed good across contexts. There does not currently exist a parameterized class of algorithms which span applications areas based on individual requirements. Our goal here is to fill this need. We begin with the fundamental observation that such problems seldom require the actual intersection of objects. Rather what is needed is information about the intersection. For example, often an application desires merely to know of the existence of an intersection and perhaps a common point or separation information or a pair of closest points in some direction. This detection problem is our focus.

In this and a companion paper, we provide a unified approach to the intersection detection problem. Here, we develop algorithms for intersecting convex regions of the plane without regard to the elements which compose the boundary. In [DS2], we extend the work to 3 dimensions. Our main results are algorithms which in a number of operation logarithmic in the size (ie number of elements of the boundary) of the regions detect the intersection of two convex planar regions. An operation here is defined as either a data structure manipulation or a computation involving a boundary element of one or both regions. This work is an outgrowth of theoretical work previously reported in [CD1,DK]. We also believe that the approach of expressing algorithms in general terms by expressing primitive operations as subroutine calls which are capable of computing in the relevant domains is of interest in its own right. This approach is explored in greater detail elsewhere[So, DS1].

The algorithms described here have been implemented. We include implementation details and code fragments as appropriate throughout the text. The figures included in the text have been generated by these programs whenever possible.

The remainder of the paper is organized as follows. In the next section, we rederive the intersection algorithm of [DK] for detecting polygonal intersections. This rederivation is done to make the generalization to curved regions easier. In addition, the process of rederiving and implementing has uncovered some minor bugs which have been fixed in this new version. Next, in section 3, we show how to generalize the notion of a polygon to that of a splinegon. A splinegon here is a polygon which has been enhanced by replacing its edges by spline curves of a prescribed type. And, we show that our definitions are such that virtually all of the development of section 2 transfers immediately to this new model. In section 4, we conclude with a description of extensions and open problems.

2. Detecting the intersection of two Convex Polygons

For this section, we will let P (resp. Q) be a convex polygon of p (resp. q) vertices and N = p+q. Now, we rederive the intersection algorithm of [DK] to show how the intersection of P and Q can be detected in time $O(\log N)$. This algorithm consists of 3 steps each of which will be the highlight of one of the following sections.

1. Split P (resp. Q) into left and right polygonal chains P_L and P_R (resp. Q_L and Q_R) such that P and Q intersect if and only if both pairs P_L and Q_R and Q_L and P_R intersect.

2. For a left half/right half polygonal chain pair, we show how to do a binary search like procedure to eliminate half of one chain in constant time.

3. After step 2. has reduced one polygon to few sides, we show how to complete the work while reporting either the intersection point or separating line.

2.1. Splitting a polygon into 2 parts

By definition, a convex polygon P of N vertices can be split at the points having maximum and minimum y-coordinate into left and right monotone chains of vertices, P_L and P_R. Assuming that the vertices of the polygon are already stored in random access memory, this splitting process requires $O(\log N)$ ordinary operations [CD1]. This is done by identifying the vertices of maximum and minimum y-coordinate. We then build a clockwise chain of vertices and a counterclockwise chain of vertices, both from bottom to top. The first (resp. second) chain is P_L (resp. P_R) and consists of those vertices which would be exposed by a light shining from the line $x = -\infty$ (resp. $x = -\infty$). In what follows, we will always consider P_L and P_R to be polygons defining semi-infinite regions of the plane. This will be achieved by adding horizontal rays towards $+\infty$ to the top and bottom vertices of P_R (and towards $-\infty$ to P_L). It is easy to see that P is contained in P_L and P_R and is equal to their intersection. Figure 1[1] shows a polygon P and the corresponding P_L and P_R.

To find the vertices of largest and smallest y-coordinate, we first break the polygon into 2 chains which are unimodal in y. The first chain will holds the top of the polygon and the second the bottom. We do this split by identifying the four vertices tl, tr, bl and br representing (t)op/(b)ottom (l)eft/(r)ight points. Next we do a search for the maximum (or minimum) of the unimodal function which assigns y-coordinates to vertices between tl and tr (or bl and br). Finally, we create copies of the vertices making up the chains.

[1] Most figures in this paper were made from actual executions of the code we describe

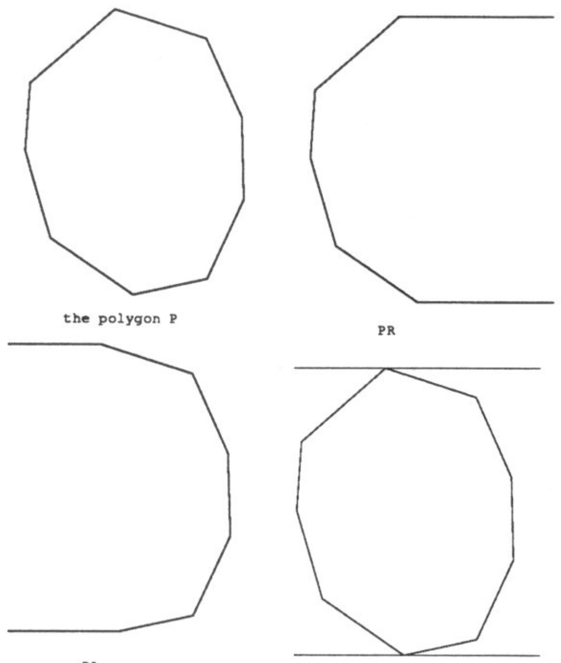

Figure 1: A polygon and its shadows

This is all achieved in the code fragment given below[2].

```
#define RGHT    -1
#define LFT     1
#define MIN     0
#define MAX     1

/*
 * shadow divides p into a left half (pl) and a right half (pr)
*/
shadow(p,pl,pr)
struct polygon *p, *pl, *pr ;
{
        int tl,tr,bl,br,hi,lo ;

/* first we find the limits to break p into two chains unimodal in y */
find_lr_lb_tr_tb(p,&tl,&tr,&bl,&br);

/* next we find the vertices of maximum and minimum y coordinate */
hi = uni_opt(p,tr,tl,MAX); lo = uni_opt(p,bl,br,MIN);

/* finally we make the two chains */
make_chain(p,lo,hi,pl,LFT); make_chain(p,lo,hi,pr,RGHT);
}
```

shadow

```
find_lr_lb_tr_tb(p, tl, tr, bl, br)
```

find_lr_lb_tr_tb

2 Code fragments given in this paper depend upon certain data structures having been defined. In particular, a **point** is a structure with the two fields **x** and y which are of type **double** and a polygon consisting of an **integer** field **nverts** giving its size and an array v of **points** which contain its vertices. Where structures are not defined it is because they should be obvious from the context.

```
struct polygon *p;
int *tl, *tr, *bl, *br ;
{
double diff1 = p->v[0].y - p->v[p->nverts-1].y ;
double diff2 = p->v[1].y - p->v[0].y ;

if (diff1 > 0.0 && diff2 > 0.0)                        {
/* vertex 0  is above vertex -1 and below vertex 1 */
                *tr = 0 ;
                *br = p->nverts-1;
                *tl = *bl = find_vertex(p,LFT);
}
else if (diff1 < 0.0 && diff2 < 0.0)                   {
/* vertex 0  is below vertex -1 and above vertex 1 */
                *tl = p->nverts-1;
                *bl = 0;
                *tr = *br = find_vertex(p,RGHT);
}
else if (diff1 >=  0.0 && diff2 <= 0.0) {
/* vertex 0 is at least as high as vertex -1
                and at least as low as vertex 1 */
                *bl = 1;
                *br = p->nverts-1;
                *tl=*tr= ((diff1==0)?p->nverts-1:0); /* checkfor horizontal edge */
}
else if (diff1 <= 0.0  && diff2 >=0.0 ) {
                *tr = 1;
                *tl = p->nverts-1;
                *bl=*br= ((diff1>-EPS1)?p->nverts-1:0);
}
}
```

```
/* this is the test for other end */
find_vertex(p,orient)
struct polygon *p;
int orient;
{
int bot = 0, top = p->nverts-1, vertex = 0, mid;
double diff1, diff2, base;

while ( !vertex )                   {
                mid = (top+bot)/2;
                base  = p->v[mid].y - p->v[0].y ;
                diff1 = p->v[mid].y - p->v[mid-1].y ;
                diff2 = p->v[mid+1].y - p->v[mid].y ;
                if (diff1 * orient < 0) vertex = mid;
                else if (diff2 * orient < 0) vertex = mid+1;
                else if (base * orient < 0) top = mid;
                else            bot = mid;
}
return (vertex);
}
```

find_vertex

```
make_chain(p,from,to,pc,orient)
struct polygon *p, *pc ;
int from, to, orient ;
{
int i = from - orient, j = 0;

/* find size of chain */
if (orient == LFT)
pc->nverts = (to-from+p->nverts)%p->nverts + 1 ;
else
pc->nverts = (from-to+p->nverts)%p->nverts + 1;
```

make_chain

```
/* allocate chain */
allocate_polygon(pc);

do          {
            i = (p->nverts + i + orient) % p->nverts ;
            copy_vertex(&pc->v[j],&p->v[i]) ;
            j++ ;
}
while (i != to) ;
}

/* find the max of the unimodal function */
uni_opt(p,i,j,maxmin)
struct polygon *p;
int i,j,maxmin;
{
int m=i, M=j, cur1,cur2,extvert ;
double diff1, diff2;

while ((M-m) > 2)           {
            cur1 = (m+m+M)/3;
            cur2 = (m+M+M)/3;
            diff1 = p->v[cur1].y - p->v[cur2].y ;
            if (maxmin != MAX) diff1 *= -1;
            if (diff1 <= EPS1) m = cur1 ;
            if (diff1 >= -EPS1) M = cur2 ;
}
}
```

uni_opt

The shadowing process is justified by the following lemma from [DK].

Lemma 1. Let P and Q be convex polygons which have been divided into P_L, P_R, Q_L, and Q_R as described above. Then P and Q intersect if and only if the polygon pairs P_L and Q_R and P_R and Q_L intersect.

2.2. Binary search on polygonal chains

Let L be a polygonal chain open infinitely to the left and R a chain open infinitely to the right. Furthermore, let λ be an edge of L which is supported by line Λ and let ψ be an edge of R which is supported by line Ψ. (See Figure 2).

We observe that L (resp. R) lies strictly to the left (resp. right) of the line Λ (resp. Ψ). And, the lines Λ and Ψ divide the plane into 4 regions. We refer to these are the *L*-region, the *R*-region, the *LR*-region (where *L*- and *R*-regions intersect) and the $\emptyset$-region (where nothing exists)[3]. If an intersection occurs, it must occur in the *LR*-region. We

[3] Actually, if Λ and Ψ are parallel, then only 3 regions will result. In this case, we observe that one of two subcases results depending upon which of Λ and Ψ is leftmost. If Λ is leftmost, the $LR-region$ doesn't exist and there can be

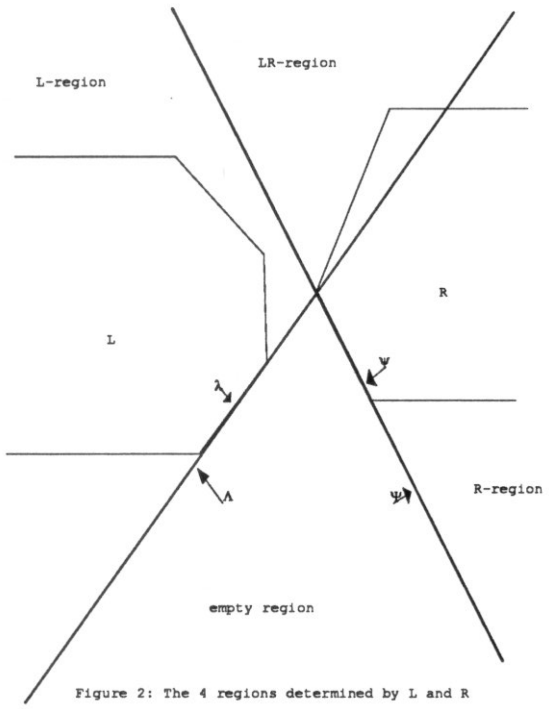

Figure 2: The 4 regions determined by L and R

use the development above to state a series of lemmas and corollaries from which the inner loop of our algorithm follows.

In the lemmas that follow, we will assume that we have confirmed that there is overlap between the vertical extents of L and R, so that an intersection is possible. Each lemma after the first, which produces a witness to the intersection of L and R will be of the form:

Lemma. If <certain conditions are met>, then the problem of intersecting L and R is reduced to the problem of intersecting <two chains named here>.

The two chains named will consist of one of the original chains along with a second chain derived from the other original chain by removing all vertices above or below a given one in a manner to be made precise below. In this statement, the phrase ‘‘the

no intersection. Otherwise, there is an intersection iff the vertical extents of L and R overlap which can be determined in constant time. Therefore, we can assume that Λ and Ψ are not parallel.

problem of intersecting L and R is reduced to the problem of intersecting L' and R'" means that L and R intersect if and only if L' and R' intersect. Furthermore, the horizontal separation of L and R is precisely that of L' and R'. The lemmas we state here follow the development given in [DK]. However, we have made significant changes to correct for degeneracies we observed from our implementation and to allow the generalization to curved regions which follows in the next section.

In these statements, we define **above, below** and **separation** with respect to y-coordinates: For vertices p,q and r, and edge e, p lies **above** (resp. **below**) q if the y-coordinate of p is larger (resp. smaller) than that of q; r **separates** p and q if its y-coordinate lies between the other two y-coordinates; and, p lies below (resp. above) e if it lies below (resp. above) both endpoints of e. In addition, we assume that the LR-region lies above the $\varnothing$-region. Should the converse hold, symmetric results can be easily stated. We also state results in terms of R and its vertices but symmetric results can be stated for L and its vertices.

Finally, we let e^+ (resp. e^-) represent the upper (resp. lower) vertex of edge e. For L a polygonal chain and v a vertex, the part of L above (resp. below) v will comprise the intersection of L with the closed halfplane above (resp. below) the horizontal line through v.

Lemma 2. If λ^+ and ψ^+ both border the LR-region and ψ^+ separates the vertices of λ, then ψ^+ is a witness to the intersection of L and R.

Proof. Since ψ^+ separates the vertices of λ and borders the LR-region, it must lie to the left of some point of λ. By definition, L contains all points to the left of each of its points. Hence, L contains the separating vertex. □

Lemma 3. If ψ^- lies below λ, then the problem of intersecting L and R is reduced to the problem of intersecting L and the part of R above ψ^-.

Proof. If ψ^- borders the $\varnothing$-region, then the portion of R lying below ψ^- is all within the R-region and so cannot intersect L. So, we may assume that ψ^- borders the LR-region.

If L and R intersect below ψ^- (See Figure 3a), the segment joining their intersection point to λ^- must intersect the horizontal ray to the right from ψ^-. By convexity, this point belongs to L. By construction, it belongs to the part of R above ψ^-.

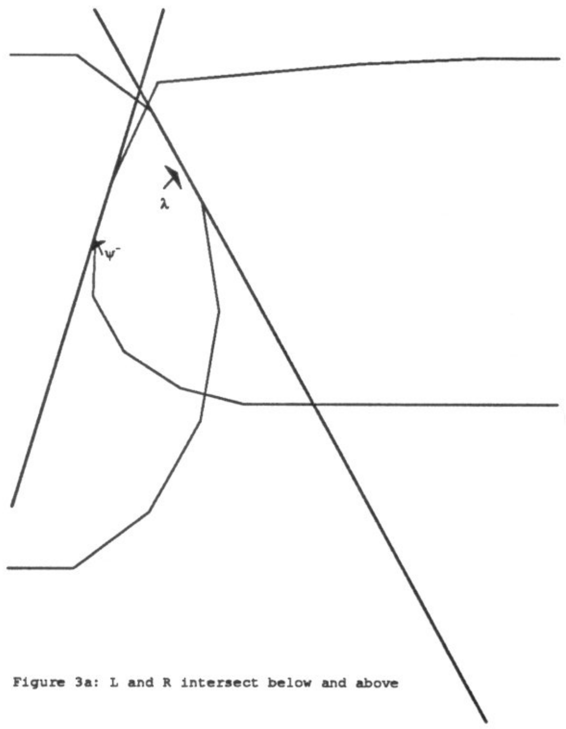

Figure 3a: L and R intersect below and above

So, it only remains to show that if L and R do not intersect their minimum horizontal distance involves a point of R above ψ^-. We assume the reverse and prove a contradiction. Let $w \in L$ and $z \in R$ which lie below ψ^- delimit the minimum horizontal distance between L and R. Now, we define the points a,b,c such that a is the intersection point of $\overline{wz}$ with Ψ, b is the intersection point of the horizontal line through ψ^- with $w\lambda^-$ and c is the intersection point of the line $\overleftrightarrow{w\lambda^-}$ with Ψ (Figure 3b). Note that w, b and c are collinear as are a, ψ^- and c. Furthermore, c lies above the other points and the lines $\overleftrightarrow{wa}$, $\overleftrightarrow{b\psi^-}$ are parallel, and convexity dictates that $b \in L$. By similar triangles, $length(\overline{wz}) > length(\overline{wa}) > length(\overline{b\psi^-})$ yielding our contradiction. □

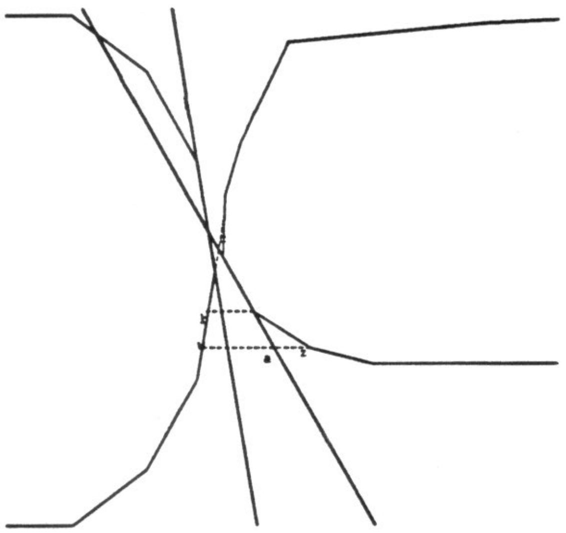

Figure3b Defining a,b,c

As long as both L and R consist of more than 2 edges, this lemma is sufficient to eliminate almost half of one of the chains. In the next subsection, we handle the case where one chain has only 1 or 2 edges. Before doing so, we state two other results which can be used to reduce the complexity of one of the chains. The proofs of these results are similar to the one just given and are not included.

Lemma 4. If ψ^+ lies above λ and λ^+ borders the *LR*-region, then the problem of intersecting L and R is reduced to the problem of intersecting L and the part of R lying below ψ^+.

Lemma 5. If λ^+ borders the *LR*-region, and and λ^- and ψ all border the $\emptyset$-region, then the problem of intersecting *L* and *R* is reduced to the problem of intersecting *L* and the part of *R* lying above ψ^-.

Lemmas 2-5 contribute to the following result:

Lemma 6. If L is a polygonal chain opening infinitely to the left and R is a polygonal chain opening infinitely to the right. Then, $O(\log n)$ iterations suffice either to find a witness to their intersection or to reduce the problem to detecting the intersection (or finding minimum horizontal separation) between two chains one of which has at most 2 edges.

Proof. The proof is by a binary search. At each iteration, we let λ be the middle edge of *L* and ψ the middle edge of *R*. We observe that either the first lemma above provides a witness to the intersection of *L* and *R* or the second removes all edges above (or below) λ or ψ. □

The code embodied by lemma 6 is:

find_int

```
/* implements the inner loop operation */
find_int(p, q, L, l, R, r, pret, rval)
struct polygon *p, *q ;
int *l, *L, *r, *R;
struct point2 *pret;
double *rval;
{
        int lc = (*l+*L)/2;
        int rc = (*r+*R)/2, state ;
        struct point2 *tlp = &p->v[lc], *nlp = &p->v[lc+1]; /* lambda*/
        struct point2 *trp = &q->v[rc], *nrp = &q->v[rc+1]; /* psi */
        double m1 = (tlp->x - nlp->x)/(tlp->y-nlp->y);
        double b1 = tlp->x - m1*tlp->y;
        double m2 = (trp->x - nrp->x)/(trp->y-nrp->y);
        double b2 = trp->x - m2*trp->y;

/* determine relative position of $LR$-region and
        test edge endpoints accordingly */
        if (AEQ(m1,m2))
                if (AEQ(b1,b2) || b1 > b2) {
                        pret[0].y = MINF(tlp->y, trp->y) - 1;
                        state = LR_is_up(p, q, tlp, nlp, trp, nrp,
                                        lc, rc, pret);
```

```
report_intersection(pd,pu,qu,in,porq)
struct point2 *pd, *pu, *qu, *in ;
int porq;
{
/*
 * this is a function which computes intersection points
 * and reports that the iteration is completed
*/
        return (DONE);
}
```

2.3. The finishing touches

Using the lemma above, we will assume that L has been reduced to no more than 3 vertices. We now consider each edge of L separately. The edge of L will be λ in what follows. Now, it remains to resolve the situation when comparing λ to edge ψ of the chain R would only want to subdivide L. Otherwise, we have made progress and can proceed.

This situation arises precisely when λ^- lies below ψ^-. Further, λ^+ must lie on the Ø-region since otherwise λ^+ would have to lie above ψ^+ (by Lemma 4) and ψ^+ would have to border the LR-region (by Lemma 5) and lie above λ^- in which case Lemma 2 asserts that ψ^+ is a witness to the intersection of λ and R. First, we show that reduction is possible in the case where ψ^+ lies above λ (Lemma 8). This leaves only the case where the vertices of ψ separate those of λ (Lemma 9).

Lemma 7. If ψ^+ lies above λ^+, then the problem of intersecting λ and R reduces to that of intersecting λ and the part of R lying below ψ^+.

Proof. The portion of R above ψ^+ can have no horizontal overlap with λ. Hence, none of its points can participate in an intersection and the horizontal distance between any of its points and a point of λ is infinite. □

Lemma 8. If the vertices of ψ separate the vertices of λ, then the problem of intersecting λ and R reduces to that of intersecting λ and the part of R above ψ^-.

report_intersection

```
report_intersection(pd,pu,qu,in,porq)
struct point2 *pd, *pu, *qu, *in ;
int porq;
{
/*
 * this is a function which computes intersection points
 * and reports that the iteration is completed
*/
        return (DONE);
}
```

2.3. The finishing touches

Using the lemma above, we will assume that L has been reduced to no more than 3 vertices. We now consider each edge of L separately. The edge of L will be λ in what follows. Now, it remains to resolve the situation when comparing λ to edge ψ of the chain R would only want to subdivide L. Otherwise, we have made progress and can proceed.

This situation arises precisely when λ^- lies below ψ^-. Further, λ^+ must lie on the $\varnothing$-region since otherwise λ^+ would have to lie above ψ^+ (by Lemma 4) and ψ^+ would have to border the *LR*-region (by Lemma 5) and lie above λ^- in which case Lemma 2 asserts that ψ^+ is a witness to the intersection of λ and R. First, we show that reduction is possible in the case where ψ^+ lies above λ (Lemma 8). This leaves only the case where the vertices of ψ separate those of λ (Lemma 9).

Lemma 7. If ψ^+ lies above λ^+, then the problem of intersecting λ and R reduces to that of intersecting λ and the part of R lying below ψ^+.

Proof. The portion of R above ψ^+ can have no horizontal overlap with λ. Hence, none of its points can participate in an intersection and the horizontal distance between any of its points and a point of λ is infinite. $\square$

Lemma 8. If the vertices of ψ separate the vertices of λ, then the problem of intersecting λ and R reduces to that of intersecting λ and the part of R above ψ^-.

Proof. The portion of R below ψ^- is curving away from λ and hence all of its points have greater horizontal distance from λ than does ψ^-. We know there can be no intersection by the conditions of the lemma which cause all vertices to lie on the $\emptyset$-region. $\square$.

These lemmas clearly make it possible to complete the effort of this section and provide an alternative proof to the following theorem Using the code fragments from above, we now define a variable **FOCUS** which keeps track of whether we are in the situation of this subsection or the previous and incorporate the previous code to yield the final implementation. In what follows, we have removed the code which computes minimum horizontal distance in the interest of clarity.

detect_intersection

```
int FOCUS;

detect_intersection(p,q,ws,m,d)
struct polygon *p, *q ;
struct point2 ws[], *m ;
double *d;
{
        struct polygon pl,pr,ql,qr ;
        struct point2 pint1[2], pint2[2] ;
        double left_right();

        shadow(p, &pl, &pr );
        shadow(q, &ql, &qr );

        if (!horiz_overlap(&pl, 0, pl.nverts-1,
                                &qr, 0, qr.nverts - 1, d, m))
                return(0);

        FOCUS = BOTH;
        *d = left_right(&pl, &qr, pint1, 0, pl.nverts - 1, 0,
                                qr.nverts - 1, m);

        if (*d == 0.0) { /* found an intersection of pl and qr */
                FOCUS = BOTH;
                *d = left_right(&ql, &pr, pint2, 0, ql.nverts - 1, 0,
                                        pr.nverts - 1, m);
                if (*d == 0.0) {
                        seg_int(&pint1[0], &pint2[1],
                                        &pint1[1], &pint2[0], ws);
                        return (1);
                        }
                }
        else    copy_seg(ws,pint1); /* separating line */
        return(0);
 }
```

left_right

```
/* returns 0 if there is an intersection or min distance otherwise */
double
left_right(left,right,ptemp,l,L,r,R,m)
struct polygon *left, *right;
struct point2 *ptemp, *m;
int l,L,r,R;
{
        int lc1,lc2,rc1,rc2, lend, lt, rt, lb, rb ;
        struct point2 sep[2], mp;
        double dis1,dis2,rval,do_int();

/* until one side has been reduced to 2 edges, FOCUS is BOTH */
        while ((((L-l) > 2 || FOCUS == RGHT) &&
                        ((R-r) > 2 || FOCUS == LFT)))
                if (find_int(left, right, &L, &l,
                                &R, &r, ptemp, &rval))
                        return(rval);

        if (FOCUS == BOTH) {
                lend = ((L - l) <= 2) ? TRUE : FALSE ;
                lt = lend ? l+1 : L ;
                rt = lend ? R : r+1 ;
                FOCUS = lend ? RGHT : LFT;

                dis1 = left_right(left, right, ptemp, l, lt, r, rt, m);
                if ((dis1>0.0) && ((lend && (l+1<L)) ||
                                ((!lend) && (r+1<R)))) {
                        dis2 = left_right(left, right, sep,
                                        l+lend, L, r+1-lend, R, m);
                        }
                }

        else    {
/* here we're reduced to only considering left or right */

                lb = (FOCUS == RGHT) ? l : l+1 ;
                lt = (FOCUS == RGHT) ? l+1 : L;
                rb = (FOCUS == RGHT) ? r+1 : r ;
                rt = (FOCUS == RGHT) ? R : r+1;

                dis1 = do_int(left, lb, lt, right, rb, rt,
                                ptemp, m, &lc1, &rc1);
                if ((dis1 > 0.0) &&
                        (((FOCUS == LFT) && (l+1 < L)) ||
                                (((FOCUS ==RGHT) && (r+1 < R))))) {
                        dis2 =do_int(left, l+1, lt, right, rb, rt,
                                        sep, &mp, &lc2, &rc2);
        /* code is eliminated here which decides which distance to use */
                        }
                }
}
```

Theorem 1 [DK]. If L is a polygonal chain opening infinitely to the left and R is a polygonal chain opening infinitely to the right. Then, $O(log\ n)$ operations suffice to either find a witness to their intersection or to find their minimum horizontal separation.

We note that while this result is not new, the proof is. Not only has the proof been reorganized to a simpler form, but we've also stated all lemmas in such a fashion that they can be easily extended to a broader class of problems as we shall see in the remaining sections. This has involved stating lemmas at less than full power in some cases to ease generalization.

3. The generalization to arbitrary convex curves

In this section, we show that the results of the previous section actually hold for a wider of class of objects. In particular, we show that by making small changes to the code presented above, an algorithm for detecting the intersection of convex planar regions results. To do so, it is necessary to subdivide the regional boundaries into curves which serve the role of the polygonal edges above. Then, intersection is detected in a number of operations which grows as the **logarithm** of the number of curves which compose the boundary. An operation now becomes a computation on curves. In what follows, we assume that the user supplies the decomposition of the boundary into curves along with software to perform certain basic operations upon these curves.

In our generalization of the algorithm above, we need to do three computations upon these curves. These computations are as follows:

A. Find the extreme point of the curve in some direction.

B. Evaluate the curve at a point along its trajectory.

C. Intersect two curves of the given type.

For example, algorithms having the flavor given in [SP] would be able to perform any of these computations. With these tools in place, the programs presented in the previous section require only minimal changes in order to detect intersections of convex regions. This has been achieved by carefully modifying the previous presentations of these results. In particular, many of the lemmas given above could be strengthened if we were only detecting polygonal intersections. In their current form, which only reduces the algorithmic running time by a lower order term, they work for all convex objects. The typical situations we have had to account for are those situations where a curve behaves differently than a polygonal straight edge.

While a curve behaves drastically different from a straight edge locally, their behaviors are quite similar when viewed from a distance. We will refer to a curved boundary as a splinegon (in analogy to the term polygon) and will refer to its curved elements as spline segments (rather than line segments). For each spline segment, we consider the line segment connecting its endpoints. This segment intersects the spline segment at the endpoints and nowhere else. From the collection of all such segments, we build a polygon called the *carrier polygon* of the splinegon. The carrier polygon gives a rough description of the splinegon.

Before proceeding, we will set some notation to be used in what follows. We will be working at intersecting the splinegons $S(P)$ and $S(Q)$ which have carrier polygons $L(P)$ and $L(Q)$. Corresponding to the spline edge $S(\lambda)$ of a splinegon will be the edge $L(\lambda)$ of the corresponding carrier polygon. As above, Edges will be represented by lower case Greek letters with their lines of support represented by the corresponding capital letter. For example, Λ will be the line of support of $L(\lambda)$. When the context allows, we will abuse this notation using P to represent $S(P)$ or $L(P)$ as the case may be, λ to represent $S(\lambda)$ or $L(\lambda)$, etc.

In this environment, convexity is used slightly differently. We observe that if $S(\lambda)$ is a spline of $S(P)$, then Λ divides the plane into 2 parts, one of which contains exactly the portion of $S(P)$ bounded by λ and $S(\lambda)$. Furthermore, if ϕ,π,σ are adjacent edges of $L(P)$, we can form a triangle $\hat{\phi}\hat{\sigma}\pi$ which limits $S(\pi)$. We do so, by defining $\hat{\phi}$ (resp. $\hat{\sigma}$) to be the portion of Φ (resp. Σ) lying between its intersections with Π and Σ (resp. Φ and Π).

These observations give a clue to the changes which will occur in generalizing the algorithm of the previous section. Informally, we can reconsider the three steps of the detection algorithm given at the beginning of the previous section: finding left and right shadows; recursing on a left and a right half until one chain is reduced to 2 edges; and finally finishing the work by considering the intersection of a 1 edge polygon with a chain. In our development for splinegons, step 1 must change to account for the possibility that the maximum and minimum y-values might occur along splines rather than at vertices. Indeed, this will typically be the case. The second step remains virtually unchanged. As before, given two edges $S(\lambda)$ and $S(\psi)$, we can subdivide the plane into *L*-, the *R*-, the *LR*- and the $\emptyset$-regions. We do this by using the lines of support Λ and Ψ. However, we are no longer assured that the regions behave according to their names. In particular, the splinegons bounded by $S(\lambda)$ and $L(\lambda)$ and by $S(\psi)$ and $L(\psi)$ lie in inappropriate regions. However, we accounted for this difficulty when stating the lemmas of the previous section, so that the proofs do not change. Finally, new details must be added to the third step.

To make the first step work for the splinegon case, we must first modify the code of section 2.1 to find the points on a splinegon of maximum and minimum *y*-coordinate. To find the maximum *y*-coordinate, we observe that this point occurs on one of the two splines entering the vertex of maximum *y*-coordinate. This vertex is found by our current procedure. We then apply procedure A from above twice to find the maximum point. Similarly, the minimum point is found from the vertex of minimum y-coordinate. Note that finding left and right half splinegons requires actually breaking splines at these maximum and minimum points and adding these two points as vertices of the splinegon. That is, we replace the splinegon by one with new vertices at the points of maximum and minimum y-coordinate and then proceed as we did in the linear case. With this change, Lemma 1 remains true.

The development of section 2.2 changes only slightly. Suppose we have selected the edge λ as a ''middle edge'' of L. Then, we can think of L as being composed of 3 parts, those edges (be they lines or splines) below λ, those above and λ (actually $S(\lambda)$). In this case, S(the edges below λ) behaves as does L(the edges below λ). Similarly for S(the edges above λ). This means that as long as we do not consider $S(\lambda)$ and $S(\psi)$, our definitions of the $L-$, $R-$, $LR-$ and empty regions in section 2.2 apply here also. This being the case, we have now proved the following:

Lemma 9. Lemmas 2 through 5 hold for splinegons as well as for polygons.

We can apply this lemma to prove the following variant of Lemma 6

Lemma 10. If S(L) is a splinegon chain opening infinitely to the left and S(R) is a splinegon chain opening infinitely to the right, then $O(\log n)$ iterations suffice either to find a witness to their intersection or to reduce the problem to detecting the intersection (or finding minimum horizontal separation) between two chains one of which has at most 2 edges.

Indeed the code given in section 2.2 works in this case with no changes.

It remains to rederive the results of section 2.3. We observe that Lemma 7 as stated actually works for the splinegon case. However, a more complex proof of Lemma 8 is needed. In the original proof, we remark

> The portion of R below ψ^- is curving away from λ and hence all of its points have greater horizontal distance from λ than does ψ^-. We know there can be no intersection by the conditions of the lemma which cause all vertices to lie on the $\emptyset$-region.

However, neither of these statements need hold for splinegons. Therefore, we must substitute a more complicated lemma and then prove it, using techniques similar to those used in the proof of Lemma 3:

Lemma 11. If $\lambda^- < \psi^- < \psi^+ < \lambda^+$, then let λ^* represent a point on λ where a line parallel to Ψ supports L. If λ^* lies above ψ^- (resp. below ψ^+), then the problem of intersecting λ and R reduces to that of intersecting λ with the part of R lying above ψ^- (resp. below ψ^+).

Proof. Suppose that $S(L)$ intersects $S(R)$ at a point z lying below ψ^- (resp. above ψ^+). Then $z \in R$-region. But no point of $S(L)$ can belong to the R-region unless λ^* also belongs to the R-region. Consequently, by convexity, both the R-region and $S(L)$ contain the line segment $\overline{\lambda^* z}$. But as λ^* lies above ψ^- (resp. below ψ^+) and z lies below ψ^- (resp. above ψ^+), $\overline{\lambda^* z}$ must intersect the ray originating at ψ^i (resp. ψ^+) and extending infinitely to the right, a ray contained within R. We conclude that the portion of R lying strictly below ψ^- (resp. above ψ^+) cannot completely contain the intersection of λ and R, and thus the portion of R strictly below ψ^- (resp. above ψ^+) can be deleted.

Now suppose that $S(L)$ and $S(R)$ do not intersect and that λ^* lies above ψ^- (resp. below ψ^+). As in the proof of Lemma 3, let the points $w \in S(L)$ and $z \in S(R)$ delimit the minimum horizontal distance between $S(L)$ and $S(R)$. Suppose that w and z lies below ψ^- (resp. above ψ^+). By convexity, $\overline{w\lambda^*} \subset \lambda$, and thus if w were to lie to the right of Ψ, λ and R would intersect. So z lies to the right of Ψ and w lies to the left, and we define the following points:

a is the intersection point of $\overline{wz}$ with Ψ.

b is the intersection point of the horizontal line through ψ^- (resp. ψ^+) with $\overline{w\lambda^*}$.

c is the intersection point of $\overleftrightarrow{w\lambda^*}$ with Ψ.

Then, we observe that c must lie above ψ^- (resp. below ψ^+) since λ^* lies above ψ^- and is the extreme point of $S(L)$ in the direction orthogonal to the line Ψ. Thus, the points w, c and a define a triangle, which contains a similar triangle $\Delta bc\psi^-$ (resp. $\Delta bc\psi^+$). Thus $length(\overline{wz}) > length(\overline{wa}) > length(\overline{b\psi^-})$ (resp. $length(\overline{wz}) > length(\overline{wa}) > length(\overline{b\psi^+})$, which means that ψ^- (resp. ψ^+) is closer to $S(L)$ than z is. Contradiction. $\square$

Unfortunately, lemma 11 depends upon the calculation of the point λ^* lying on a curved edge. Nonetheless, it leads directly to the final lemma and our main result. We assume here that A operations are required to intersect two spline curves, B operations are required to evaluate a spline at a point along its trajectory and C operations are required to find the maximum of a spline in a given direction.

Lemma 12. $O(A + C \log N)$ operations suffice to detect the intersection of a right and a left convex semi-infinite splinegon of at most N vertices each: in the case of intersection, a witness point is reported; in the case of no intersection, a pair of parallel supporting lines is reported which delimits the minimum horizontal distance between the splinegons.

Theorem 2. The intersection of two convex splinegons of at most N vertices can be detected in $O(A+B+C\log N)$ operations.

An example of the algorithm described here is given in Figure 4. The figure is to be read from left to right and down the page. Captions tell which part of which splinegon can be eliminated at each step. The initial squares show the shadowing process. After shadows have been computed, the intersection of P_L and Q_R is considered until it is observed that there is no such intersection.

It is interesting to consider this algorithm relative to published algorithms for intersecting curved regions. Our algorithm benefits from considering only convex regions and so is less general. A simple polygon, however, can be decomposed into convex pieces using a number of standard decomposition algorithms (see e.g. [CD2,FP,Gr,Ke,Sc]). Decomposing a simple splinegon into convex pieces is not so straightforward; a simple splinegon with a single concave curve, for example, can never be decomposed into the union of a finite number of convex pieces. In [DSV], a number of alternate strategies are presented. The algorithm presented here could be applied iteratively to the product of applying an appropriate decompostion scheme to more general regions. Our first step, the shadowing, requires the knowledge of the splinegon only twice, at the top and bottom to find extremal points. Finding these extrema could be done by any technique since this is not an inner loop operation. Our second step has the potential of achieving significant gains in comparison with standard algorithms. It consists of an inner loop which needs no specific knowledge of the splinegon (other than that it is convex). Because of this, no recursion or subdivision of individual curves is necessary as is standard in algorithms for this problem. Finally, our third step could require $O(log\ N)$ steps at which curve operations may be performed. However, theoretical

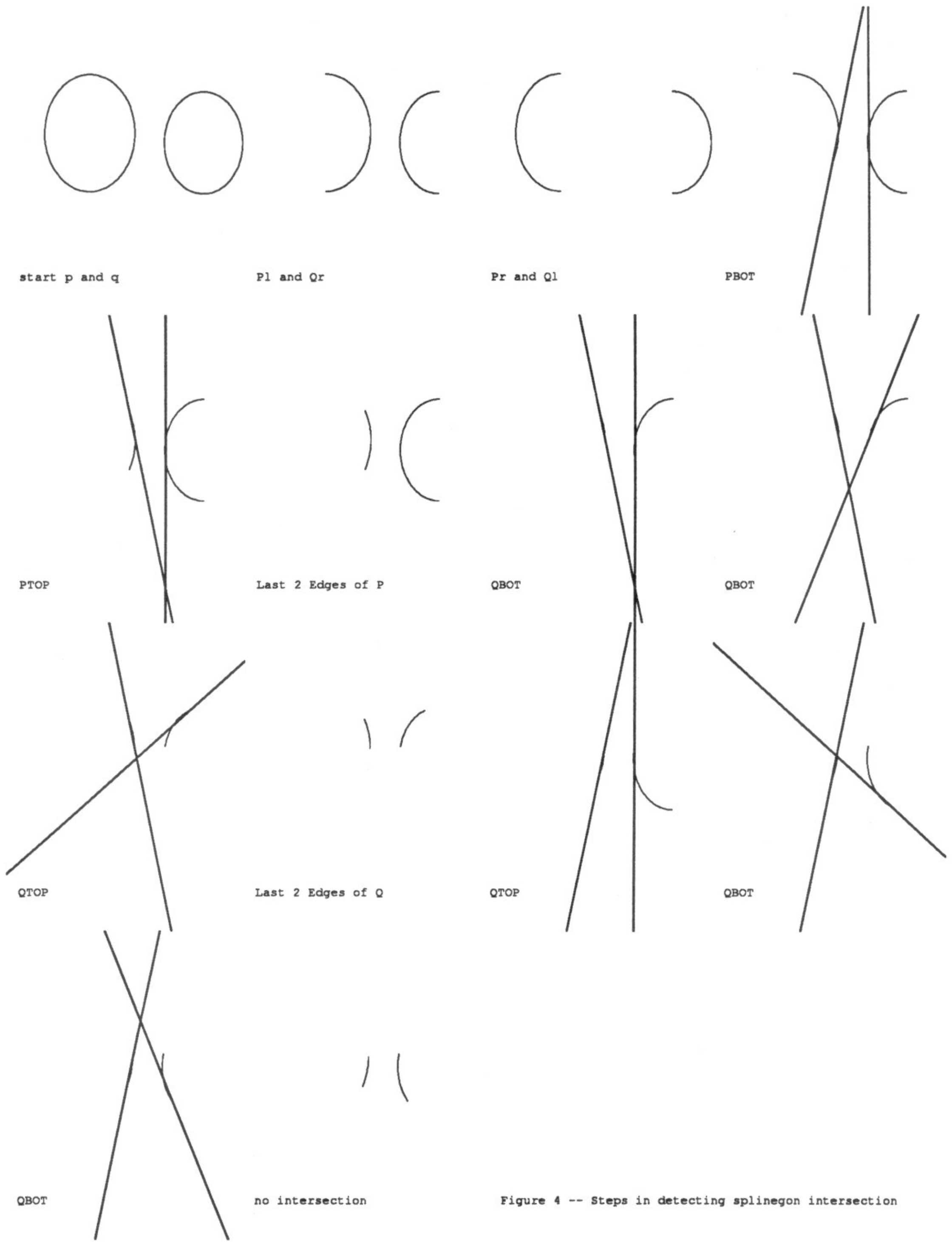

Figure 4 -- Steps in detecting splinegon intersection

results suggest and empirical results confirm that this is seldom the case. The $C \log N$ term in Theorem 2 is actually a loose bound. It should be $C \log d$ where d represents the maximum number of edges of one splinegon which may lie within the vertical span of a single spline on the other splinegon. Although it is possible that $d = \Theta(N^{\varepsilon})$ for some $\varepsilon>0$, in practice, d seems to grow far less quickly.

4. Conclusions

This paper provides the theory and software necessary to determine whether two curved convex planar regions have a common point. In the case they do, a common point is determined. And, in the case that they do not, two points of minimum horizontal distance are reported. The algorithm requires a number of operations which grows asymptotically as the logarithm of the input size. The constant of proportionality here is dependent on the difficulty of computing with the basic curves which make up the pieces of the region boundary.

In addition to this specific result, this paper makes two other contributions. First, we give actual code for the implementation of an algorithm of computational geometry rather than merely asymptotic arguments about complexity. Second, we give an application of a general technique for making algorithms which work on polygons also apply to curved boundaries. We believe that each of these contributions is significant.

Numerous further extensions of our approach are possible. In [DS1, So], the splinegon approach is discussed in far greater detail and techniques are described for generalizing a wide class of algorithms from the linear to the curved case. Included in this class is the extension of the algorithm of Shamos [Sh] for actually computing the intersection of two polygons. Using the results of this paper, first determine if the splinegons intersect. In the case that they do, we have a point interior to their intersection. This point can then be used to order the vertices of each splinegon which allows us to trace the boundary of the intersection in a manner analogous to that of [Sh]. The nature of this algorithm is such that optimized versions of it are likely to perform as well as or better than the standard algorithms for this problem [SP]. And, the class of algorithms to which our approach applies can certainly be enhanced even further.

The implementation described here required the implementation of a number of geometric primitive operations. In addition, portions of a computing environment for geometry had to be created as a debugging tool. Indeed, the code fragments presented here represent less than 20% of the total code written!! So, the existence of libraries of geometric primitives and a good programming environment in which to use them would be of invaluable assistance in future implementations.

5. References

[Br] Brown, C., PADL-2: a technical survey, *IEEE CG&A*, 2, 69-84, 1982

[CD1] Chazelle, Bernard M. and Dobkin, David P., Intersection of convex objects, *JACM*, 34, 1, 1-27, 1987.

[CD2] Chazelle, B., and Dobkin, D., Optimal convex decompositions, *Machine Intelligence and Pattern Recognition 2: Computational Geometry,* G.T. Toussaint, ed., Elsevier Science Publishers, North Holland, 1985, pp. 63-133.

[DK] Dobkin, David P. and Kirkpatrick, David G., Fast detection of polyhedral intersections, *TCS*, 27, 241-253, 1983.

[DS1] Dobkin, David P. and Souvaine, Diane L. , Computational geometry in a curved world, *Algorithmica*, to appear.

[DS2] Dobkin, David P. and Souvaine, Diane L. , Detecting the Intersection of

Convex Objects in the 3 dimensions, in preparation.

[DSV] Dobkin, D., Souvaine, D., and Van Wyk, C., Decomposition and intersection of simple splinegons, *Algorithmica* 3, 1988, 473-486.

[FP] Feng, H. and Pavlidis, T., Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition, *IEEE Transactions on Computing* C-24, 1975, pp.636-50.

[Gr] Greene, D. H., The decomposition of polygons into convex parts, *Advances in Computing Research,* F. Preparata, ed., JAI Press, 1984, pp.235-59.

[GY] Guibas, L. and Yao, F., On translating a set of rectangles, Proceedings of Twelfth ACM STOC, Los Angeles, CA, 1980, 154-160.

[Ke] Keil, J. M., Decomposing a polygon into simpler components, *SIAM Journal of Computing*, 14, 1985, pp. 799-817.

[KS2] Keil, J. M. and Sack, J. R., Minimum decompositions of polygonal objects, *Machine Intelligence and Pattern Recognition 2: Computational Geometry*, G. T. Toussaint, ed., Elsevier Science Publishers, North Holland, 1985, pp. 197-216.

[Mo] Mortenson, M. *Geometric Modeling*, John Wiley & Sons, 1985.

[Sc] Schacter, B. Decomposition of polygons into convex sets, *IEEE Transactions on Computers*, C-27, 1978, pp. 1078-82.

[SP] Sederberg, TW and Parry, SR, Comparison of three curve intersection algorithms, Computer Aided Design. 18, 1, 58-63,1986.

[So] Souvaine, Diane L., Computational geometry in a curved world, Ph.D.

dissertation, Princeton University, 1986.

[Sh] Shamos, Michael I., Geometric complexity, Proceedings of Seventh ACM STOC, Albuquerque, New Mexico, 1975.

[Wh] Whitted, T., An improved illumination model for shaded displays, *CACM*, 23, 6, June, 1980, 343-349.