1987

Convex Hull of Objects Bounded by Algebraic Curves

Chanderjit Bajaj

Myung-Soo Kim

Report Number:

87-697

CONVEX HULL OF OBJECTS BOUNDED BY ALGEBRAIC CURVES

Chanderjit Bajaj
Myung-Soo Kim

CSD-TR-697
December 1987

CONVEX HULL OF OBJECTS BOUNDED BY ALGEBRAIC CURVES[†]

Chanderjit Bajaj and Myung-Soo Kim

Computer Sciences Department
Purdue University

Technical Report CSD-TR-697
CAPO Report CER-87-2
December, 1987

† Research supported in part by NSF grants MIP-8521356, CCR-8619817, and a David Ross Fellowship.

Convex Hull of Objects Bounded by Algebraic Curves[†]

Chanderjit Bajaj and Myung-Soo Kim

Department of Computer Science,
Purdue University,
West Lafayette, IN 47907.

Abstract

We present an $O(n \cdot d^{O(1)})$ algorithm to compute the convex hull of a curved object bounded by $O(n)$ algebraic curve segments of maximum degree d.

1. Introduction

The convex hull computation is a fundamental one in computational geometry. Several linear-time algorithms for computing the convex hull of simple planar polygons are known, McCallum and Avis (1979), Graham and Yao (1983), D.T. Lee (1983), Bhattacharya and El Gindy (1984). These algorithms achieve the more efficient $O(n)$ bound whereas the $\Omega(n \log n)$ lower bound applies to general problem of computing the convex hull of n points in the plane, see Preparata and Shamos (1985). The above algorithms for planar polygons are all vertex-based and it is not straightforward to modify them to deal with curved planar objects with piecewise algebraic boundary curves. Figure 1.1 shows a difference between simple polygons and curved objects, where a single edge may intersect two different pockets. By generalizing Graham and Yao (1983) to an edge-based algorithm Schaffer and Van Wyk (1987) extended these results to a linear-time algorithm for curved objects bounded by piecewise-smooth Jordan curves. Even though this algorithm gives an $O(n \cdot d^{O(1)})$ bound for high order boundary curves with maximal degree d, the practical efficiency is limited to lower degree algebraic curves because of the ubiquitous computation of common tangents of two curved edges. To improve the practical efficiency for high degree algebraic boundary curves, it is necessary to reduce the time consuming curved edge operations. As we will see in §3, the common tangent computation is the most expensive edge operation for both rational and non-rational curves. The philosophy of designing our algorithm is to reduce the number of common tangent computations by detecting only those edges with right orientations. By generalizing the method of D.T. Lee (1983) to a coordinate-based algorithm we suggest an $O(n \cdot d^{O(1)})$ time algorithm to compute the convex hull of objects bounded by algebraic curve segments (possibly non-smooth) with maximum degree d, which form a Jordan boundary curve. To simplify the whole design of algorithm and improve the overall efficiency we segment each boundary curved edge into monotone subsegments in a preprocessing step. Though an expensive computation, $O(d^3 \cdot \log d)$ (resp. $O(d^6 \cdot \log d + T(d))$) time for rational (resp. non-rational) curves on an arithmetic operation model, it greatly simplifies further operations on boundary curve segments, where $T(d)$ is the time required for the curve segment tracing. Our algorithm computes the convex hull in a single pass using a single stack and subdivides the original problem into four subproblems. The convex hull computation by Schaffer and Van Wyk (1987) requires two passes and subdivides the original problem into two subproblems. Souvaine (1986) also suggests a convex hull computation algorithm based on *bounding polygon approach* for a class of curved objects (termed *splinegons*). Figure 1.1 also shows a difference between our algorithm and other algorithms for the curved case. Both Souvaine (1986) and Schaffer and Van Wyk (1987) consider the edge C_j as an *event edge* and apply the time-consuming common tangent operation for such edges. Since edges with such orientation cannot belong to

† Research supported in part by NSF grants MIP-85 21356, CCR-8619817 and a David Ross Fellowship.

the convex hull boundary, these edges are ignored and not considered as event edges in our algorithm. Half-plane containments and/or line-curve segment intersections are sufficient to detect edges with wrong orientations, which cost less than common tangent computations. Further, our algorithm maintains a very simple loop invariant, see the property (∗) in §4.1, which makes the design of a simple algorithm easier.

The rest of the paper is as follows. In §2 we describe the boundary representation for a planar geometric model with algebraic boundary curves. In §3.1 we segment each boundary curved edge into monotone subsegments as a preprocessing step. Crucial too is the internal representation of algebraic curves, i.e., whether they are parametrically or implicitly defined. All algebraic plane curves are implicitly defined by a single polynomial equation $f(x,y) = 0$. A subclass of algebraic curves known as rational curves, have an alternate representation in terms of rational functions, $x = f(t)/h(t)$ and $y = g(t)/h(t)$, with f, g, h being polynomials in t. We present algorithms for both these internal representations. In §4 we present an $O(n \cdot (d^8 \log d + d^2 T(d)))$ (resp. $O(n \cdot (d^{26} \log d + d^2 T(d)))$) algorithm to compute the convex hull of the geometric models of §2 which have all boundary curves as rational (resp. non-rational). Here $T(d)$ is the worst case time taken to trace an algebraic curve segment of degree d, see Bajaj, Hoffmann and Hopcroft (1986). The worst case timing analysis in computing the exponent of the $d^{O(1)}$ is for the general implicitly defined algebraic curves. This exponent is considerably less for curves in their parametric representation. Our model of computation is the arithmetic model where arithmetic operations have unit time cost, see Aho, Hopcroft, and Ullman (1974).

2. Algebraic Boundary Geometric Model

In a general boundary representation, an object with algebraic boundary curves consists of the following:

(1) A face bounded by a single oriented cycle of edges which forms a Jordan curve. (i.e. non-intersecting)

(2) A finite set of directed edges, where each edge is incident to two vertices. Each edge also has a curve equation, represented implicitly and when possible also with parametric equation. Further an interior point is also provided on each edge which helps remove any geometric ambiguity in the representation for high degree algebraic curves, Requicha (1980).

(3) A finite set of vertices usually specified by Cartesian coordinates.

The curve equation for each edge is chosen such that the direction of the normal at each point of the edge is towards the exterior of the object. For a simple point on the curve the normal is defined as the vector of partials to the curve evaluated at that point. For a singular point on the curve we associate a range of normal directions determined by normals to the tangents at the singular point. Further the edges are oriented such that the interior of the object is to the left when the cycle of edges is traversed. Straightforward assumptions are also made, e.g. object has no holes; edges may be singular, however loop-free; different edges do not intersect except at vertices etc.

3. Computations with Algebraic Curves

In this section we consider some primitive geometric operations to manipulate algebraic curve segments. Prior work, of relevance to later sections here, have considered the generation of rational parametric equations for implicitly defined rational algebraic plane curves and surfaces, see Abhyankar and Bajaj (1987a, b, c), the generation of implicit equations for parametrically defined algebraic curves and surfaces, see Bajaj (1987), as well as the robust tracing of algebraic curve segments with the correct connectivity, especially when tracing through complicated singularities, see Bajaj, Hoffmann and Hopcroft (1986). The latter for instance is very useful in determining when a given point lies within a certain implicitly defined algebraic curve segment. In §3.1 we consider how to segment the boundary curve segments into monotone pieces, and in §3.2 we consider operations to compute curve-line segments intersection, halfplane containments, and common supporting lines.

3.1. Monotone Segmentation

In many of the computational geometry problems dealing with curved objects, it is very convenient to design efficient algorithms when we assume each curve segment is monotone (i.e. monotone in x and y coordinates and without any singular or inflection point). We thus first consider how to achieve this monotone segmentation, as a preprocessing step, for a general algebraic curve. This monotone segmentation requires adding singular points, inflection points and extreme points on the curve as extra vertices.

First we take care of singular points on curved edges. Singularities are determined for each curved edge and are computed by using Lemma 3.1.1: I (i) and II (i). The boundary of the object is next modified such that nonsingular edges are either *convex*, *concave* or *linear* segments. Such conditions are easily met by adding extra vertices to inflection points of curved edges. Inflection points of curves can be obtained and the edges are marked *convex*, *concave* or *linear* respectively by using Lemma 3.1.1: I (iv), (v), (vi) and II (iv), (v), (vi). We may also assume edges are further segmented so that extreme points along x or y directions added as vertices. These extreme points are computed by using Lemma 3.1.1: I (ii), (iii) and II (ii), (iii).

Definition 3.1 : Let C be a directed boundary edge without any inflection or singular point. Then
(1) C is *convex* if and only if the gradient of C turns counter–clockwise along C
(2) C is *concave* if and only if the gradient of C turns clockwise along C
(3) C is *monotone* if and only if C is either *convex*, *concave* or *linear*, and the interior of C doesn't include any extreme point along the x or y directions.

Lemma 3.1.1 : (I) Let $C : (a,b) \rightarrow R^2$ be a curve parametrized by $t \in (a,b)$ and $p = C(t) = (c_1(t), c_2(t))$ be a point on this curve. Then
(i) p is a (non-self-intersecting) singular point if and only if $c'_1(t) = c'_2(t) = 0$,
(ii) p is a non-singular x-extreme point if and only if $c'_2(t) = 0$ and $c'_1(t) \neq 0$,
(iii) p is a non-singular y-extreme point if and only if $c'_1(t) = 0$ and $c'_2(t) \neq 0$, and
(iv) p is an inflection point of the curve C if and only if $\kappa(p) = 0 = c'_1(t)\cdot c''_2(t) - c'_2(t)\cdot c''_1(t)$.
If C has no inflection point, then
(v) C is *convex* if and only if $c'_1(t)\cdot c''_2(t) - c'_2(t)\cdot c''_1(t) > 0$, and
(vi) C is *concave* if and only if $c'_1(t)\cdot c''_2(t) - c'_2(t)\cdot c''_1(t) < 0$.
(II) Let C be a curve implicitly defined by $f(x,y) = 0$ and $p = (x,y)$ be a point on the curve C. Then
(i) p is a singular point if and only if $f = f_x = f_y = 0$,
(ii) p is a non-singular x-extreme point if and only if $f = f_y = 0$ and $f_x \neq 0$,
(iii) p is a y-extreme point if and only if $f = f_x = 0$ and $f_y \neq 0$, and
(iv) p is an inflection point if and only if $f_{xx}\cdot(f_y)^2 - 2f_{xy}\cdot(f_x\cdot f_y) + f_{yy}\cdot(f_x)^2 = 0$.
If C has no inflection point, then
(v) C is *convex* if and only if $f_{xx}\cdot(f_y)^2 - 2f_{xy}\cdot(f_x\cdot f_y) + f_{yy}\cdot(f_x)^2 > 0$, and
(vi) C is *concave* if and only if $f_{xx}\cdot(f_y)^2 - 2f_{xy}\cdot(f_x\cdot f_y) + f_{yy}\cdot(f_x)^2 < 0$.

Proof : Most of these results are classical, see for example Walker (1978). □

Lemma 3.1.2 : (I) For a parametric curve segment C of degree d, a monotone segmentation can be obtained in $O(d^3 \cdot \log d)$ time in the worst case and in $O(d^2 \cdot \log^2 d)$ time on the average.
(II) For an implicit algebraic curve segment C of degree d, a monotone segmentation can be

obtained in $O(d^6 \cdot \log d + T(d))$ time in the worst case and in $O(d^4 \cdot \log^2 d + T(d))$ time on the average, where $T(d)$ is the time required for the curve segment tracing.

Proof : (1) The above equations are of degree $O(d)$ in a single variable t, and the squarefree parts can be calculated in $O(d \log^2 d)$ time and be solved using root isolations in $O(d^3 \log d)$ time in the worst case and in $O(d^2 \cdot \log^2 d)$ time on the average, see Schwartz and Sharir (1983).
(2) The above equations are two simultaneous polynomial equations of degree $O(d)$ in two variables x and y. We can eliminate one variable (say, y) using the Sylvester resultant in $O(d^4 \log^3 d)$ time, see Collins (1971), Bajaj and Royappa (1987), and solve the resulting equation of degree $O(d^2)$ in one variable (say, x) in $O(d^6 \log d)$ time in the worst case and $O(d^4 \log^2 d)$ time on the average. Similarly, we can solve for another variable (say, y) with the same time complexity. Finally, the solution points on the curve segment C can be found by tracing the curve segment in $T(d)$ time. □

3.2. Basic Operations on Monotone Curve Segments

In this section, we consider four types of primitive operations on the monotone curve segments. As we will see in §4, these are all the required geometric operations on the monotone curve segments to compute the convex hull of planar curved object once the boundary curves are segmented into monotone pieces. All the other operations are based on the coordinates of the vertices of the monotone curve segments. These primitive operations compute :

(A) The intersection(s) of a monotone curve segment C and a line segment L

(B) The containment of a monotone curve segment C in the upper-left halfplane $H^{UL}(L)$ of a line L

(C) The common supporting line L of a monotone curve segment C and a point q such that $C \cup \{q\} \subset H^{UL}(L)$, and the corresponding supporting point p of L at C

(D) The common supporting line L of two monotone curve segments C and D such that $C \cup D \subset H^{UL}(L)$, and the corresponding supporting points p and q of L at C and D respectively.

Line-Curve Segments Intersection

Suppose C is a monotone curve segment and L is a line segment. Let $R(C)$ and $R(L)$ be the minimal rectangles with sides parallel to coordinate axes and containing C and L respectively, and $T(C)$ be the minimal triangle defined by the line connecting both end points of C and two tangent lines of C at both end points of C. The intersection of C and L can be computed as follows.

```
if ($R(C) \cap R(L) = \emptyset$) then $C \cap L = \emptyset$;
else if ($T(C) \cap L = \emptyset$) then $C \cap L = \emptyset$;
else  Let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be the starting and ending points of $L' = T(C) \cap L$, then
        the intersect point(s) of $C$ and $L'$ can be computed by the following Lemma;
```

Lemma 3.2.1 : (I) If C is a parametric curve segment given by $C(s) = (x(s), y(s))$ with $a \le s \le b$, then C intersects with L' at a point $p = C(s) = t \cdot p_1 + (1-t) \cdot p_2$ if and only if s and t satisfy

$$\begin{cases} a \le s \le b, \text{ and } 0 \le t \le 1 & (1) \\ x(s) = t \cdot x_1 + (1-t) \cdot x_2 & (2) \\ y(s) = t \cdot y_1 + (1-t) \cdot y_2 & (3) \end{cases}$$

(II) If C is an implicit algebraic curve segment given by $f(x, y) = 0$, then C intersects with L' at a point $p = t \cdot p_1 + (1-t) \cdot p_2$ if and only if t satisfies

$$\begin{cases} p \in C \text{ and } 0 \le t \le 1 & (1) \\ f(t \cdot x_1 + (1-t) \cdot x_2, t \cdot y_1 + (1-t) \cdot y_2) = 0 & (2) \end{cases}$$

Proof : Straightforward. □

Lemma 3.2.2 : (I) For a parametric curve segment C, the curve-line segments intersection can be computed in $O(d^3 \log d)$ time.
(II) For an implicit algebraic curve segment C, the curve-line segments intersection can be computed in $O(d^3 \log d + T(d))$ time, where $T(d)$ is the time required for a curve tracing along C.

Proof : (I) The elimination of t can be done in $O(1)$ time resulting in a single polynomial of degree d in a single variable s. This polynomial can be solved in $O(d^3 \log d)$ time using root isolation, see Schwartz and Sharir (1983). There are at most d solutions for s with $a \le s \le b$ and the corresponding t to each s can be solved in $O(1)$ time.
(II) When we expand the equation (2) in an increasing order of t, it gives a polynomial of degree d in a single variable t. The expansion can be done in $O(d^2)$ time and the polynomial can be solved in $O(d^3 \log d)$ time using root isolation. Finally, we need to trace along the curve segment C to check whether these solutions are on the curve segment C in $T(d)$ time. □

Containment in a Halfplane

The halfplane containment for points and line segments can be done in $O(1)$ time. Suppose C is a *convex* monotone edges along which x and y-coordinates are strictly increasing, L is an infinite line with slope $m_. \ge 0$, and $H^{UL}(L)$ is the upper-left closed halfplane of the line L. These are the only types of curved edges and halfplanes considered in §4. Then, $C \subset H^{UL}(L) \Longleftrightarrow$ p_S and $p_E \in H^{UL}(L)$, and $C \cap L' \ne \varnothing$, where $L' = T(C) \cap L$; Hence, the time complexity of halfplane containment testing is the same as that of Line-Curve segments intersection.

Common Supporting Line of a Curve Segment and a Point

Suppose L is a common supporting line of a monotone curve segment C and a point $q = (\alpha, \beta) \notin C$ such that $C \cup \{ q \} \subset H^{UL}(L)$. Then the supporting point p of L at C is given by the following Lemma.

Lemma 3.2.3 : (I) If C is given by a parametric curve $C(t) = (x(t), y(t))$ with $a \le t \le b$, then $p = (x(t), y(t))$ is given by

$$\begin{cases} a \le t \le b & (1) \\ (x(t)-\alpha) \cdot y'(t) - (y(t)-\beta) \cdot x'(t) = 0 & (2) \end{cases}$$

(II) If C is given by an implicit curve $f(x, y) = 0$, then the point $p = (x, y)$ is given by

$$\begin{cases} f(x,y)=0 \text{ and } p=(x,y) \in C & (1) \\ (x-\alpha) \cdot f_x + (y-\beta) \cdot f_y = 0 & (2) \end{cases}$$

Proof : Straightforward. □

Lemma 3.2.4 : (I) For a parametric curve segment C, the common supporting line of C and q can be computed in $O(d^3 \log d)$ time.
(II) For an implicit algebraic curve segment C, the common supporting line of C and q can be computed in $O(d^6 \log d + T(d))$ time, where $T(d)$ is the time required for a curve tracing.

Proof : Similar to Lemma 3.1.2. □

Common Supporting Line of Two Curve Segments

Suppose L is a common supporting line of two disjoint monotone curve segments C and D such that $C \cup D \subset H^{UL}(L)$. Then the supporting points $p = (x, y)$ and $q = (\alpha, \beta)$ of L at C and D respectively are given by the following Lemma.

Lemma 3.2.5 : (I) If C and D are given by parametric curves $C(s) = (x(s), y(s))$ with $a \le s \le b$ and $D(t) = (\alpha(t), \beta(t))$ with $c \le t \le d$, then $p = (x(s), y(s))$ and $q = (\alpha(t), \beta(t))$ are given by

$$\begin{cases} a \le s \le b \ \text{ and } \ c \le t \le d & (1) \\ (x(s)-\alpha(t)) \cdot y'(s) - (y(s)-\beta(t)) \cdot x'(s) = 0 & (2) \\ (x(s)-\alpha(t)) \cdot \beta'(t) - (y(s)-\beta(t)) \cdot \alpha'(t) = 0 & (3) \end{cases}$$

(II) If C is given by a parametric curve $C(s) = (x(s), y(s))$ with $a \le s \le b$ and D is given by an implicit curve $g(\alpha,\beta) = 0$, then $p = (x(s), y(s))$ and $q = (\alpha, \beta)$ are given by

$$\begin{cases} a \le s \le b & (1) \\ g(\alpha,\beta)=0 \text{ and } q=(\alpha,\beta) \in D & (2) \\ (x(s)-\alpha) \cdot y'(s) - (y(s)-\beta) \cdot x'(s) = 0 & (3) \\ (x(s)-\alpha) \cdot g_\alpha + (y(s)-\beta) \cdot g_\beta = 0 & (4) \end{cases}$$

(III) If C and D are given by implicit curves $f(x,y) = 0$ and $g(\alpha,\beta) = 0$, then $p = (x, y)$ and $q = (\alpha, \beta)$ are given by

$$\begin{cases} f(x,y)=0 \text{ and } p=(x,y) \in C & (1) \\ g(\alpha,\beta)=0 \text{ and } q=(\alpha,\beta) \in D & (2) \\ (x-\alpha) \cdot f_x + (y-\beta) \cdot f_y = 0 & (3) \\ (x-\alpha) \cdot g_\alpha + (y-\beta) \cdot g_\beta = 0 & (4) \end{cases}$$

Proof : Straightforward. □

Lemma 3.2.6 : (I) For parametric curve segments C and D, the common supporting line of C and D can be computed in $O(d^6 \log d)$ time.
(II) For a parametric curve segment C and an implicit algebraic curve segment D, the common supporting line of C and D can be computed in $O(d^{12} \log d + T(d))$ time, where $T(d)$ is the time required for a curve tracing.
(III) For implicit algebraic curve segments C and D, the common supporting line of C and D can be computed in $O(d^{24} \log d + T(d))$ time, where $T(d)$ is the time required for curve tracings along C and D.

Proof : Similar to Lemmas 3.1.2 and 3.2.4. □

4. Convex Hull of Geometric Model

In this section we present an algorithm to compute the convex hull of planar curved objects bounded by $O(m)$ monotone curve segments which have been obtained in the preprocessing step from $O(n)$ algebraic curve segments of maximum degree d. In §3 each implicit (resp. parametric) algebraic curve segment of degree d has been segmented into at most $O(d^2)$ (resp. $O(d)$) monotone curve segments by adding extra vertices into singular points, inflection points and extreme points. After this preprocessing step of monotone segmentation, the total number of boundary edges becomes m which is $O(n \cdot d^2)$. The algorithm presented in this section runs in $O(m)$ steps, where each step would take either (a) a constant time if a simple coordinate comparison is enough to process an edge or (b) a polynomial time in the degree d if a nontrivial operation on curve segments such as line-curve intersection or tangent line computation is required to process an edge. Detailed timing analysis will be given in §5. In the following we consider the construction of the lower-right subpart of the convex hull boundary which lies between the bottom most vertex and the rightmost vertex of the original object. The entire convex hull is obtained by applying the same algorithm to the remaining three subparts. Since the minimum horizontal (resp. vertical) line segment containing all the bottommost (resp. rightmost) vertices is on the convex hull boundary, w.l.o.g. we may assume there are unique bottommost and rightmost vertices. In the following, suppose $C_1, C_2, ..., C_M$ is a connected sequence of edges from the bottommost vertex p_0 to the rightmost vertex p_M, and each C_i has p_{i-1} and p_i as its starting and ending vertices. For a point $p_\#$ (resp. $p^\#$) with subscript # (resp. superscript #) we denote the x and y-coordinates of $p_\#$ (resp. $p^\#$) by $x_\#$ and $y_\#$ (resp. $x^\#$ and $y^\#$). We also denote the line segment connecting two points p and q by $L(p,q)$ and the path from p to q along the boundary curve by $\gamma(p,q)$.

4.1. Sequences of Event Edge and Current Hull

We make a constructive definition of a sequence of *event edges* $\{C_{i_k}\}_{k=1}^{N}$ with $C_{i_N} = C_M$ and a sequence of *current hulls* $\{CH_k\}_{k=1}^{N}$. We show the following property of the k-th *current hull* CH_k :

(∗) *If $p \in \gamma(p_0, p_{i_k})$ is a point on the convex hull boundary, then $p \in CH_k$ and the front subarc of CH_k between p_0 and p is on the convex hull boundary.*

This property (∗) and the fact that p_M is the end point of the N-th *current hull* CH_N imply that CH_N is the lower-right subpart of the convex hull boundary between two extreme points p_0 and p_M.

Definition of C_{i_k} and CH_k

Let $i_0 = 0$ and $CH_0 = \{ p_0 \}$. Assume that the index i_k and the k-th *current hull* CH_k $(0 \le k < N)$ have been defined. We define the $(k+1)$-th *event edge* $C_{i_{k+1}}$ and the $(k+1)$-th *event component* $EC_{k+1} \subset C_{i_{k+1}}$ in terms of i_k and CH_k as follows, see Figures 4.1.1–4.1.3.

(A) If $x_{i_k+1} \le x_{i_k}$ and the inner angle of $p_{i_k} < \pi$, then $i_{k+1} = \min \{ j \mid j > i_k \text{ and } x_j > x_k \}$. Further, (a) $EC_{k+1} = C_{i_{k+1}}$ if $y_{i_{k+1}-1} < y_{i_{k+1}}$ and $C_{i_{k+1}}$ is *convex*, and (b) $EC_{k+1} = p_{i_{k+1}}$ otherwise.

(B) If $x_{i_k} < x_{i_k+1}$ and $y_{i_k} < y_{i_k+1}$, then $i_{k+1} = i_k+1$. Further, (a) $EC_{k+1} = C_{i_{k+1}}$ if $C_{i_{k+1}}$ is *convex*, and (b) $EC_{k+1} = p_{i_{k+1}}$ otherwise.

(C) Otherwise, let $ymin_{j-1}(L(p',p'')) = \min \{ y_* \mid p_* \in \gamma(p_{i_k}, p_{j-1}) \cap L(p',p'') \}$ for $i_k < j-1$ (we define the *lid* $L(p',p'')$ later). Further, let $\Omega_k = \{M\} \cup \{ j \mid \text{(i) } i_k+1 < j \le M, \text{(ii) } y_{j-1} < y_j,$ and (iii) either C_j is totally outside of any pocket of CH_k or (C_j is not totally inside of a pocket of CH_k and intersects with a lid $L(p',p'')$ at a point p_{**} with $y_{**} < ymin_{j-1}(L(p',p''))) \}$, and let $j_0 = \min \Omega_k$. (a) If $x_{j_0-1} < x_{j_0}$, then $i_{k+1} = j_0$ and ((a1) $EC_{k+1} = C_{i_{k+1}}$ if $C_{i_{k+1}}$ is *convex*, and (a2) $EC_{k+1} = L(p_{i_{k+1}-1}, p_{i_{k+1}})$ otherwise). (b) Otherwise, $i_{k+1} = j_0 - 1$ and $EC_{k+1} = p_{i_{k+1}}$.

Next we define the $(k+1)$-th *current hull* CH_{k+1}. It is easy to show there is a unique common supporting line L of CH_k and EC_{k+1} at the points p' and p'' (with $x' < x''$ and $y' < y''$) respectively such that $CH_k \cup EC_{k+1} \subset H^{UL}(L)$. If there is more than one choice of p' (resp. p''), we choose p' (resp. p'') so that the distance between p' and p'' to be minimal. Further, let $FRONT\_CH_{k+1}$ denote the front subarc of CH_k between the points p_0 and p', and $REAR\_CH_{k+1}$ denote the rear subarc of EC_{k+1} between the points p'' and $p_{i_{k+1}}$. The $(k+1)$-th *current hull* CH_{k+1} is defined as the connected union $FRONT\_CH_{k+1} \cup L(p',p'') \cup REAR\_CH_{k+1}$, see Figure 4.1.4. CH_{k+1} is a convex arc along which both x and y-coordinates are strictly increasing. $L(p',p'')$ is called as the *lid* determined by p' and p''. Let $\bar{\gamma}$ be the closed path given as $\gamma(p',p'')$ followed by a path from p'' to p' along $L(p',p'')$. If $\bar{\gamma}$ has no self-intersection, the region bounded by $\bar{\gamma}$ is called as the *pocket* determined by the lid $L(p',p'')$. Otherwise, $\gamma(p',p'')$ has an even number of intersections with the lid $L(p',p'')$ counting intersections with multiplicities and $\bar{\gamma}$ divides the plane into finite number of connected regions. The union of all the regions which are to the right of $\bar{\gamma}$ is the *pocket* implied by $L(p',p'')$, see Figure 4.1.5.

Properties of CH_k

The proof of property (*) follows from the following Lemmas 4.1.1-4.1.2.

Lemma 4.1.1 : If a point $p \in C_i$ $(1 \le i \le i_k)$ is on the convex hull boundary, then $p \in CH_k$.

Proof : The interior of the path $\gamma(p_{i_k}, p_{i_{k+1}})$, the arc $C_{i_{k+1}} - EC_{k+1}$, and $(CH_k \cup C_{i_{k+1}}) - CH_{k+1}$ are in the convex hull interior. □

Lemma 4.1.2 : If a point $p \in CH_k$ is on the convex hull boundary, then the front subarc of CH_k between p_0 and p is entirely contained in the convex hull boundary.

Proof : The case $k = 1$ is easy to show. By induction, we assume for k $(1 \le k < N)$ and consider $k+1$. Suppose a point $p \in CH_{k+1}$ is on the convex hull boundary. (a) If $p \in FRONT\_CH_{k+1} \subset CH_k$, then the statement follows by induction. (b) If $p \in L(p',p'')$, then $L(p',p'')$ is also on the convex hull boundary. Further, $FRONT\_CH_{k+1}$ is on the convex hull boundary by induction. (c) If $p \in REAR\_CH_{k+1}$, then there is a supporting line L_p at p. We prove the lid $L(p',p'')$ is on the convex hull boundary. Suppose there is a boundary point q in the region R_1, see Figure 4.1.6. We may assume q is extreme to the outward normal direction of the lid and thus on the convex hull boundary. (i) If $q \in C_j$ $(1 \le j \le i_{k+1})$, then Lemma 4.1.2 implies $q \in CH_{k+1}$. But, it's impossible since CH_{k+1} is convex. (ii) Otherwise, there is a continuous path from $p_{i_{k+1}}$ to q. This path should pass through either the region R_2 or R_3, but both are impossible. Hence, the lid $L(p',p'')$ is on the convex hull boundary, and by induction $FRONT\_CH_{k+1}$ is also on the convex hull boundary. Similarly one can show that the subarc of $REAR\_CH_{k+1}$ between p'' and p is on the convex hull boundary. □

Hence, each *current hull* CH_k has the property (*). Since p_M is the end point of both C_{i_N} and CH_N, Theorem 4.1.1 follows easily from Lemmas 4.1.1-4.1.2.

Theorem 4.1.1 : CH_N is the lower-right part of the convex hull boundary between two extreme points p_0 and p_M.

4.2. Description of Algorithm

We describe an algorithm to compute the sequences of *event edges* $\{C_{i_k}\}_{k=1}^{N}$ and *current hulls* $\{CH_k\}_{k=1}^{N}$ by using a single stack CH. CH contains segments of the k-th *current hull* CH_k which are subarcs of some *convex* edges, some *linear* edges and the lids of pockets. Adjacent elements on the stack share a common end point and the connected sequence of elements on the stack CH generates the *current hull* CH_k (we call as the stack CH *implies* the *current hull* CH_k).

Computing Event Edges

We start with pushing a single point interval $[p_0, p_0]$ into an empty stack CH. The stack CH implies the *current hull* $CH_0 = \{ p_0 \}$. In general, suppose i_k is detected and the stack CH implies the k-th *current hull* CH_k $(0 \le k < N)$. To detect i_{k+1} we call the procedure *Detect_Event_Edge* of Appendix I. Since the correctness of the cases (A) and (B) is obvious, we will consider the case (C) in the following.

An integer variable j initialized to i_k is used to detect i_{k+1}. The initial value of j satisfies the following property.

(**) C_j is not totally contained in an interior of any pocket of CH_k, and TOP(CH) is not strictly below the horizontal line $y = y_j$.

Within the branch (C) there is a while-loop. Upon completion of each loop either i_{k+1} has been detected or the loop-invariant (**) holds for the new j. We show this fact while describing the sub-branches (C1)-(C3) in the following.

At the beginning of each loop, (**) holds and $j < \min \Omega_k$. After we increment j by 1, in (C1) and (C2), $j = \min \Omega_k$ and i_{k+1} and EC are correctly set to appropriate values. Further, in (C3), $j < \Omega_k$. To make the loop-invariant (**) hold we manipulate the stack CH. First of all, we check whether p_j is an interior point of any pocket of CH_k. For this j, p_j is an interior point of a pocket bounded by a lid $L(p',p'')$ if and only if p_j is in the upper left open half plane of $L(p',p'')$ and C_j intersects with $L(p',p'')$. Further, if C_j intersects with $L(p',p'')$, then $L(p',p'')$ is neither strictly above the horizontal line $y = y_{j-1}$ nor strictly below the line $y = y_j$. Since the current stack CH contains all these elements, we need to examine only those elements not strictly below the line $y = y_j$ in a finite step. If the top stack element TOP(CH) lies strictly above the line $y = y_j$, it cannot be a lid bounding a pocket with p_j in its interior. Since it is in the convex hull interior, we pop TOP(CH). We repeat this popping procedure until (a) an intersecting lid is found which bounds a pocket with p_j in its interior, or (b) a non-intersecting element is found which is not strictly above $y = y_j$. In the case of (a), we have one right-to-left cut on the edge C_j. *Right-to-left cut* (resp. *left-to-right cut*) at a point p_* is a transversal intersection of an edge C with a lid at an interior point p_* such that C traverses from the right to the left (resp. from the left to the right) of the lid in the neighborhood of p_*. When the boundary curve transversally intersects at a vertex p_j, we consider the point p_j as a point of C_{j+1} and determine the cut direction in a similar way. Whenever we meet a transversally intersecting edge with a lid, we update the total number of right-to-left cuts and/or left-to-right cuts. In Figure 4.1.7, a path $\gamma(p_*,p_j)$ is totally contained in a self-intersecting pocket and has its first interior intersection with a lid $L(p',p'')$ in a right-to-left direction. When $\gamma(p_*,p_j)$ comes out of a pocket through a point p_{**}, the total cuts in both directions are equal. Hence, we can detect the next edge not totally contained in a pocket by counting the cuts in both directions properly. To detect the next event edge satisfying the condition (C), we skip all the subsequent edges until an edge $C_{j'}$ intersecting with $L(p',p'')$ at a point p_{**} such that the total number of right-to-left cuts and left-to-right cuts upto p_{**} are the same and $y_{**} < ymin_{j-1}(L(p',p''))$. (i) If $y_{j'-1} < y_{j'}$, then $j' = \min \Omega_k$ and i_{k+1} is set to j' correctly. (ii) Otherwise, the loop-invariant (**) holds after the assignment $j = j'$. In (b), the loop-invariant (**) holds. Hence, we proved the following Theorem.

Theorem 4.2.1 : The algorithm *Detect_Event_Edge* detects the $(k+1)$-th *event edge* $C_{i_{k+1}}$.

Computing Current Hulls

Now, we describe how to compute the $(k+1)$-th *current hull* CH_{k+1} from the k-th *current hull* CH_k and the $(k+1)$-th *event component* EC_{k+1} by using the stack CH. Since we are popping some stack elements so that TOP(CH) is neither strictly above nor strictly below the horizontal line $y = MIN$, the stack CH may not *imply* the k-th *current hull* CH_k when EC_{k+1} is computed. However, the stack CH always contains all the elements necessary for the construction of the new stack CH implying the $(k+1)$-th *current hull* CH_{k+1}, i.e. the stack elements not strictly above the horizontal line $y = MIN$. A detailed procedure to update the stack CH to a new stack CH implying CH_{k+1} is given as the procedure *Update_Current_Hull* in Appendix II, where SUPP1(C,D) (resp. SUPP2(C,D)) returns the supporting point p' at C (resp. p'' at D) of the supporting line L such that $C \cup D \subset H^{UL}(L)$.

In each loop we check whether TOP(CH) contains the common supporting point p' of CH_k. We have (α) $p' = p_E$ if and only if $EC \subset H^{UL}(L_{p_E})$ and EC is not strictly below the horizontal line $y = y_E$, (β) $p' \in$ TOP(CH) if and only if $EC \subset H^{UL}(L_{p_S})$, and ($\gamma$) $p' \notin$ TOP(CH) otherwise, where p_S and p_E are the starting and ending points of TOP(CH), and L_{p_S} and L_{p_E} are the tangent lines of TOP(CH) at p_S and p_E respectively. In the cases (α) and (β), the loop terminates after setting p' and p'' appropriately. The correctness of the branches for (α) and (β) is obvious. And, in (γ), we pop the element TOP(CH) from the stack CH and repeat the loop. Since EC is in the upper half plane of the horizontal line $y = y_0$, the loop terminates in a finite step and the correctness of the algorithm follows by induction. Hence, we proved the following Theorem.

Theorem 4.2.2 : The algorithm *Update_Current_Hull* computes the $(k+1)$-th *current hull* CH_{k+1}.

Computing the Convex Hull

Using the above procedures to detect event edges and update current hulls we can design an algorithm to compute the lower-right subpart of the convex hull boundary between the bottommost vertex and the rightmost vertex as follows.

```
procedure Lower_Right_Convex_Hull;
begin
Let $CH = \emptyset$ and $i = 0$;
Push $[p_0, p_0]$ into $CH$;
while ($i < N$) do begin
    Detect_Event_Edge ($CH, i, i, EC$);
    Update_Current_Hull ($CH, EC$); end; (* while *)
end; (* Lower_Right_Convex_Hull *)
```

The correctness of the procedures *Detect_Event_Edge* and *Update_Current_Hull* follows from Theorems 4.2.1–2. Since the above while-loop terminates at a finite step and C_M is an event edge at some N-th step, the correctness of *Lower_Right_Convex_Hull* follows by induction and Theorem 4.1.1.

Theorem 4.2.3 : The algorithm *Lower_Right_Convex_Hull* computes the lower-right subpart of the convex hull boundary between the bottommost vertex and the rightmost vertex.

5. Algorithm Analysis

Complexity of Event Edge Detection

We consider the time complexity required to manipulate the edge segments in the process of detecting next event edges. In the cases of (A) and (B), each edge requires $O(1)$ processing time since a simple coordinate comparison is enough to process an edge. But, in the case of (C), either finite number of simple coordinate comparisons or intersections with a line segment are required to process an edge. The total numbers of coordinate comparisons and curve-line segments intersections are $O(m)$. Since many of curve-line segments intersection testings can be done in a finite number of simple operations using bounding triangle testings, etc., the actual number of curve-line segment intersections based on curve tracing, say m_1, would be much less than m though the worst case complexity can be $O(m)$. Hence, the overall time complexity for event edge detections is $O(m + m_1 \cdot d^3 \log d) = O(m \cdot d^3 \log d) = O(n \cdot d^5 \log d)$. Even though we are also manipulating the stack CH while detecting next event edges, we will consider this cost in the following.

Complexity of Current Hull Computation

We consider the time complexity required to manipulate the stack elements in the process of updating the current hulls. Each stack element is either (1) popped, (2) pushed, or (3) replaced by a subsegment of it. (1) The cost of popping a stack element can be charged to the stack element popped itself, and hence to the event edge which pushed this stack element. Since a stack element can be decided to be popped after an half-plane containment testing to the next event edge, each popping could be done within either (i) a constant time if finite simple operations are sufficient, or (ii) $O(d^3 \cdot \log d + T(d))$ time if a curve-line segment intersection is required, where $T(d)$ is the time required for curve tracing. (2) Each stack element first pushed is either a subsegment of an event edge or a lid common to this event edge. (3) Some stack elements need to be replaced to shorter subsegments. The replaced segment has a common lid with a new stack element which is a subsegment of the new event edge. We charge the cost for pushing and replacing to the new lid, and eventually to the new event edge. A lid is computed within either (iii) a constant time, (iv) $O(d^6 \cdot \log d + T(d))$ time if a tangent line computation from a point to a curve segment is required, or (v) $O(d^{24} \cdot \log d + T(d))$ time if a common tangent line computation between two curve segments is required. Let m_2 be the number of event edges, and $m_{2,1}, m_{2,2}, m_{2,3}, m_{2,4}, m_{2,5}$ be the number of elements of type (i), (ii), (iii), (iv), (v). $m_{2,1}, m_{2,2}, m_{2,3}, m_{2,4}, m_{2,5}$ are $O(m_2)$, and $O(m_2)$ is $O(m)$. Hence, the total cost for manipulating stack elements takes $O(m_{2,1} + m_{2,3} + m_{2,2} \cdot (d^3 \cdot \log d) + m_{2,4} \cdot (d^3 \cdot \log d) + m_{2,5} \cdot (d^6 \cdot \log d)) = O(m_2 \cdot (d^6 \log d)) = O(m \cdot d^6 \log d) = O(n \cdot d^8 \log d)$ (resp. $O(n \cdot (d^{26} \log d + d^2 T(d))))$ for object with all boundary curves as rational (resp. non-rational).

6. Conclusion

In this paper, we suggested an $O(n \cdot d^8 \log d)$ (resp. $O(n \cdot (d^{26} \log d + d^2 T(d))))$ algorithm to compute the convex hull of planar curved object bounded by $O(n)$ rational (resp. non-rational) algebraic curve segments. The true bound on d for the latter can be substantially reduced by use of the multivariant resultant, see Salmon (1885), Macaulay (1902), and Bajaj (1987). An accurate timing analysis on computing the multivariate resultants is currently underway. Though within the same asymptotic time complexity, this improves Schaffer and Van Wyk (1987) to the case of planar curved objects bounded by arbitrary algebraic curve segments. Main differences between this algorithm and Schaffer and Van Wyk (1987) are : (1) the boundary curves are segmented into monotone curve segments by adding inflection points and extreme points as vertices in a preprocessing step, (2) the original problem is divided into 4 subproblems instead of 2 subproblems, (3) the convex hull boundary is computed using a stack in a single pass instead of two pass, (4) it is a coordinated based algorithm instead of edge based algorithm, (5) this algorithm reduces the number of common tangent computation by detecting next event edges with a correct orientation.

7. References

Abhyankar, S., and Bajaj, C., (1987a)
Automatic Rational Parameterization of Curves and Surfaces I: Conics and Conicoids, *Computer Aided Design*, 19, 1, 11-14.

Abhyankar, S., and Bajaj, C., (1987b)
Automatic Rational Parameterization of Curves and Surfaces II: Cubics and Cubicoids, *Computer Aided Design*, 19, 9, 499-502.

Abhyankar, S., and Bajaj, C., (1987c)
Automatic Rational Parameterization of Curves and Surfaces III: Algebraic Plane Curves, Computer Science Technical Report CSD-TR-619, Purdue University.

Abhyankar, S., and Bajaj, C., (1987d)
Automatic Rational Parameterization of Curves and Surfaces IV: Algebraic Space Curves,

Computer Science Technical Report CSD-TR-703, Purdue University.

Aho, A., Hopcroft, J., and Ullman, J., (1974)
The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass.

Bajaj, C., (1987)
Algorithmic Implicitization of Algebraic Curves and Surfaces, Computer Science Technical Report, CSD-TR-681, Purdue University.

Bajaj, C., Hoffmann, C., and Hopcroft, J., (1986)
Tracing Planar Algebraic Curves, Computer Science Technical Report CSD-TR-637, Purdue University.

Bajaj, C., and Royappa, A., (1987)
A Note on an Efficient Implementation of the Sylvester Resultant for Multivariate Polynomials, Computer Science Technical Report CSD-TR-718, Purdue University.

Bhattacharya, B., and El Gindy, H., (1984)
A New Linear Convex Hull Algorithm for Simple Polygons, *IEEE Trans. Inform. Theory*, 30, 1, 85-88.

Collins, G., (1971)
The Calculation of Multivariate Polynomial Resultants, *Journal of the ACM*, 18, 4, 515-532.

Graham, R., and Yao, F., (1983)
Finding the Convex Hull of a Simple Polygon, *Journal of Algorithms*, 4, 324-331.

Lee, D.T., (1983)
On Finding the Convex Hull of a Simple Polygon, *International Journal of Computer and Information Sciences*, 12, 2, 87-98.

Macaulay, F., (1916)
The Algebraic Theory of Modular Systems, *Cambridge Tracts*, No 19, Combridge University Press.

McCallum, D., and Avis, D., (1979)
A Linear Algorithm for Finding the Convex Hull of a Simple Polygon, *Information Processing Letters*, 9, 5, 201-206.

Preparata, F., and Shamos, M., (1985)
Computational Geometry, Springer Verlag, New York.

Requicha, A., (1980)
Representations of Rigid Solid Objects, *Computer Aided Design*, Springer Lecture Notes in Computer Science 89, 2-78.

Salmon, G., (1885)
Lessons Introductory to the Modern Higher Algebra, Chelsea, New York.

Schaffer, A., and Van Wyk C., (1987)
Convex Hulls of Piecewise-Smooth Jordan Curves, *Journal of Algorithms*, 8, 66-94.

Schwartz, J. and Sharir, M., (1983)
On the Piano Movers Problem: II, General Techniques for Computing Topological Properties of Real Algebraic Manifolds, *Adv. Appl. Math.* 4, 298-351.

Souvaine, D., (1986)
Computational Geometry in a Curved World, Computer Science Technical Report CS-TR-094-87, Ph.D Thesis, Princeton University.

Walker, R., (1978)
Algebraic Curves, Springer Verlag, New York.

Appendix I

```
procedure $Detect\_Event\_Edge$ $(CH, i_k, i_{k+1}, EC)$;
begin
(A)  if ($x_{i+1} \le x_i$ and the inner angle of $p_i < \pi$) then begin
          Let $i = \min\{ j \mid j > i+1$ and $x_j > x_i \}$;
          if ($y_{i-1} < y_i$ and $C_i$ is convex) then Let $EC = C_i$; else Let $EC = p_i$; end; (* (A) *)
(B)  else if ($x_i < x_{i+1}$ and $y_i < y_{i+1}$) then begin
          Let $i = i+1$;
          if ($C_i$ is convex) then Let $EC = C_i$; else Let $EC = p_i$; end; (* (B) *)
(C)  else begin
          Let $j = i_k$ and $FOUND$ = false;
          while (not $FOUND$) do begin
               Let $j = j + 1$;
          (C1)  if ($x_{j-1} < x_j$ and $y_{j-1} < y_j$) then begin
                    Let $i_{k+1} = j$ and $FOUND$ = true;
                    if ($C_j$ is convex) then Let $EC = C_{i_{k+1}}$; else Let $EC = L(p_{i_{k+1}-1}, p_{i_{k+1}})$; end; (* (C1) *)
          (C2)  else if ($y_{j-1} < y_j$) then
                    Let $i_{k+1} = j-1$, $EC = p_{i_{k+1}}$ and $FOUND$ = true;
          (C3)  else begin
                    Pop all the stack elements until (a) a lid $L(p', p'')$ which contains $p_j$ in the interior
                    of the pocket bounded by the lid $L(p', p'')$ or (b) a stack element which is not
                    strictly above the horizontal line $y = y_j$;
               (a)  if ($p_j$ is an interior point of the pocket bounded by the lid $L(p', p'')$ which inter-
                    sects with $C_j$ at $p_*$) then begin
                         Let $DONE$ = false, $RightLeftCut = 1$, $LeftRightCut = 1$, and $YMIN = y_*$;
                         repeat
                              Skip all the subsequent edges until an edge $C_{j'}$ which transversally
                              intersects with $L(p', p'')$;
                              for (each transversal intersection point $p_{**}$ on $C_{j'}$) do begin
                                   if (the intersection is a left-to-right cut) then
                                        Let $LeftRightCut = LeftRightCut + 1$;
                                   else Let $RightLeftCut = RightLeftCut + 1$;
                                   if ($y_{**} < YMIN$ and $RightLeftCut = LeftRightCut$) then
                                        Let $DONE$ = true;
                                   else Let $YMIN = \min(YMIN, y_{**})$; end; (* for *)
                         until ($DONE$);
                    (i)   if ($y_{j'-1} < y_{j'}$) then Let $i_{k+1} = j'$ and $FOUND$ = true;
                    (ii)  else Let $j = j'$; end; (* (a) *)
                    end; (* (C3) *)
        end; (* while *) end; (* (C) *)
end; (* $Detect\_Event\_Edge$ *)
```

Appendix II

```
procedure $Update\_Current\_Hull$ $(CH, EC)$;
begin
Let $DONE$ = false;
while (not $DONE$) do begin
      Let $p_S$ and $p_E$ be the starting and ending points of TOP($CH$);
      Let $L_{p_S}$ and $L_{p_E}$ be the tangent lines of TOP($CH$) at $p_S$ and $p_E$ respectively;
(α)   if ($EC \subset H^{UL}(L_{p_E})$ and $EC$ is not strictly below the horizontal line $y = y_E$) then begin
            if ($EC$ is a point $p_{i_{k+1}}$) then Let $p'' = p_{i_{k+1}}$;
            else if ($EC$ is a convex edge $C_{i_{k+1}}$) then Let $p''$ = SUPP2 $(p_E, C_{i_{k+1}})$;
            else if ($EC$ is a line segment $L(p_{i_{k+1}-1}, p_{i_{k+1}})$) then
                  if ($p_{i_{k+1}-1} \in H^{UL}(L(p_E, p_{i_{k+1}}))$) then Let $p'' = p_{i_{k+1}}$; else Let $p'' = p_{i_{k+1}-1}$;
            Push the lid $L(p_E, p'')$ into $CH$;
            if ($p'' \neq p_{i_{k+1}}$) then
                  if ($EC = C_{i_{k+1}}$) then Push the subsegment of $C_{i_{k+1}}$ between $p''$ and $p_{i_{k+1}}$ into $CH$;
                  else if ($EC = L(p_{i_{k+1}-1}, p_{i_{k+1}})$) then Push the lid $L(p_{i_{k+1}-1}, p_{i_{k+1}})$ into $CH$;
            Let DONE = true; end; (* (α) *)
(β)   else if ($EC \subset H^{UL}(L_{p_S})$) then begin
            if ($EC$ is a point $p_{i_{k+1}}$) then Let $p'' = p_{i_{k+1}}$ and $p'$ = SUPP1 (TOP($CH$), $p_{i_{k+1}}$);
            else if ($EC$ is a convex edge $C_{i_{k+1}}$) then
                  Let $p'$ = SUPP1 (TOP($CH$), $C_{i_{k+1}}$) and $p''$ = SUPP2 (TOP($CH$), $C_{i_{k+1}}$);
            else if ($EC$ is a line segment $L(p_{i_{k+1}-1}, p_{i_{k+1}})$) then begin
                  Let $p'$ = SUPP1 (TOP($CH$), $p_{i_{k+1}}$);
                  if ($p_{i_{k+1}-1} \in H^{UL}(L(p', p_{i_{k+1}}))$) then Let $p'' = p_{i_{k+1}}$;
                  else Let $p'' = p_{i_{k+1}-1}$ and $p'$ = SUPP1 (TOP($CH$), $p_{i_{k+1}-1}$); end; (* else if *)
            Let $C$ = the subsegment of TOP($CH$) between $p_S$ and $p'$;
            Pop TOP($CH$) from $CH$, and push $C$ and the lid $L(p', p'')$ into $CH$;
            if ($p'' \neq p_{i_{k+1}}$) then
                  if ($EC = C_{i_{k+1}}$) then Push the subsegment of $C_{i_{k+1}}$ between $p''$ and $p_{i_{k+1}}$ into $CH$;
                  else Push the lid $L(p_{i_{k+1}-1}, p_{i_{k+1}})$ into $CH$;
            Let DONE = true; end; (* (β) *)
(γ)   else Pop TOP($CH$) from the stack $CH$; end; (* while *)
end; (* $Update\_Current\_Hull$ *)
```

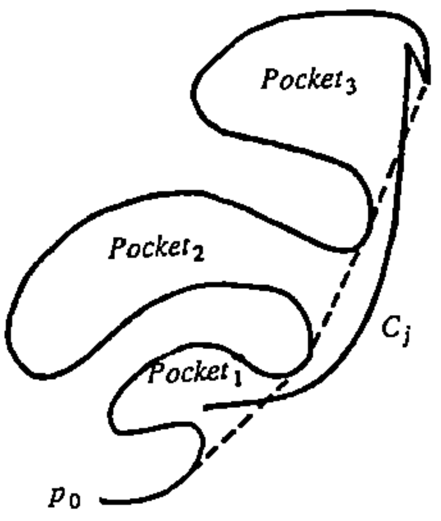

Figure 1.1 Difference between Polygonal and Curved Cases

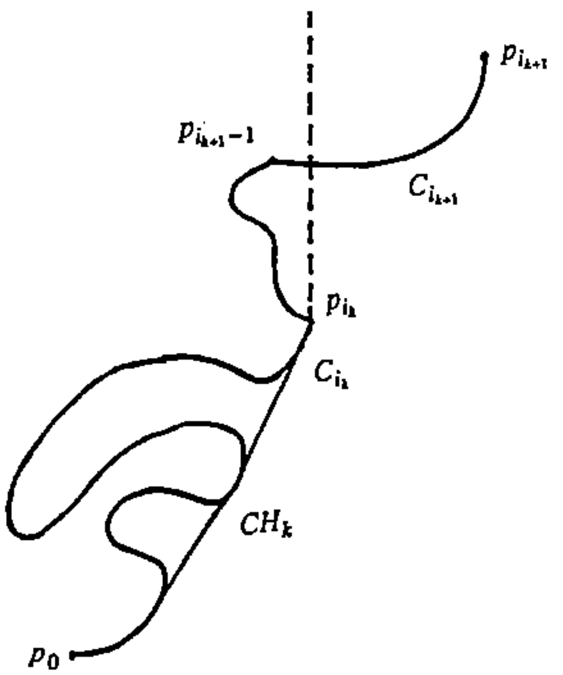

Figure 4.1.1 (a) Event Edge of Type (A) with EC = $C_{i_{k+1}}$

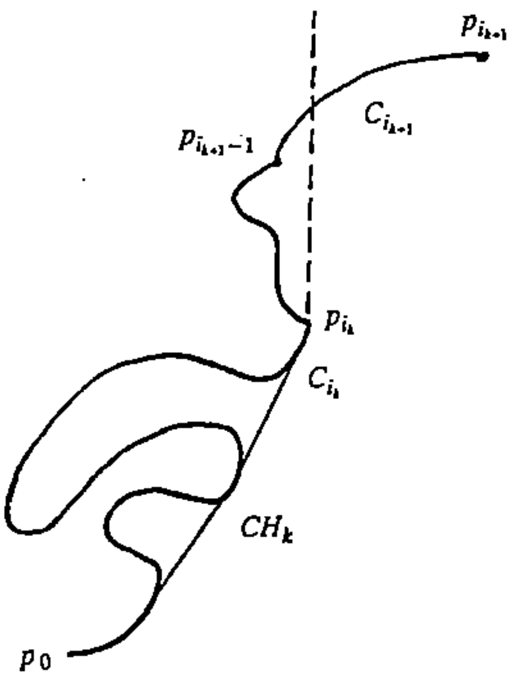

Figure 4.1.1 (b) Event Edge of Type (A) with EC = $p_{i_{k+1}}$

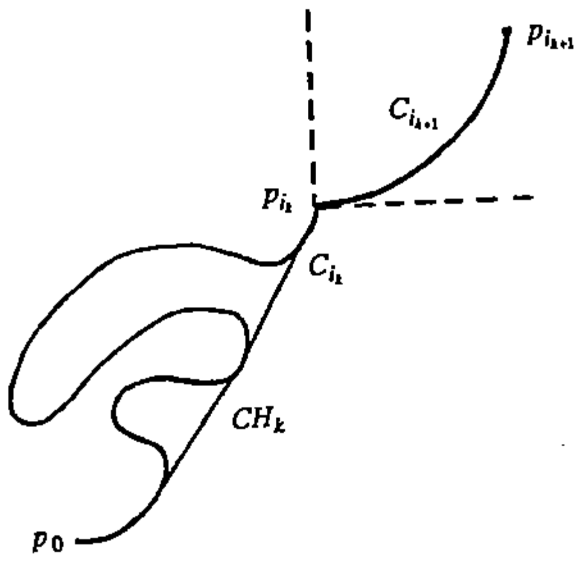

Figure 4.1.2 (a) Event Edge of Type (B) with EC = $C_{i_{k+1}}$

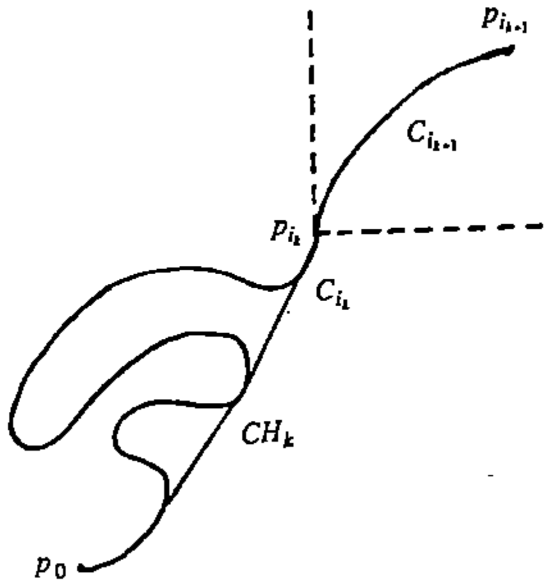

Figure 4.1.2 (b) Event Edge of Type (B) with EC = $p_{i_{k+1}}$

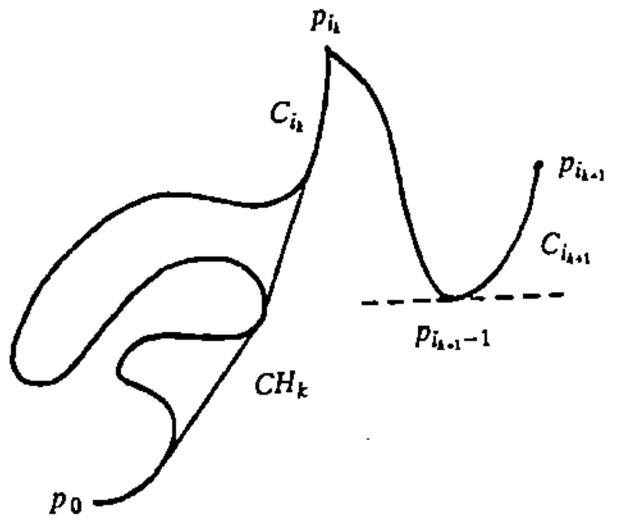

Figure 4.1.3 (a) Event Edge of Type (C) with EC = $C_{i_{k+1}}$

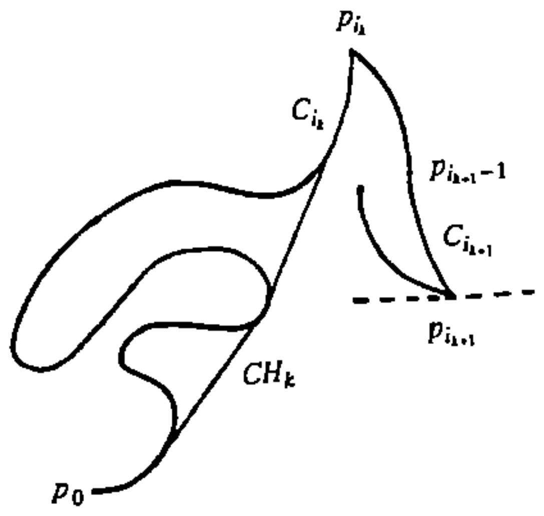

Figure 4.1.3 (b) Event Edge of Type (C) with EC = $p_{i_{k+1}}$

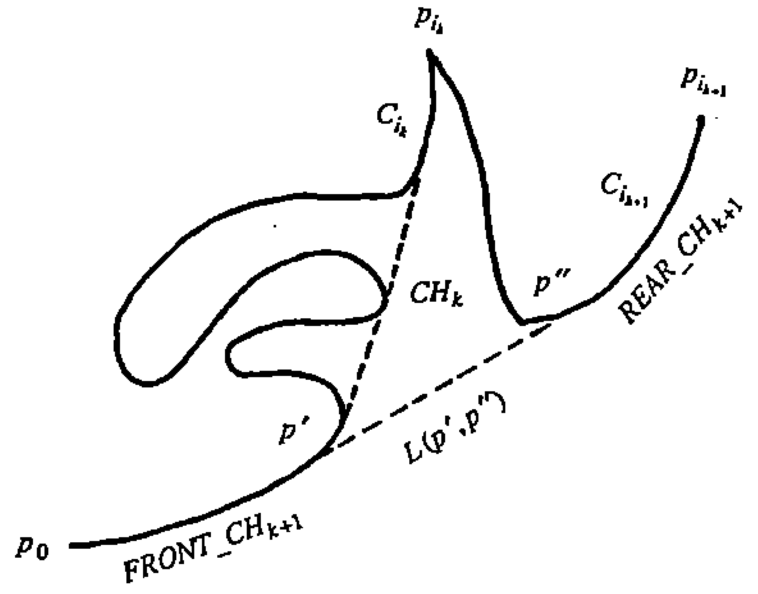

Figure 4.1.4 The $(k+1)$-th *current hull* CH_{k+1}

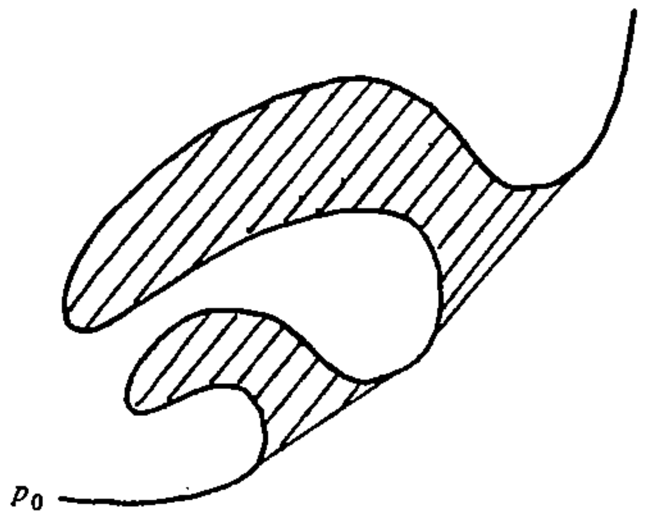

Figure 4.1.5 (a) Simple Pockets

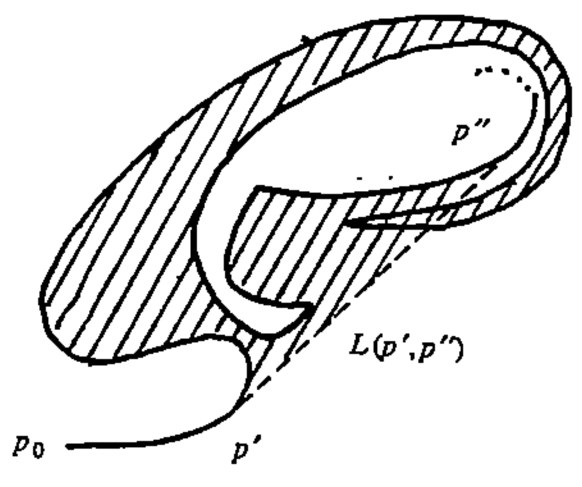

Figure 4.1.5 (b) Self-intersecting Pocket

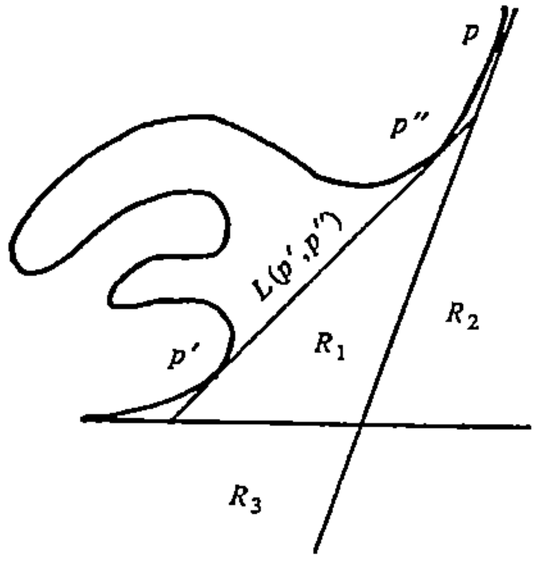

Figure 4.1.6 The Regions R_1, R_2 and R_3

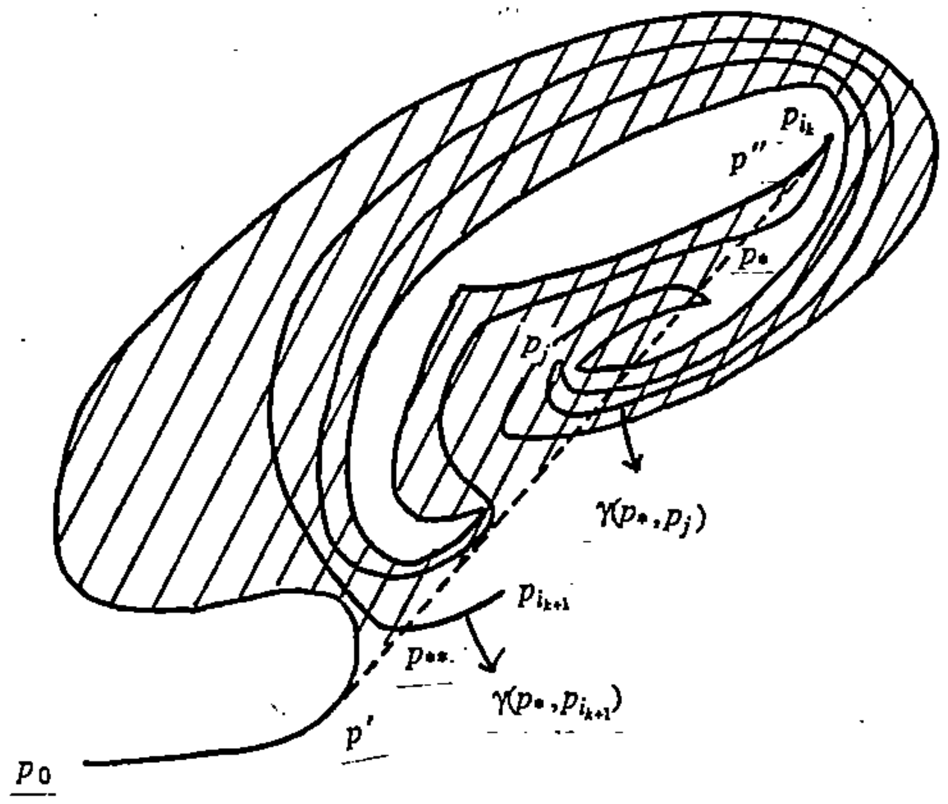

Figure 4.1.7 $\gamma(p_*, p_j)$ in the interior of a pocket