THE POLYGON CONTAINMENT PROBLEM

Bernard Chazelle

ABSTRACT

One feature of pattern recognition is to treat geometric objects as physical entities. This approach, also found in many applications areas, gives rise to a host of problems that essentially involve matching a mobile object against a static one. This article investigates the particular problem of determining whether a given polygon P (with p vertices) can fit into a polygon Q (with q vertices), if translations and/or rotations are allowed. The main result is an $O(pq^2)$ algorithm for solving this problem, for the case where Q is convex. We also give an $O(p + q)$ procedure, valid if only translations are allowed; and we show that the naive algorithm for the general problem requires as much as $O[p^3q^3(p + q) \log(p + q)]$.

Et iterum dico vobis: Facilius est camelum per foramen acus transire, quam divitem intrare in regnum caelorum.

(Matth. 19:24)

Advances in Computing Research, Volume 1, pages 1–33

ISBN: 0-89232-356-6

1. INTRODUCTION

Questions one may ask about the relationships between two polygons fall into two categories. If polygons are regarded as mere sequences of vertices with fixed coordinates in a given system of reference, studying relationships between polygons involves computing their intersection or testing for inclusion of one within the other. All these problems have been well studied [3, 9, 10], and efficient methods for solving them are available. Polygons also can be regarded as coordinate-free objects, defined by the relative position of their vertices or their edges with respect to one another. For example, specifying the angle between two consecutive edges as well as the length of each edge in a traversal of the boundary suffices to define a polygon unambiguously. One can still give a coordinate-dependent description of the vertices, if it is understood that rotating or translating the given set of vertices still gives the same polygon.

This second approach involves giving a physical reality to geometric objects, an approach that is a prime feature of pattern recognition as well as a frequent viewpoint in many applications areas such as graphics, modeling, land planning, or architecture. Typical problems with this setting include determining whether two polygons are similar[1] or whether one can contain the other. The first problem was solved in [6], where an optimal algorithm based on a linear pattern-matching algorithm was proposed. The latter problem is more difficult and, to our knowledge, no satisfying results have as yet been found.

In this article we will investigate the *general containment problem*, which, in its full generality, can be stated as follows:

> *Given two arbitrary, simple[2] polygons P and Q, determine whether Q can contain P, and if yes, report a location for P and Q that realizes containment.*

In other words, the goal is to determine whether it is possible to translate or rotate P so as to make it fit into Q. Here is a summary of our results: In Section 2, we will attack the general problem and will give a naive, $O[p^3q^3(p + q)\log(p + q)]$ algorithm for solving it, with p [resp. q] the number of vertices in P [resp. Q]. In Section 3, we will present an improved $O(pq^2)$ algorithm for the case where the containing polygon Q is convex. It also describes a simple procedure requiring only $O(p + q)$ steps for solving the problem, with now translations allowed only. In Section 4, we will further restrict the generality of the problem by requiring the convex hull of P to be a triangle. Although the complexity of the algorithm that we will present for this particular case is asymptotically of the same order as for the general method, the underlying idea is radically different and makes the algorithm substantially simpler, hence faster. Note that all of our algorithms handle the case where mirror-image transformation of one of the polygon is allowed as well, because it is

reducible to our statement of the problem in linear time. Similarly, they can be used to answer the following question. What is the maximum scaling factor applied to P that will still allow for P to fit into Q? To do so, we proceed by binary search, discriminating on the scaling factor, until the accuracy of the approximation obtained is judged satisfactory.

2. THE GENERAL PROBLEM

Let P [resp. Q] be a simple polygon with vertices $v_1, \ldots, v_p$ [resp. $w_1, \ldots, w_q$], listed in clockwise order. Wlog, we assume that no pair of consecutive edges in P or Q are collinear. For the sake of convenience, we consider Q to be fixed in the plane, whereas P is mobile yet subject to translations and rotations only. We define a *placement* of P as any given position of P relative to Q. A placement can be specified by the location of an edge of P, say v_1v_2, or similarly, as we will do throughout, by the location of v_1 along with the value of the angle $\theta = (v_1X, v_1v_2)$, v_1X being the horizontal line passing through v_1 (Figure 1). A placement is said to be *containing* if it corresponds to a position of Q and P such that P lies inside Q. We also say that a placement has one *contact* if one of two situations occurs: either one vertex of P lies on an edge of Q (contact of *type 1*), or one vertex of Q lies on an edge of P (contact of *type 2*). Because contacts are viewed as pairs (vertex, edge) and are independent of the precise location of the vertex on the edge, two contacts are considered distinct if they differ either in the vertex or in the edge. As a result, a vertex of P that coincides with a vertex of Q gives rise to two contacts. Note that these kinds of contacts are of both types. A placement of P is said to be *stable* if it exhibits three distinct contacts. As it turns out, only stable placements will be of interest when testing

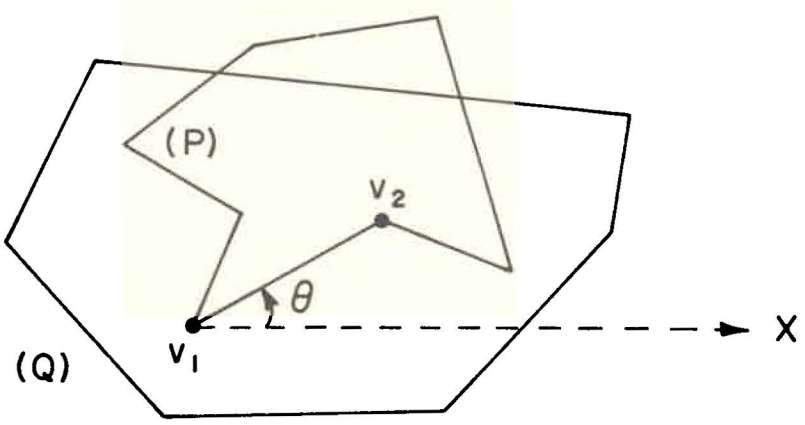

Figure 1. Specifying a placement of P.

for containment. In order to assert this claim, we need to investigate the nature of stable or near-stable placements.

Consider a placement with two contacts of type 1. A and B are the vertices of P involved in the contacts, and aa' and bb' are the two edges of Q supporting A and B, respectively. We define an origin O as the intersection of the two infinite lines passing through aa' and bb', and we let C be an arbitrary point in the polygon P. We choose a system of X- and Y-axes so that $\alpha = (OX, Ob) = \pi - (OX, Oa)$. Finally, we introduce the angle $\beta = (AB, BC)$ (Figure 2). We wish to study the motion of C as A and B *slide* along Oa and Ob, respectively. Initially we assume that AB is parallel to the X-axis; but, in general, this segment forms a nonnull angle θ with the X-axis [$\theta = (AX, AB)$] (Figure 2). Using the well-known sine relation, we can easily find the location of C. We have

$$\sin(AO, AB)/|OB| = \sin(2\alpha)/|AB|$$

and because $(AO, AB) = (\alpha + \theta)$, it follows that

$$|OB| = |AB| \sin(\alpha + \theta)/\sin(2\alpha) \tag{1}$$

Letting x_C [resp. y_C] denote the X- (resp. Y-) coordinate of the point C, we derive from this relation

$$\begin{aligned} x_C &= |AB| \sin(\alpha + \theta)/2 \sin(\alpha) + |BC| \cos(\beta + \theta) \\ y_C &= |AB| \sin(\alpha + \theta)/2 \cos(\alpha) + |BC| \sin(\beta + \theta) \end{aligned} \tag{2}$$

For the sake of clarity, we introduce the set F of functions of the form

$$f(\theta) = \lambda \sin(\theta + \gamma) + \mu$$

defined on $[0, 2\pi]$ for any reals λ, γ, and μ. Note that $\mathbf{F}$ is a vector space over the field of real numbers. An important property of the functions in $\mathbf{F}$ is that (1) they are continuous, and (2) on any given interval of size less than π, they take on any value at most twice. Note that the functions

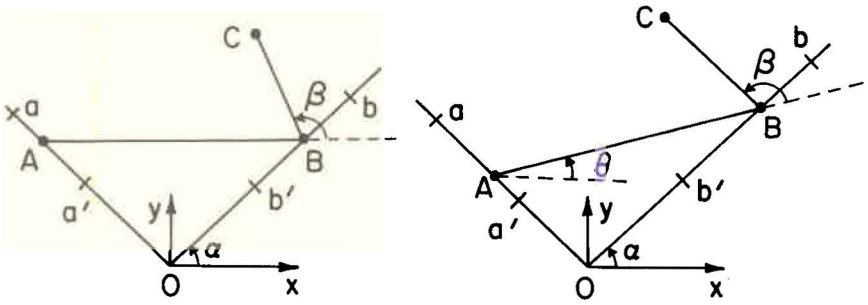

Figure 2. Moving a placement with two contacts.

$x_C(\theta)$ and $y_C(\theta)$ belong to **F**. We next study the behavior of the vertices A and B. It is easy to show that, like (1)

$$| OA | = | AB | \sin(\alpha - \theta)/\sin(2\alpha)$$

Because θ varies between $-\alpha$ and $+\alpha$, and $0 < \alpha < \pi/2$, the constraint $| Oa' | \leq | OA | \leq | Oa |$ restricts the range of feasible values for θ to at most two intervals. Relation (1) shows that the same applies to $| OB |$, which ultimately limits θ to taking on values on three possible disjoint intervals in $[-\alpha, \alpha]$.

It is now easy to test all the possible situations in which C has a contact of type 1, while both A and B have contacts of type 1 on aa' and bb', respectively. To do so, we observe that the locus of C is an ellipse. Therefore, we can compute its intersection with all the edges of Q and keep only the intersection points that correspond to feasible values of θ. Note that to compute the intersection of a line with the locus of C, we have to solve an equation of the form:

$$\lambda x_C(\theta) + \mu y_C(\theta) = \nu \tag{3}$$

Since **F** is a vector space and both $x_C(\theta)$ and $y_C(\theta)$ are functions in **F**, as mentioned earlier, the left-hand side is a function in **F**. Therefore, Equation (3) reduces to

$$\lambda' \sin(\theta + \gamma) = \mu'$$

which we assume to be solvable in constant time. Throughout, our model of computation will be a RAM with infinite arithmetic precision and unit time trigonometric functions. We also observe that the intervals of feasibility for θ do not span a range of size greater than π; therefore, as previously noted, the equation has at most two feasible solutions. We can conclude:

LEMMA 1. *Given any vertices A, B, C of P and any edges a, b, c of Q, there are at most two stable placements with contacts of type 1 involving A on a, B on b, and C on c. Moreover, these placements can be computed in constant time.*

Next, we proceed to generalize this result to any kind of stable placements. Note that because P and Q play symmetrical roles, the only case that remains to be studied involves stable placements with two contacts of type 1 (A on aa', B on bb') and one contact of type 2 (vertex w of Q on edge CC' of P). Let D be the intersection of the vertical line passing through w with $line(CC')$[3] (Figure 3). It is clear that there is a contact of type 2 between w and CC' if and only if D lies on CC' and coincides with

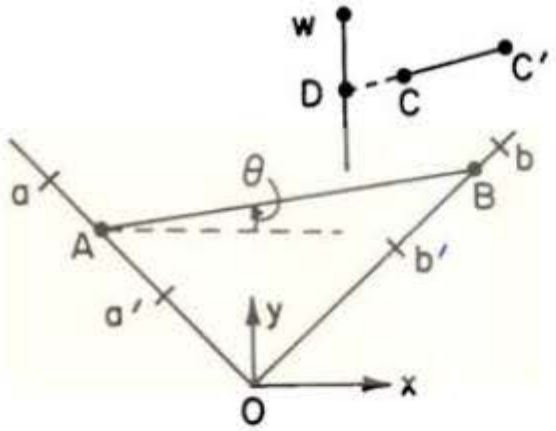

Figure 3. Testing for stable placements with one contact of type 2.

w. We can now generalize our previous result.

LEMMA 2. *Given any triple of vertices and any triple of edges in P and Q, any stable placement involving these edges and vertices can be detected and computed in constant time.*

We must now show the utility of these arguments by establishing the following.

LEMMA 3. *If Q can contain P, there exists a stable containing placement.*

PROOF. Starting from a containing placement (i.e., for which Q contains P), we can move P by a vertical translation until it first hits Q. At

this point, we have at least one contact of type 1 or 2. If there are three of them, the claim is true. If there are only two, and two vertices happen to coincide, we can pivot P around these vertices until P hits Q for a second time; at this stage the placement is necessarily stable. In the general case, the first translation creates only one contact. Whatever its type, there is always one edge involved, so we can translate P along this edge until P hits Q again, which will create a second contact. If both contacts are of the same type, we are in the case of Figure 2 (exchanging P and Q, if the contacts are of type 2). Then we move A and B along aa' and bb', respectively, until either A or B hits an end point of their respective edge aa' or bb', or a third point C hits the boundary of Q, whichever happens first. In all cases, a third contact is thus created. The last possibility to investigate corresponds to the two contacts created so far being of distinct types. Let aa' and A be, respectively, the edge of Q and the vertex of P involved in the contact of type 1. Similarly, bb' and B denote the edges of P and the vertex of Q forming the contact of type 2 (Figure 4). A divides the edge aa' into two segments, one of which, say $u = Aa$, wlog, has the property that all of its points M satisfy $|MB| \geq |AB|$. Similarly, the segment $v = bB$ can be assumed to share the same property with respect to A. It then follows that we can have A move along u toward vertex a in a continuous motion, while B slides similarly along v. This motion will proceed until either A reaches vertex a or B reaches b, or until P hits Q anywhere else, whichever happens first. In all cases, the motion terminates with the addition of a third contact to the placement, which completes the proof. □

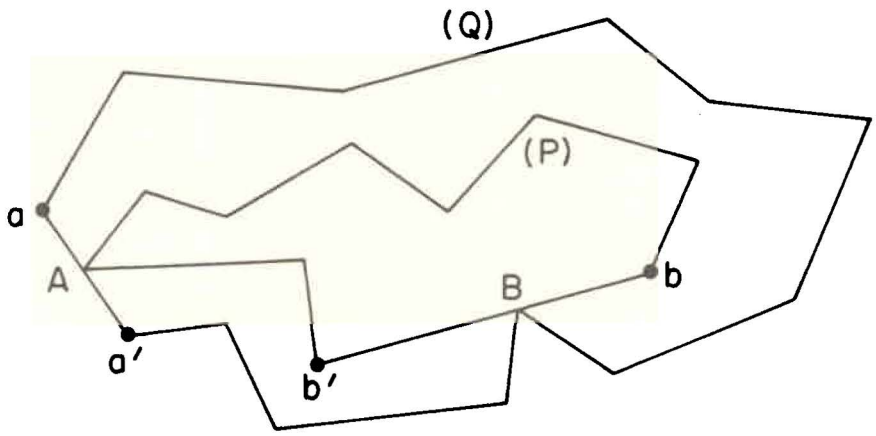

Figure 4. Dealing with contacts of different types.

We are now in a position to propose a naive algorithm for solving the general containment problem:

Enumerate all possible stable placements, and for each of them, determine whether Q contains P.

For each feasible stable placement, the detection procedure just described provides the location, with respect to Q, of at least three distinct points. So, it is straightforward to obtain the coordinates of all vertices of P for this particular placement. With this information available, we can next test whether P lies inside Q, using any standard polygon-inclusion algorithm [1, 9]. These methods usually require sorting the vertices of P and Q beforehand, which contributes to the $O(q + p \log p)$ running time of each iteration. Since there are $O(p^3q^3)$ candidates for stable placements, we conclude

THEOREM 4. *For any pair of simple polygons with, respectively, p and q vertices, it is possible to solve the general polygon containment problem in* $O[p^3q^3(p + q) \log(p + q)]$ *steps.*

3. THE CONVEX CASE

We will see in this section how the introduction of a convexity requirement leads to significant improvements in the containment algorithm. From now on, we will assume that the polygon Q is convex. If P has a containing placement, Q certainly also contains the convex hull of P. So we will first compute this convex hull, using any linear time algorithm [7], which then permits us to deal exclusively with convex objects. We can state the new version of our problem as follows:

Given two convex polygons P and Q, solve the general containment problem for P and Q.

Once again, we assume that Q is fixed and P mobile. We choose to specify a placement of P with the XY-coordinates of one of its points, say v_1, and the direction of v_1v_2, measured by the angle $\theta = (v_1X, v_1v_2)$.

3.1 The Translation-Restricted Problem

Mostly in order to introduce our notation and present some of the techniques used later on, we will begin with a simplified version of the problem. We require here that P be moved only by translation. For any fixed value of θ, there is a very convenient way of characterizing containing placements.

Let $H_i (i = 1, \ldots, q)$ be the half-plane that is delimited by $line(w_i w_{i+1})$ (indices taken mod q), and contains Q. For any position of P in H_i, there is in general one vertex (c_i), or in any case at most two (consecutive) vertices of P that are closest to $line(w_i w_{i+1})$. Since P and Q are convex, it is easy to see that the line passing through c_i and parallel to $w_i w_{i+1}$ *rolls* around P, as w_i progresses in clockwise order around Q. As a result, $\{c_1, \ldots, c_q\}$ is a subsequence of $\{v_1, \ldots, v_p\}$, and it is thus straightforward to compute the c_i's in $O(p + q)$ time. We will omit the details.

We will now move the polygon P by translation so that c_i coincides with w_i. This completely specifies a placement of P, which enables me to define the infinite line $t_i(\theta)$ passing through v_1 and parallel to $w_i w_{i+1}$. Finally we let $ht_i(\theta)$ denote the half-plane defined by $t_i(\theta)$ that does not contain the edge $w_i w_{i+1}$. Note that P lies entirely on the same side of $line(w_i w_{i+1})$ as Q if and only if v_1 lies in $ht_i(\theta)$. Therefore, we know that for any given value of θ, Q can contain P if and only if the intersection $I(\theta)$ of the half-planes $ht_1(\theta), \ldots, ht_q(\theta)$ is not empty. Moreover if the intersection is indeed not empty, it is a convex polygon that is exactly the locus of v_1 for all containing placements with direction θ (Figure 5). We can use this fact to derive an algorithm for solving our restricted problem. It simply consists of computing all the half-planes $ht_1, \ldots, ht_q$, which requires $O(q \log p)$ time; then determining their intersection (using Shamos and Hoey's algorithm [10], for example), which adds $O(q \log q)$ to the running time. This results in an $O[p + q \log(p + q)]$ time algorithm for computing all containing placements of P obtainable by translation only. We will show in Section 3.2.2 that, actually, the intersection of half-planes can be computed in linear time, leading to:

THEOREM 5. *For any pair of convex polygons P and Q, with, respectively, p and q vertices, it is possible to find all the containing placements of P obtainable by translation only, in time and space* $O(p + q)$.

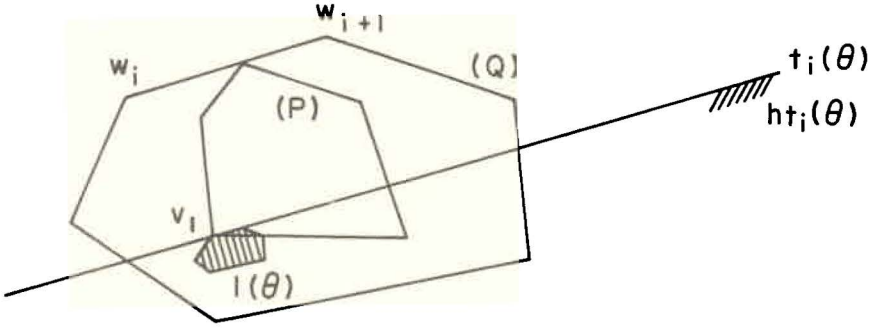

Figure 5. The locus of the vertex v_1.

3.2 The Unrestricted Problem

3.2.1 *Introduction*

Turning back to our original problem, i.e., allowing both translations and rotations in the motion, we are tempted to carry out the previous procedure for all values of θ, or since it is obviously impossible to do so, at least apply the previous technique for all "significant" values of θ. To give substance to this idea, we must first investigate the transformation that the intersection $I(\theta)$ undergoes as θ varies from 0 to 2π. The lines $t_i(\theta)$ are translated in a continuous motion, so the only angles θ that need to be considered are those for which three lines $t_i(\theta)$ intersect at the same point. Note that these values of θ correspond exactly to the stable placements of P. If our goal is to improve upon the naive algorithm, however, we cannot compute all possible candidates for containing placement, and we must show that it is indeed possible to avoid the exhaustive enumeration.

To that end, we will give a more convenient characterization of stable placements. We use the concept of duality, found in [2, 8], which consists of reducing geometric problems to others by means of mathematical transforms. In this case, we borrow from K. Brown's half-space algorithm [2], which proceeds by reduction to a convex hull problem.

We start out by partitioning the set of edges of Q into two parts: one including the upper edges of Q and the other the lower edges. Wlog, we can assume that no edge of Q is parallel to the Y-axis. Then it is possible in linear time to determine the unique vertex of Q with the minimum X-coordinate, say w_1. Similarly we obtain the vertex w_k with the maximum X-coordinate, and we partition the edges of Q into two subsets $\{w_1w_2, \ldots, w_{k-1}w_k\}$ and $\{w_kw_{k+1}, \ldots, w_qw_1\}$. Now $I(\theta)$ can be viewed as the intersection of two unbounded convex regions (Figure 6)

$$UP(\theta) = ht_1(\theta) \cap \ldots \cap ht_{k-1}(\theta)$$

and

$$LO(\theta) = ht_k(\theta) \cap \ldots \cap ht_q(\theta)$$

Since none of the lines $t_i(\theta)$ is vertical, each can be represented by an equation of the kind:

$$Y = a_i(\theta)X + b_i(\theta)$$

So we have a one-to-one mapping

$$t_i(\theta) \leftrightarrow U_i(\theta) = [a_i(\theta), b_i(\theta)]$$

Note that $a_i(\theta) = (y_{w_{i+1}} - y_{w_i})/(x_{w_{i+1}} - x_{w_i})$, and is thus independent of θ; so from now on, we will refer to it as simply a_i.

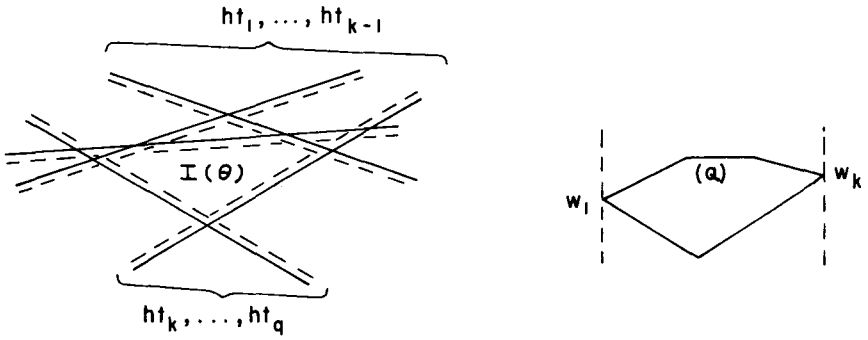

Figure 6. The partition of the half-planes defining $I(\theta)$.

The boundary of $UP(\theta)$ consists of several edges, one or two of which are semiinfinite lines. The lines $t_i(\theta)$ that do not contribute an edge to $UP(\theta)$ are called redundant. It is shown in [2] that a line $t_r(\theta)$ is redundant if and only if there exist two other lines $t_i(\theta)$ and $t_j(\theta)$, such that their intersection lies below $t_r(\theta)$, and the slope of $t_r(\theta)$ lies between the slope of $t_i(\theta)$ and that of $t_j(\theta)$, that is, such that (wlog)

$$a_i < a_r < a_j$$

and

$$[b_r(\theta) - b_i(\theta)]/(a_r - a_i) > [b_j(\theta) - b_i(\theta)]/(a_j - a_i)$$

It follows that the point $U_r(\theta)$ lies above the segment $U_i(\theta)U_j(\theta)$ (Figure 7).

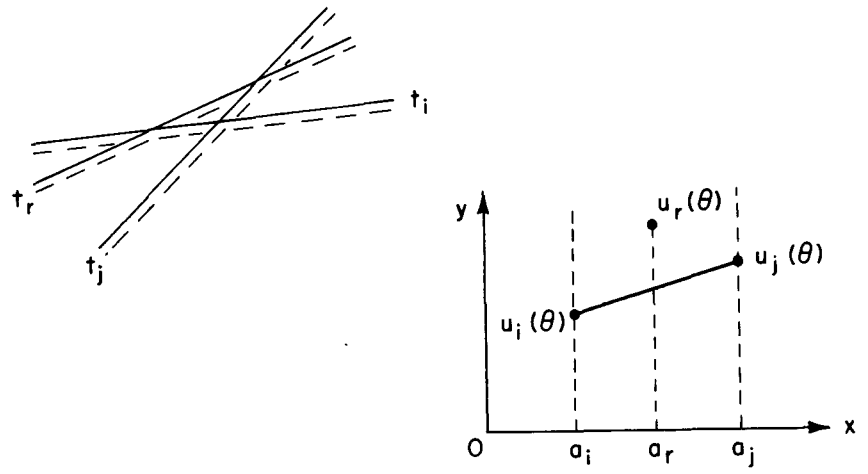

Figure 7. The notion of redundancy.

From this fact, we can prove that all the nonredundant lines for $UP(\theta)$ are exactly those corresponding to the vertices $U_i(\theta)$ of the bottom part of the convex hull of $\{U_1(\theta), \ldots, U_{k-1}(\theta)\}$.[4] Note that since Q is convex, the slope of w_iw_{i+1} decreases as i varies from 1 to $k-1$; therefore, the points $U_i(\theta)$, for $i = k-1, \ldots, 1$, appear in this order with increasing X-coordinates. Let $CU(\theta)$ be the bottom part of the convex hull of $\{U_1(\theta), \ldots, U_{k-1}(\theta)\}$. We can apply the same reasoning to $LO(\theta)$, which leads us to define $CL(\theta)$ as the top part of the convex hull of $\{U_k(\theta), \ldots, U_q(\theta)\}$ (Figure 8). If for $\theta = 0$, there exists a containing placement of P, the algorithm of Theorem 5 can be used to detect it. If this is not the case, difficulties arise, and a more thorough analysis is needed.

Assume that for some value of θ, exactly one vertex α of $UP(\theta)$ lies on an edge $\beta\gamma$ of $LO(\theta)$, and all of $UP(\theta)$ lies below the line t_k (Figure 9A). Using the notation of Figure 9, with the correspondence between lines t_x and points M_x, we can show that the dual figures $CU(\theta)$ and $CL(\theta)$ share the same property. More precisely, we will prove that

LEMMA 6. *The statements "α lies on $\beta\gamma$ and $UP(\theta)$ lies below $LO(\theta)$" and "M_k lies on M_iM_j and $CL(\theta)$ lies below $CU(\theta)$" are equivalent.*

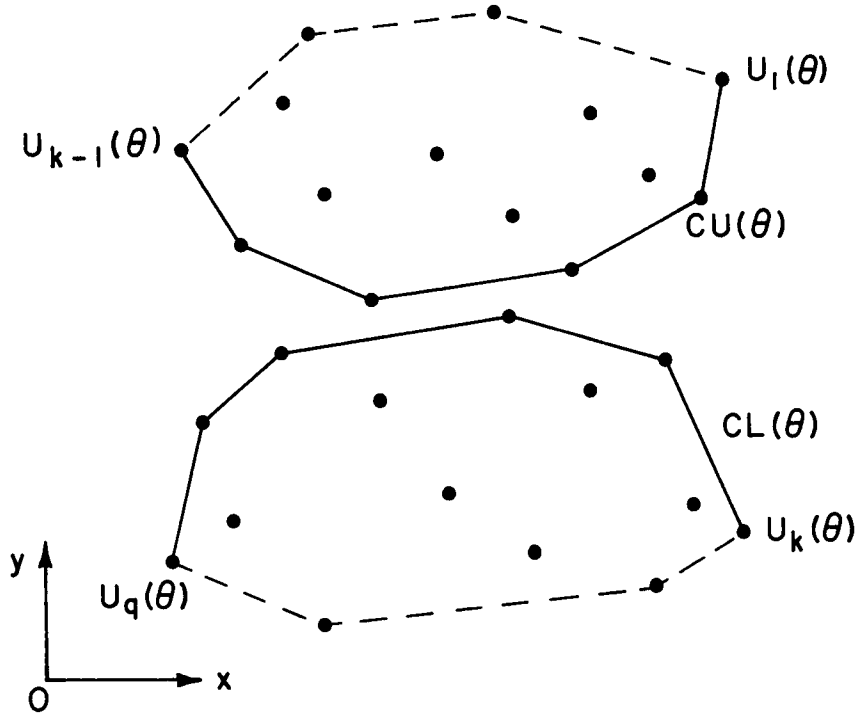

Figure 8. The top and bottom parts, $CL(\theta)$ and $CU(\theta)$.

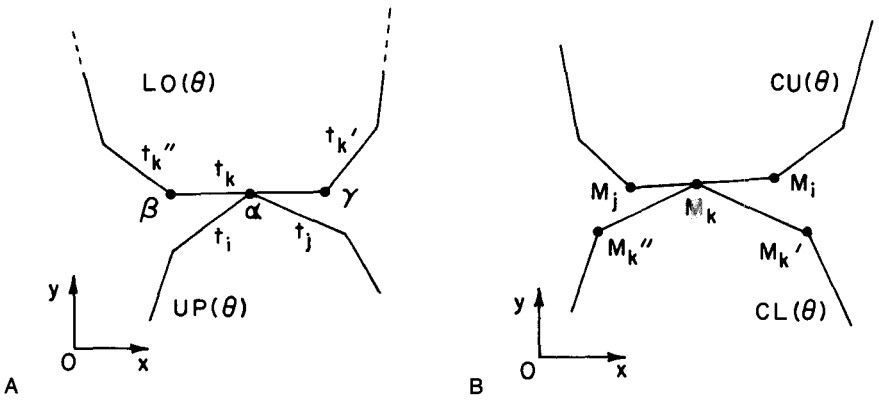

Figure 9. Characterizing containing placements.

PROOF. First we observe that the three lines $Y = a_tX + b_t (t = i, j, k)$ have a common intersection if and only if the three points $M_t = (a_t, b_t)$ $(t = i, j, k)$ are collinear. Also it is clear that $slope(t_i) \geq slope(t_k) \geq slope(t_j)$ is exactly equivalent to $a_i \geq a_k \geq a_j$. As a result, the intersection of t_i and t_j lies on t_k and $UP(\theta)$ lies below t_k if and only if M_k lies on M_iM_j. The last condition is that α actually lies on the segment $\beta\gamma$. We express this by saying that the X-coordinates of β, α, γ are increasing in this order. Analytically, this is equivalent to

$$(b_k - b_{k''})/(a_{k''} - a_k) \leq (b_j - b_i)/(a_i - a_j) \leq (b_{k'} - b_k)/(a_k - a_{k'})$$

that is,

$$slope(M_kM_{k'}) \leq slope(M_jM_i) \leq slope(M_{k''}M_k)$$

which completes the proof (Figure 9B). □

Since what we just proved still applies if we exchange the roles of $UP(\theta)$ and $LO(\theta)$, that is, assuming that a vertex of $LO(\theta)$ lies on an edge of $UP(\theta)$, we can summarize this result as follows: "$LO(\theta)$ and $UP(\theta)$ intersect in exactly one point if and only if $CU(\theta)$ and $CL(\theta)$ do so, too." Now the simple observation that if we translate $LO(\theta)$ vertically upward [resp. downward], $CL(\theta)$ will also be translated vertically upward [resp. downward], leads to the following characterization lemma:

LEMMA 7. *$LO(\theta)$ and $UP(\theta)$ intersect in the interior if and only if $CL(\theta)$ and $CU(\theta)$ do not intersect. So, there is a containing placement with the angle θ if and only if $CU(\theta)$ and $CL(\theta)$ do not intersect.*

PROOF. A direct consequence of the preceding remark. □

3.2.2 *Computing a Description of CU*

To be able to use the characterization of Lemma 7, we need a description of both $CU(\theta)$ and $CL(\theta)$ as a sequence of vertices in, say, clockwise order. Since the polygonal lines are defined in a similar manner, we may concentrate on $CU(\theta)$ exclusively. For $\theta = 0$, we may use any standard convex hull algorithm [5] to obtain $CU(\theta)$ in time $O(q \log q)$. Actually it is possible to do it in time $O(q)$ by observing that the sequence of points $U_1(\theta), \ldots, U_{k-1}(\theta)$ can be viewed as a clockwise traversal of a polygon V whose convex hull contains $CU(\theta)$ as its bottom part (Figure 10). The convex hull can be obtained in $O(q)$ time, using McCallum and Avis' algorithm for example [7], and a simple traversal of the boundary of this convex hull permits us to find its rightmost [resp. leftmost] vertex, from which we derive its bottom part. This gives us yet another, more efficient method for computing a containment placement of P achievable by translation only. Indeed, from Section 3.1 and the preceding result, it follows that we can compute $CU(\theta)$ and $CL(\theta)$ in $O(p + q)$ time, and since they are convex, we can test their intersection in $O(q)$ time [10] which, with Lemma 7, completes the proof of Theorem 5. For the general problem, however, computing the convex hull explicitly will not be necessary, since $CU(\theta)$ will be computed recursively.

First, we need to define a data structure that will afford us a complete description of $CU(\theta)$, for any θ between 0 and 2π. To do so, we must observe the behavior of the points $U_i(\theta)$, $1 \leq i < k$. Each of them moves along the vertical axis $X = a_i$, and its motion is a continuous function of θ. To be more precise, let us define some notation. For all v_i, $1 \leq i \leq p$, d_i is the length of the segment $v_1 v_i$, and α_i is the angle $(v_i v_{i+1}, v_{i-1} v_i)$. Note that $\alpha_1 + \ldots + \alpha_p = 2\pi$. Whatever the value of θ, we will assume for the time being that P always rests above the X-axis, that is, its inter-

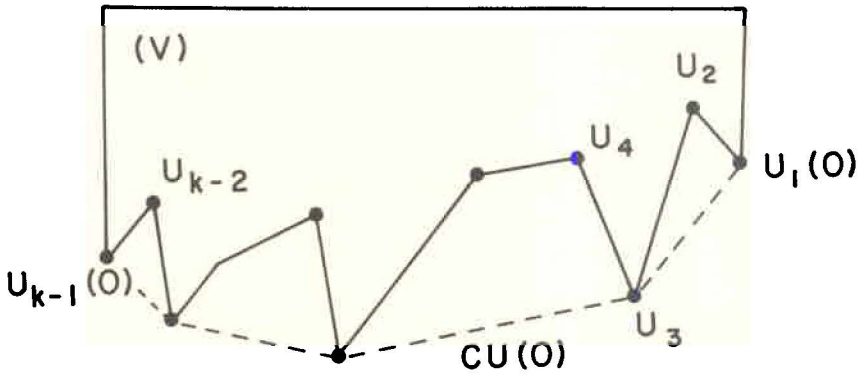

Figure 10. Constructing $CU(0)$.

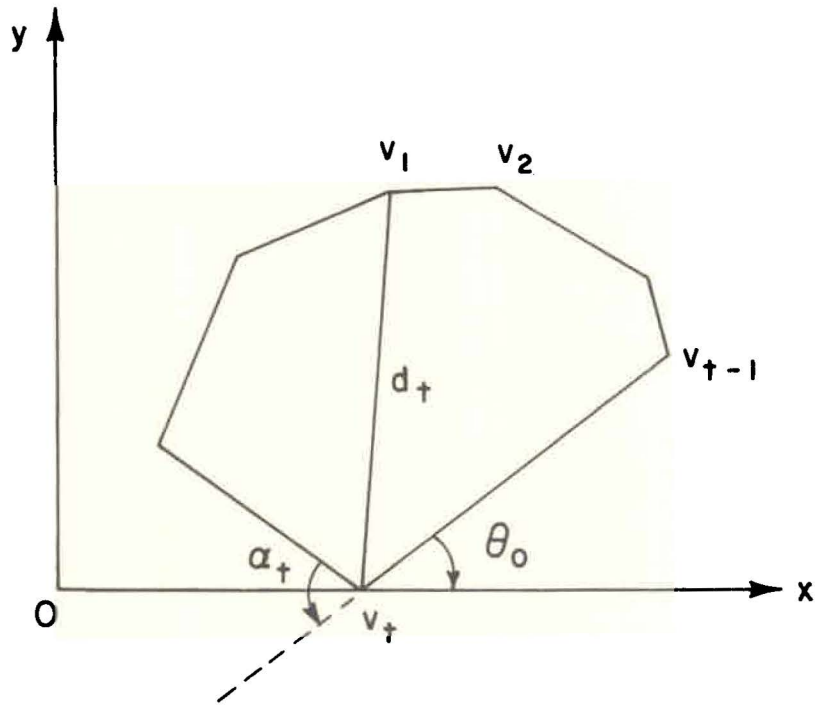

Figure 11. Expressing the function $y_1(\theta)$.

section with the X-axis consists of exactly one vertex v_t or one edge $v_{t-1}v_t$. As we start with $\theta = 0$, we introduce the angle $\theta_0 = (v_t v_{t-1}, OX)$, which permits us to express $y_1(\theta)$, the Y-coordinate of v_1, in terms of θ for any value of this angle. Note that θ_0 always lies in the interval $[-\pi, 0]$ (Figure 11).

For any value of θ in $[\theta_0, \theta_1]$, with $\theta_1 = \theta_0 + \alpha_t$, we have $y_1(\theta) = d_t \cos(\theta + \gamma_0)$ for some constants γ_0. Similarly we can see that there exists a constant γ_i such that for any θ in $[\theta_i, \theta_{i+1}]$, where θ_{i+1} is defined by the recurrence relation[5] $\theta_{i+1} = \theta_i + \alpha_{t+i}$, we have the following relation.

$$y_1(\theta) = d_{t+i} \cos(\theta + \gamma_i) \tag{1}$$

These relations induce a partition of the interval $[0, 2\pi]$ into $p + 1$ intervals $[\theta_i, \theta_{i+1}]$ with

$$0 \leq \theta_1 < \theta_2 < \ldots < \theta_p \leq 2\pi$$

such that, on any of these intervals, the function $y_1(\theta)$ belongs to the vector space **F** defined in Section 2. Figure 12 provides an illustration of this result.

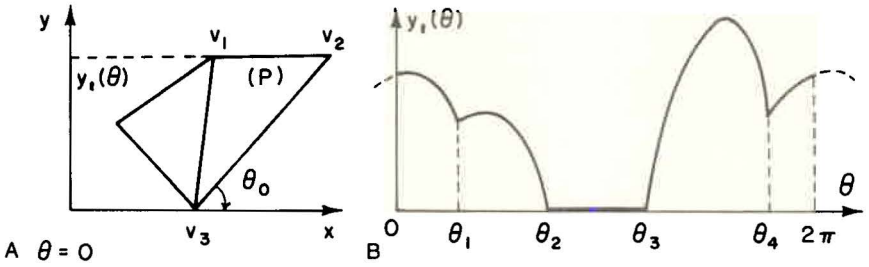

Figure 12. The behavior of $y_1(\theta)$.

We can now turn back to the behavior of the points $U_i(\theta)$, $1 \leq i < k$. As observed, the X-coordinate of $U_i(\theta)$ is independent of θ, so we simply have to express its ordinate $b_i(\theta)$. Note that if for any j, $1 \leq j < k$, we study the motion of the point v_1, as P rests on $line(w_j w_{j+1})$, lying on the same side of the line as W with θ varying from 0 to 2π, we will observe a behavior strictly similar to that of $y_1(\theta)$. More precisely, the distance h_j from v_1 to $line(w_j w_{j+1})$ can be expressed as $y_1(\theta + \beta_j)$ for some constant β_j. Note that the function $y_1(\theta)$ is periodic with period 2π; therefore, the function $y_1(\theta + \beta_j)$ is well defined. Finally, we can derive from this fact that the intersection of the line $t_j(\theta)$ with the Y-axis has an ordinate of the kind $b_j(\theta) = \gamma_i y_1(\theta + \beta_j) + \delta_j$, where γ_j and δ_j are two constants (Figure 13). It is easy to see that $\gamma_j = -1/\cos(OX, w_j w_{j+1})$, with $(OX, w_j w_{j+1})$ measured between 0 and 2π, and that $\delta_j = y_{w_j} - x_{w_j}(y_{w_{j+1}} - y_{w_j})/(x_{w_{j+1}} - x_{w_j})$. From this, we conclude that for any j, $1 \leq j < k$, we have the relation

$$b_j(\theta) = \gamma_j y_1(\theta + \beta_j) + \delta_j \tag{2}$$

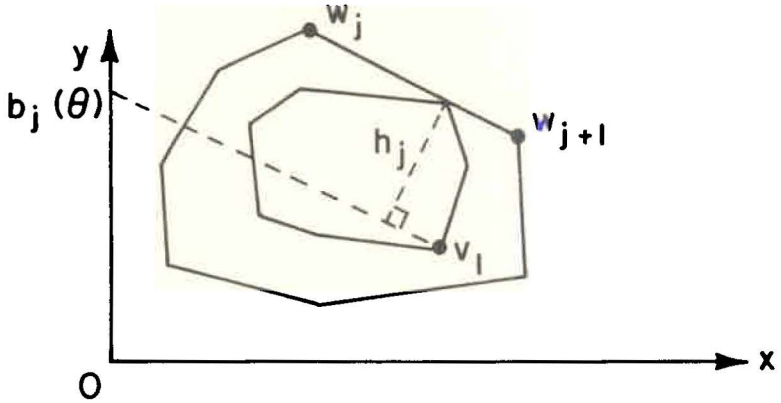

Figure 13. Expressing $b_j(\theta)$.

We are now in a position to give a full description of $CU(\theta)$, the bottom part of the convex hull of the points $\{U_1(\theta), \ldots, U_{k-1}(\theta)\}$. For simplicity, we regard the points $U_i(\theta)$ as points U_i moving vertically between the instants $\theta = 0$ and $\theta = 2\pi$, according to relation (2). Similarly $CU(\theta)$ can be viewed as a moving convex polygonal line. We observe that CU may, at times, lose or gain vertices, so we can say that a vertex U_i is *active* at time θ, if $U_i(\theta)$ belongs to $CU(\theta)$. Note that both U_1 and U_{k-1} are always active.

We must next tackle the following problem. How do we represent CU so that we can efficiently check the conditions of Lemma 7? Assume for the time being that in the time interval $[0, 2\pi]$, CU never loses or gains vertices. Let $CU_1, \ldots, CU_t$ be a list of its vertices in clockwise order. Since CU_j obeys relation (2), its ordinate is a function of $\mathbf{F}$ on each interval $[-\beta_j, \theta_1 - \beta_j], \ldots, [\theta_p - \beta_j, 2\pi - \beta_p]$. Taken modulo 2π, this sequence partitions $[0, 2\pi]$ into $p + 1$ intervals. Therefore we can describe the behavior of CU by setting up a doubly linked list L consisting of t cells, each containing enough space for three real numbers:

$$L = Cell(CU_1) \leftrightarrow \cdots \leftrightarrow Cell(CU_t)$$

The INFO part of each cell $Cell(CU_j)$, at each instant θ, consists of three numbers A, B, C specifying that at that instant, the ordinate of CU_j is $A \cos(\theta + B) + C$. Along with the data structure L given for $\theta = 0$, i.e., describing $CU(0)$, we need a list of instructions I, in order to update the INFO parts of the cells of L at the appropriate times.

Since a partition of $[0, 2\pi]$ into $p + 1$ intervals is needed to describe the behavior of each CU_j, we must set up a list I of $t(p + 1)$ instructions of the kind (t_i, j, A, B, C) $(1 \leq i \leq t(p + 1))$ with $t_1 = 0$, meaning that at instant t_i, the INFO part of CU_j should be updated and set to (A, B, C). Of course, we require that the t_i's appear in increasing order, which corresponds to merging the t partitions of $[0, 2\pi]$. Note that for any i, $1 \leq i < t(p + 1)$, every point CU_j has an ordinate of the form $A \cos(\theta + B) + C$ for any θ in $[t_i, t_{i+1}]$, where the triplet (A, B, C) is given by the contents of $Cell(CU_j)$.

Now what if CU happens to gain or lose vertices? We can extend the preceding data structure to handle these cases as well. Such situations are characterized by the collinearity of three consecutive vertices of CU, say CU_i, CU_j, CU_k, in clockwise order. If t is the time when this gain or loss occurs, the list I must contain the instruction $[t, j, i, \text{insert}, A, B, C]$ if CU_j is to be inserted next to CU_i in clockwise order, and $[t, j, \text{delete}]$ if CU_j is to be deleted. This being self-explanatory, we simply note that by giving fixed locations to the cells, we can ensure that any instruction of I can be executed in constant time. Whereas the representation by means of data L and instructions I is clearly adequate for describing the

behavior of CU, we must still investigate size issues and give an efficient method for computing L and I.

We next turn to the latter problem. Efficiency will be ensured by means of a divide-and-conquer strategy. Since the method is straightforward for size 1, we will directly attack the general recursive step. We assume two pairs of lists (L_1, I_1) and (L_2, I_2), corresponding to $CU^{(1)}$ and $CU^{(2)}$, respectively, where $CU^{(1)}$ [resp. $CU^{(2)}$] is the bottom part of the convex hull of $\{U_1, \ldots, U_{[k/2]}\}$ [resp. $\{U_{[k/2]+1}, \ldots, U_{k-1}\}$). We start out by computing the line of support of $CU^{(1)}$ and $CU^{(2)}$ at time $\theta = 0$, which gives us $CU(0)$, hence the list L. To do so, we use any standard linear time algorithm[6] [3] (Figure 14).

Next we have to set up the list of instructions I for CU. To this end, we introduce some notation. Let a_1 and a_2 be the two vertices of $CU^{(1)}$ and $CU^{(2)}$, respectively, in contact with the line of support (Figure 14). The points a_1 and a_2 are to be regarded as variables taking on the name of the particular vertices that they happen to represent at any given time. Similarly, we define b_1, c_1, and b_2, c_2, as the neighbors of a_1 and a_2, respectively (Figure 14). Wlog, we can assume that such neighbors are always well defined. The first task consists of merging the two sets of instructions I_1 and I_2 in chronological order. Then we proceed with scanning the resulting list I, keeping, deleting, or adding instructions to the set. The major work is to keep track of the points a_1 and a_2. We define the interval $\delta(a_1, a_2)$ to be the set of indices corresponding to the points U_i lying between a_1 and a_2. More precisely, if $a_1 = U_u$ and $a_2 = U_v$ (note that $1 \leq u \leq k/2$ and $k/2 + 1 \leq v < k$), $\delta(a_1, a_2)$ is the set $\{u + 1, u + 2, \ldots, v - 1\}$.

As we scan down the list I, it is clear that all instructions relating to an index in $\delta(a_1, a_2)$ should be discarded.[7] Difficulties arise when $\delta(a_1, a_2)$ must be updated. This corresponds to a situation where one of the triplets (a_1, a_2, c_2), (a_1, a_2, b_2), (a_2, a_1, b_1), (a_2, a_1, c_1) becomes collinear.

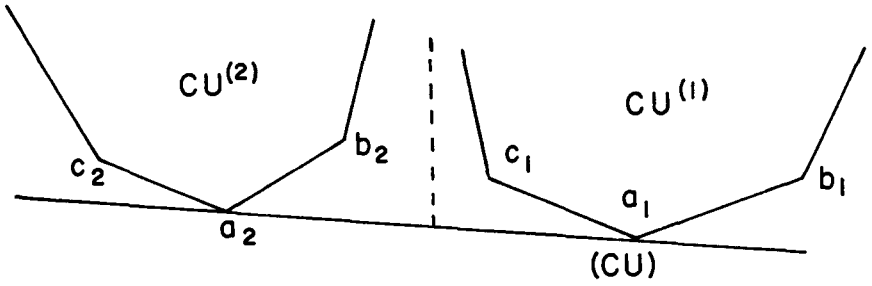

Figure 14. Computing $CU(\theta)$.

Since each of these four triplets involves a_1 and a_2, we can associate to each of them a function f_i of θ, defined as the distance of the third point in the triplet to *line*(a_1, a_2). All these functions f_1, f_2, f_3, f_4 can be expressed in a similar way

$$\alpha y_{ui} + \beta y_{uj} + \gamma y_{uk}$$

for some constants α, β, and γ, depending only on the X-coordinates of the three points U_i, U_j, U_k involved in the function. The functions y_{ui}, y_{uj}, and y_{uk} belong to **F** over some intervals partitioning $[0, 2\pi]$ into $p + 1$ parts. Therefore each of the functions $f_i(\theta)$ belongs to **F** (recall that **F** is a vector space) over intervals that partition $[0, 2\pi]$ into at most $3p + 1$ parts. As a result, if we are given the interval $[t, t']$ in which θ falls, we can compute in constant time the smallest θ', $t \leq \theta \leq \theta' \leq t'$, if it exists, such that $f_i(\theta') = 0$.

With these facts in mind, we are ready to describe the algorithm for setting up the list I. It essentially consists of going through the list I, as initially set up, removing or adding instructions to the set, and always executing the remaining instructions on the list L, so that at any step of the procedure, L gives the actual picture of CU at this step. I is assumed to be a linear list so that scanning down I allows for constant time insertions and deletions.

Step 1. Let L be the doubly linked list describing $CU(\theta)$ and let I be defined as the merge of I_1 and I_2. Let R denote the number of instructions in I, with $t_1, \ldots, t_R$, being the instants associated with these instructions. Determine the vertices a_1, a_2, b_1, b_2, c_1, c_2, as defined earlier, and for each of them, find its corresponding triplet (A, B, C). To do so, look up in L. Set i to 0.

Step 2. Set i to $i + 1$, and θ to t_i. If $i = R + 1$, then STOP.

Step 3. Compute the smallest θ', $\theta < \theta' \leq 2\pi$, such that the product $\prod f_i(\theta') = 0 (1 \leq i \leq 4)$. If no such θ' is found, let $\theta' = \infty$.

Step 4. If $\theta' \geq t_{i+1}$, the value of a_1 and a_2 remains the same over the interval $[\theta, t_{i+1}]$. The only necessary updating may come from the instruction $[t_{i+1}, j, \ldots]$ of I. In all cases, we execute it on L if and only if j does not belong to $\delta(a_1, a_2)$, so as to update L for the forthcoming instants $> t_{i+1}$. If the instruction is of the form $[t_{i+1}, j, A, B, C]$, check if one of the variables a_1, a_2, b_1, b_2, c_1, c_2 has the value U_j, in which case its corresponding triplet (A, B, C) should be updated as specified by the instruction. Slightly more complicated is a situation where the instruction involves deleting or inserting a vertex U_j, i.e., $[t_{i+1}, j,$ delete] or $[t_{i+1}, j, l,$ insert, $A, B, C]$. Up to symmetrical cases, there

are only two situations to investigate:

1. The variable c_2 has the value U_j, and U_j must be deleted. Here, a simple look-up in L gives the successor of U_j in increasing order, which is precisely the new value to which c_2 should be set.
2. The vertex U_j is to be inserted, and j belongs to $\delta(a_2, c_2)$. Then c_2 should simply be set to U_j.

In all cases, we can also update the triplets (A, B, C) corresponding to the updated variables, in constant time. At this point, both L and the sextuplet $S = (a_1, a_2, b_1, b_2, c_1, c_2)$ have been updated, so we can iterate by setting θ to t_{i+1} and go to Step 2.

Step 5. If $\theta' < t_{i+1}$, we have $f_i(\theta') = 0$ for some i, so the sextuplet S must be updated. Up to symmetrical cases, there are two situations to examine. Either (a_1, b_2, a_2) or (a_1, a_2, c_2) are collinear.

1. In the first case, set c_2 to the value of a_2, and a_2 to the value of b_2. Finally, by looking up in L, the new value of b_2 (the successor of b_2 in decreasing order) can be found in constant time.
2. Handling the latter case involves updating c_2 to the neighbor (in clockwise order) of its current value by looking up in L, setting a_2 to the former value of c_2 and b_2 to the former value of a_2.

In all cases, it is easy to update the corresponding triplets (A, B, C) in constant time.

Step 6. The final step is to update I and L. For I, we simply have to add an instruction of the kind [θ', *, insert, *] or [θ', *, delete] between the instructions [θ, *] and [t_{i+1}, *]. We will omit the details, as this operation is very straightforward. In all cases, also execute the newly added instruction on L, for reasons already stated.

Except for boundary conditions, which we do not think necessary to dwell upon (i.e., terminal cases of the recursive procedure), the preceding algorithm correctly computes the (I, L)-description of CU, as θ varies from 0 to 2π. We will omit the proof of correctness, which we believe to be essentially included in the presentation of this rather involved, yet conceptually simple algorithm.

Turning next to an analysis of its complexity, we observe that the crucial parameter is, of course, the number of instructions in I_1, I_2, and I. Let $R(N)$ be the maximum number of instructions in the set I relative to the bottom part of the convex hull of a set of N vertices of the kind $\{U_j, \ldots,$

$U_{j+N-1}\}$. Both the time and space requirements of the algorithm are bounded, up to within a multiplicative factor, by the function $T(N)$, defined by $T(1) = 0$ and

$$T(N) = 2T(N/2) + R(N) + 2R(N/2) + O(N) \tag{3}$$

$R(N)$ accounts for two very distinct kinds of instructions: (1) the instructions of the kind $[t, j, A, B, C]$, which correspond to a change of interval over which $U_j(\theta)$ belongs to F, and thus do not number more than Np; (2) the delete or insert instructions, which correspond to the situations where three consecutive vertices of CU become collinear. Let $S(N)$ be an upper bound on the number of such instructions.

$$R(N) \leq Np + S(N)$$

Since the collinearity of three vertices U_i, U_j, U_k corresponds to values of θ for which

$$f(\theta) = \alpha b_i(\theta) + \beta b_j(\theta) + \gamma b_k(\theta) = 0$$

for some constants α, β, γ, relation (1) and (2) show that for any given triplet (i, j, k), $f(\theta)$ belongs to $\mathbf{F}$ over $3p + 1$ intervals.[8] Furthermore, since each interval is less than π wide, $f(\theta)$ takes on any given value at most twice over each interval, therefore the number of distinct values of θ for which $f(\theta) = 0$ is bounded above by $6p + 2$. This gives a naive, conservative estimate of $S(N)$, namely, $S(N) = O(pN^3)$.

It is possible to improve on this result with a more careful analysis of the problem. To begin with, we view the line $CU(\theta)$ a little differently. Instead of representing its evolution in time with a sequence of instructions, we regard it as a sequence of triplets $[e, t_1(e), t_2(e)]$, where e is an edge of CU, and $[t_1(e), t_2(e)]$ is the maximum time interval during which e is not altered by an instruction of I. To make this notion clearer, we will examine the various cases where this happens. Recall that there are three kinds of instructions in I:

1. $[t, j, A, B, C]$; "update vertex j"
2. $[t, j,$ delete$]$; "delete vertex j"
3. $[t, j, i,$ insert$, A, B, C]$; "insert vertex j next to vertex i"

Therefore, up to symmetrical cases, the edge ab will be altered if one of the following actions takes place:

4. a is updated by (1) (Figure 15A)
5. a is deleted by (2) (Figure 15B)
6. c is inserted by (3) (Figure 15C)

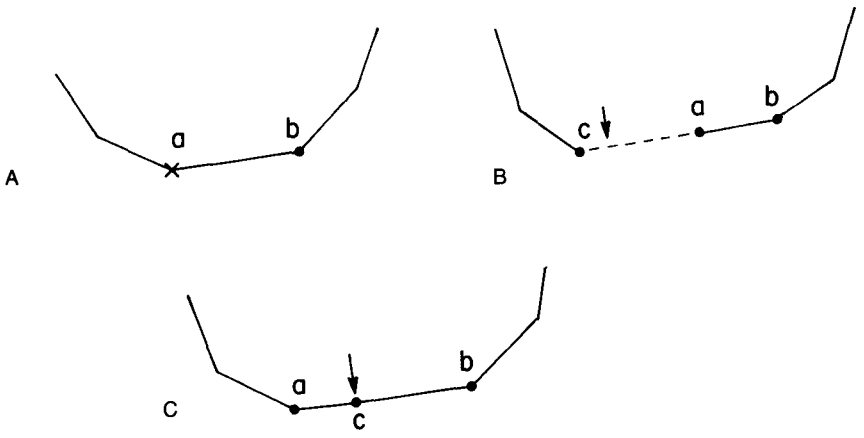

Figure 15. Updating the edges of CU.

Now we can legitimately define the maximum time interval $[t_1(ab), t_2(ab)]$ over which none of the previous actions takes place. Consider the list of all triplets $\{(e, t_1, t_2)\}$, sorted in increasing order with respect to the second component, t_1, and let $C(N)$ be the list with maximum cardinality defined for N points $U_j(\theta), \ldots, U_{j+N-1}(\theta)$. Note that an instruction of type (1) or (3) creates two triplets, whereas an instruction of type (2) creates one triplet, therefore $| C(N) | \geq R(N)$. The next step must be to evaluate the cardinality of $C(N)$. To do so, we first introduce the notion of *temporal* segment. For any pair of vertices U_i, U_j, we can partition $[0, 2\pi]$ into $2p + 1$ intervals over which the ordinate of both U_i and U_j is a function in **F**, i.e., of the form $A \cos(\theta + B) + C$, for constants A, B, C. This suggests breaking down the motion of U_iU_j into $2p + 1$ parts, i.e., representing U_iU_j by $2p + 1$ *temporal* segments $e_1, \ldots, e_{2p+1}$. Each segment e_i is defined only on one interval in $[0, 2\pi]$ and has the property that the ordinate of both of its end points is a function in **F**. Before proceeding, we will exhibit an important feature of temporal segments.

Let $e = U_iU_j$ and $f = U_iU_k$ be two temporal segments, with either $i \leq j, k$ or $k, j \leq i$. It is clear that e and f can never both belong to CU at the same time. Moreover, we can show that if e is an edge of CU at time t_1, f becomes one at time $t_2 > t_1$, and finally e regains its status as an edge of CU at time $t_3 > t_2$, then f can never become an edge of CU again. Indeed, if it did, U_k would have to cross the line passing through e a third time, which is impossible since the collinearity of U_i, U_j, U_k corresponds to values of θ for which $A \cos(\theta + B) + C = 0$, for some coefficients

A, B, C, and at most two such values can be found on any interval less than π wide (note that this result still holds, if we exchange e and f).

This fact is subtler than it may appear. Indeed, one may think that f could actually become an edge of CU again, once e has "died," i.e., has ceased to be defined as a temporal segment. Recall that temporal segments are defined only on a subinterval of $[0, 2\pi]$ less than π wide. This is, however, impossible because of relations (1) and (2) (Figure 12). Indeed, if the lifetime of temporal segments were extended to $[0, 2\pi]$, they would always lie above the line $CU(\theta)$; therefore, even "dead," e would have to recross f a third time, to let it become an edge of CU again. Collinearity being still expressed with the same coefficients A, B, C, this leads to an impossibility.

Turning our attention back to the list $C(N)$, we observe that each triplet $[e, t_1(e), t_2(e)]$ is associated with exactly one temporal segment (the converse is not true, in general), so we can replace each triplet in $C(N)$ by its corresponding temporal segment, and thus we obtain a list of same cardinality, yet with possible duplicates. The next step is to show that there cannot be very many duplicates. Consider two successive duplicates x in the list $C(N)$: $(\ldots x \ldots x \ldots)$, the first corresponding to a triplet (e, t_1, t_2), the second to (e, t_1', t_2'). We have $t_1 < t_2 < t_1' < t_2'$, so at time t_2, x ceases to be an edge of CU. This involves either the insertion of a vertex between the end points of x or the deletion of either of its end points. In both cases, there is at least one temporal segment y that becomes an edge of CU at time t_2 and fits exactly into the conditions of the fact established earlier. As a result, y will never appear again in the list $C(N)$ after the second x. In other words, between any pair of successive occurrences of x, there exists a y such that the configuration $C(N) = (\ldots x \ldots y \ldots x \ldots y)$ is impossible.

This gives us a powerful means to bound the number of duplicates.[9] Before proceeding, we adopt the notation $C(N) = (\ldots x_1 \ldots y_1 \ldots x_2 \ldots y_2 \ldots)$ to mean that x_1 and x_2 [resp. y_1 and y_2] are two occurrences of the same temporal segment, namely, x [resp. y], and that the order among indices corresponds to precedence in time (i.e., in the list $C(N)$, too). We observe that y_1 is the actual cause of the disappearance of x_1 as an edge of CU, hence of the occurrence of the duplicate x_2. Therefore, to each duplicate x, we can associate a unique element $f(x)$, such that $f(x)$ precedes x in the list and no occurrence of $f(x)$ can be found after x. In our example, $f(x_2) = y_1$. Unfortunately it is not clear at all that f may not map several duplicates to the same element.

To investigate this possibility, we observe that a temporal segment that becomes an edge of CU through insertion causes the disappearance of exactly one edge of CU, whereas a case of deletion causes two edges to

disappear. Figure 16 illustrates this problem, with $x = U_iU_j$ and $y = U_iU_k$. An insertion of U_k causes U_iU_j to disappear, whereas a deletion of U_j will result in the disappearance of both U_iU_j and U_jU_k. This is not a major difficulty, however, and we can overcome it by breaking down the function f into two functions f_1 and f_2.

Recall that there is always either $j, k \leq i$ or $j, k \geq i$. In the first case, we set $f_1(x_2) = y_1$, and in the latter $f_2(x_2) = y_1$. This ensures that every duplicate is mapped to a preceding element in the list and that both f_1 and f_2 are one-to-one on their respective domain. Of course we redefine $f(x)$ to be $f_1(x)$ or $f_2(x)$, whichever is defined. The next step is to show that not only f_1 and f_2 are one-to-one in the list $C(N)$, but also that for any x and y, $f_1(x)$ and $f_1(y)$, if defined, are not duplicates of each other, i.e., are distinct temporal edges. This is the object of the following technical lemma.

LEMMA 8. *For any x and y in the list $C(N)$, if $f_1(x)$ and $f_1(y)$ are defined, they are distinct temporal segments.*

PROOF. From now on, the relation $<$ pertains to the order in the list $C(N)$. Suppose that there exist two duplicates x and y in $C(N)$, such that $f_1(x)$ and $f_1(y)$ are the same temporal segment z. Note that by definition of f_1, it follows that all three segments have a common end point U_i. Therefore only one of the segments x, y, z can be an edge of CU at any given time. Wlog, assume that $f_1(x) < f_1(y)$. This means that z causes the disappearance of x before that of y. As a result, one occurrence of y is needed between the two occurrences of z, namely, $C(N) = (\ldots x \ldots z_1 \ldots y \ldots z_2 \ldots)$, and since f_1 maps elements backward, i.e., to preceding elements in the list, we are left with only the following possibilities.

1. x precedes y, i.e., $C = (\ldots x_1 \ldots z_1 \ldots y_1 \ldots z_2 \ldots x_2 \ldots y_2)$. This order implies that y crosses x at least once in order

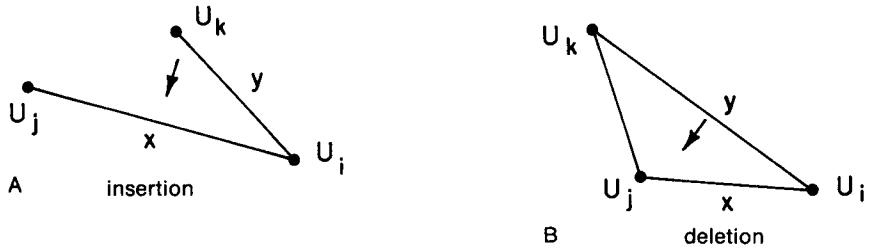

Figure 16. The two cases in the definition of f. (A) Insertion; (B) deletion.

to become y_1, then at least once for x to become x_2, and finally at least a third time to become y_2. As already seen, however, two temporal segments sharing one end point cannot cross three times, hence an impossibility.

2. y precedes x, i.e., $C = (\ldots x_1 \ldots z_1 \ldots y_1 \ldots z_2 \ldots y_2 \ldots x_2)$. In this case, y must cross z at least once from z_1 to y_1, once from y_1 to z_2, and once from z_2 to y_2, hence contradiction.

This completes the proof of the lemma, which of course also applies to f_2. □

We are now in a position to derive a bound on the size of $C(N)$.

LEMMA 9. *The number of instructions $R(N)$ in we is $O(pN^2)$.*

PROOF. Let d_1 be the total number of distinct elements in $C(N)$, and d_2 the number of duplicates; $| C(N) | = d_1 + d_2$. We will represent each of the d_1 distinct elements in $C(N)$ by a pile of bricks. Initially all piles are empty; then we proceed to scan the list $C(N)$ from the first to the last element, one element at a time, updating the piles accordingly. Let x be the next element scanned. We begin by putting one brick into its corresponding pile, then if x is a duplicate, we put one STOP-brick onto the pile of $f(x)$. Note that, by definition of f, once a pile has received a STOP-brick, it can never grow again with regular bricks. Moreover, Lemma 8 shows that any pile can receive at most two STOP-bricks, one from f_1, the other from f_2. This implies that there are at most $2d_1$ STOP-bricks, hence $d_2 \leq 2d_1$. Now, from $| C(N) | = d_1 + d_2$, we derive $| C(N) | \leq 3d_1$. Finally since the number of distinct elements in $C(N)$ is dominated by the total number of temporal segments, we have $d_1 = O(pN^2)$, which completes the proof. □

Relation (3) shows that the algorithm for setting up the set of instructions I for CU requires time and space $T(q)$, with $T(q) = 2T(q/2) + O(pq^2) = O(pq^2)$. We conclude

LEMMA 10. *Setting up we and L for $CU(\theta)$ requires $O(pq^2)$ time.*

3.2.3 *The Final Phase*

The main algorithm will start by calling upon the procedure of Theorem 5 to check whether there exists a containing placement for $\theta = 0$. If there is one, the algorithm can report the location of P inside Q in time $O[q \log(p + q)]$. Otherwise, the algorithm proceeds with computing the pairs $(I,$

L) and (I', L') relative to $CU(\theta)$ and $CL(\theta)$, respectively, which can be done in time $O(pq^2)$, according to Lemma 10. Once the pairs (I, L) and (I', L') have been set up, we can use them to provide a description of the intersection $CU(\theta) \cap CL(\theta)$ at all times and can detect the first occurrence of a nonintersection, which in Lemma 7 has been shown to correspond to a containing placement. Such an occurrence corresponds to a one-point intersection; and knowing this point as well as the corresponding value of θ will permit us to report the location of the containing placement, and thus terminate.

At time $\theta = 0$, L and L' provide us with a description of $CL(\theta)$ and $CU(\theta)$. These convex lines necessarily intersect at this point, since we assume that there is no containment placement for that particular value of θ (Lemma 7). We can use any linear algorithm for computing the intersection of two convex polygons [10], in order to determine the two intersection points a and b. Let a_1a_2 and $a_1'a_2'$ [resp. b_1b_2 and $b_1'b_2'$] be the two edges of CL and CU, respectively, which intersect at point a [resp. b] (Figure 17). In a procedure identical to the recursive step of the construction of $CU(\theta)$, we first merge the lists I and I', then we proceed to scan the resulting list, updating the variables (a_1, a_2, a_1', a_2') and (b_1, b_2, b_1', b_2'). A case analysis of the relative position of these eight points will handle the updating as before. Note that, as usual, we must actually execute the instructions of I and I' on L and L', respectively, in order to gain access to the neighbors of these eight vertices and ensure the updating. Because of the similarity with the recursive step in computing $CU(\theta)$, we omit the details. The purpose of this computation is to keep track of the intersection points a and b, so that the first occurrence of a nonintersection can be detected immediately.

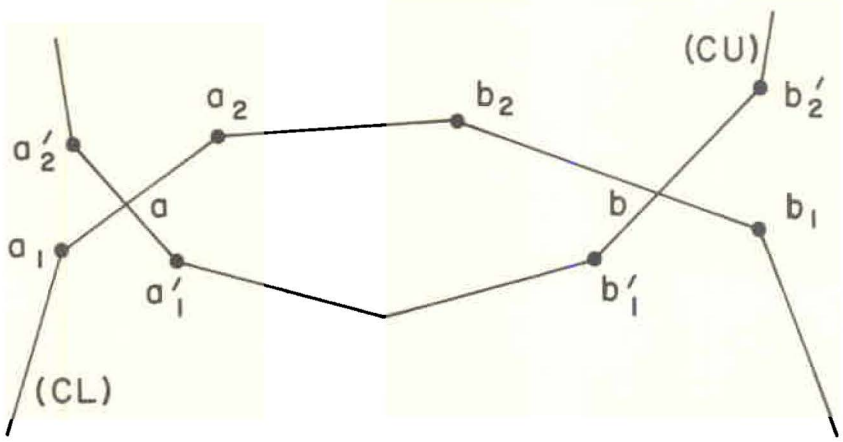

Figure 17. Computing the intersection of CU and CL.

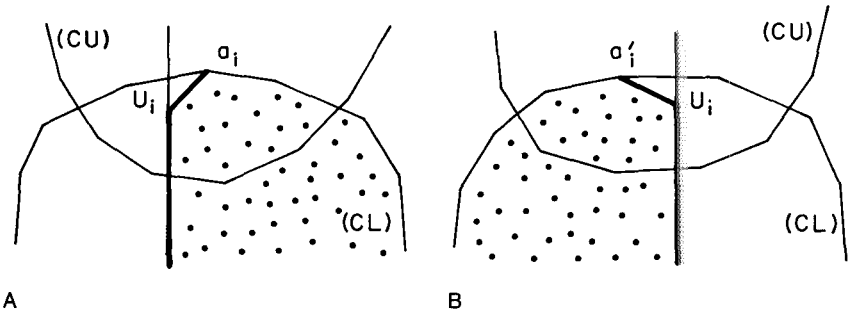

Figure 18. The definition of $U_i a_i$ and $U_i a_i'$.

Up to within a constant factor, the execution time of this final procedure is dominated by the total number of triplets assigned to the variables a_1a_2, $a_1'a_2'$, b_1b_2, $b_1'b_2'$ over the interval $[0, 2\pi]$. This quantity, in turn, is certainly bounded above by the total number of triplets assigned to CU and CL, i.e., $O(pq^2)$, added to the number of times a vertex of CU crosses an edge of CL (i.e., a_1a_2 or b_1b_2), or a vertex of CL crosses an edge of CU. Because of the similarity of these two cases, we may consider only T, the number of times a vertex of CU intersects a_1a_2.

LEMMA 11. $T = O(pq^2)$.

PROOF. To evaluate T, we will use a technique very similar to that introduced for bounding the description size of CU. For any point U_i inside CU (i.e., $1 \leq i \leq k - 1$), let S_i be the set of points U_j lying within CL to the right of U_i, that is, such that $k \leq j \leq q$ and the X-coordinate of U_i is smaller than that of U_j. We consider the convex hull of the set $S_i \cup \{U_i\}$ and, in particular, the edge U_ia_i of the convex hull that lies above S_i (Figure 18A). Note that a_i may or may not be a vertex of CL. Next we form the list C_i consisting of all the triplets assigned to U_ia_i over $[0, 2\pi]$, and we merge the $k - 1$ lists $C_1, \ldots, C_{k-1}$ into a list C, ordered chronologically as usual, that is, according to the second element of the triplets. Recall that the presence of a triplet (e, t, t') in S_i signifies that U_ia_i is the temporal segment e from time t to time t'.

As we did before, we proceed to replace each triplet (e, t, t') in C by the single element e, so that C becomes a list of temporal segments with possible duplicates. We can still define a function f mapping each duplicate to the unique element that caused its disappearance as an edge U_ia_i, so Lemma 8 can still be applied. Note that with this setting of the problem, there is no need to introduce two subfunctions f_1 and f_2. A similar reasoning will show that, once again, the total number of elements

in C is bounded by $O(pq^2)$, therefore if q_i is the total number of triplets assigned to U_ia_i, we have $q_1 + \ldots + q_{k-1} = O(pq^2)$. Similarly we can define the set S_i', consisting now of all the points $U_j(k \leq j \leq q)$ that lie to the left of U_i, and can define the variable U_ia_i. Repeating the preceding reasoning and introducing the number q_i' of triplets assigned to U_ia_i' (Figure 18B), we derive $q_1' + \ldots + q_k' = O(pq^2)$.

The motivation for these considerations comes from the following fact: every time a vertex of CU crosses CL, there exists a vertex U_i such that a_i, U_i, a_i' are collinear. Thus, we can use the preceding results to bound the number of these occurrences. For each $U_i(1 \leq i \leq k - 1)$, we merge the triplets assigned to U_ia_i and U_ia_i' into an ordered list D_i. The $q_i + q_i'$ time intervals present in D_i may overlap; however, we can still break them down into at most $2(q_i + q_i' - 1)$ nonoverlapping intervals having the property that, on each of them, exactly one triplet is assigned to U_ia_i and one to U_ia_i'. As a result, it appears that during each interval, U_i can cross a_ia_i' at most twice; therefore, the number of times U_i can cross CL is bounded by $4(q_i + q_i') - 2$. Summing up for all U_i, $1 \leq i \leq k - 1$, we derive $T = O(pq^2)$. □

Since T is an upper bound on the complexity of our final procedure, we can put the results together and conclude

THEOREM 12. *For any pair of simple polygons P and Q (Q convex) with, respectively, p and q vertices, it is possible, in time $O(pq^2)$, to determine whether there exists a containing placement of P reachable by translation and rotation, and if there is one, report its location.*

4. FITTING A TRIANGLE INTO A CONVEX POLYGON

As is often the case, a general-purpose algorithm can be found to be advantageously replaced by a more restrictive procedure, when the problem at hand loses some of its generality and lends itself to handy, tailored characterizations. The containment problem gives an illustration of this feature. Assume that P, the polygon to fit, is now a triangle ABC. The general method described in the previous section becomes essentially quadratic, but the overhead involved is still rather heavy, so a simpler algorithm may reasonably be sought. From an investigation of the geometric properties hidden behind the new statement of the problem, we are able to devise a very straightforward method for testing containment. Although the complexity of this algorithm is still quadratic, its simplicity makes it a much more efficient, hence desirable, alternative.

To begin with, we will show that we may consider only *coinciding*

placements, that is, placements with one vertex of P coinciding with one vertex of Q (Figure 20).

LEMMA 13. *Any stable containing placement of P can be transformed into a coinciding containing placement.*

PROOF. Starting from a stable containing placement of P and using the notation of Figure 19, we distinguish between two cases:

1. To begin with, we assume that P is free to move in one direction, e.g., Bb, wlog. More precisely, we assume that the intersection of $line(aa')$ and $line(bb')$ lies on the same side of AB as C (Figure 19A). Wlog we suppose that the two lines are not parallel. Consider the two lines parallel to AB and passing respectively through a and b, and let D be the one closer to $line(AB)$. Wlog we assume that D passes through b. Since Q is convex, none of its vertices between b and a in clockwise order lies above D, therefore we can translate P along bb' toward b, until vertex B coincides with vertex b, which precisely gives us a coinciding placement.

2. In the general case, however, the assumption made at the beginning is not valid, and we must contemplate a different strategy. Earlier we saw

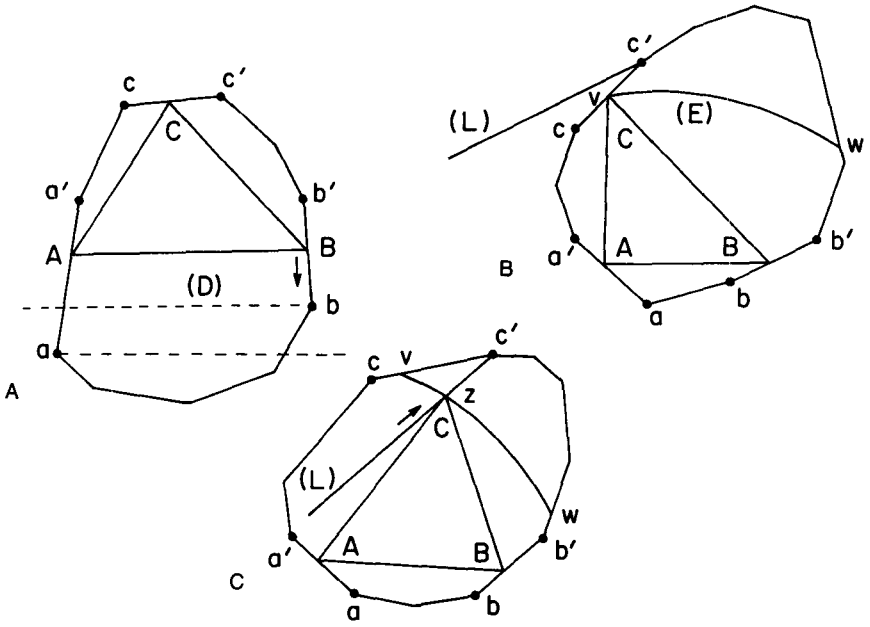

Figure 19. Obtaining a coinciding placement.

that, as A and B slide on aa' and bb', respectively, C describes an ellipse (E). We can assume that (E) intersects the boundary of Q in at least two points v and w, such that v lies on cc' and the portion of (E) between v and w lies entirely inside Q (wlog we also assume that w follows v in a clockwise traversal of the boundary of Q). This may not be true only if A [resp. B] hits an end point of aa' [resp. bb'], in which case we have once again reached a coinciding placement.

Consider the infinite line L passing through c' and parallel to bb'. Suppose that $line(cc')$ and $line(bb')$ intersect on the same side of vB as A (Figure 19B). This resets the conditions of the first case considered in this proof; therefore, we may assume that it is not the case. This implies that L intersects the ellipse at a point z between v and w (Figure 19C). Thus, we can start rotating P so as to bring C at the point z, from which we proceed to translate P along L toward c'. This motion inside Q may be interrupted by three possible causes:

(i) C reaches c', which leads to a coinciding placement.
(ii) B reaches b', and we are in a situation similar to the previous case.
(iii) A hits an edge dd' of Q. This is possible only if A lies outside of the strip comprised between L and $line(bb')$, and $line(dd')$ intersects $line(bb')$ on the same side of $line(AB)$ as C. Although we do not have a stable containing placement at this point, the reasoning used at the beginning of the proof is still applicable and still shows how to exhibit a coinciding placement. □

It is clear that any coinciding placement can be made stable simply by rotating P around its coinciding vertex, until a vertex of P first encounters an edge of Q. From this fact we derive a much simpler containment algorithm:

For any pair of vertices, one in P, the other in Q, determine if there exists a containing stable placement, for which these two vertices coincide.

The only difficulty may be to detect a containing placement for a given pair of coinciding vertices. Let A be the coinciding vertex of P. Let $\alpha = (AB, AC)$ and $\theta = (AX, AB)$ (Figure 20). As θ varies from 0 to 2π, the segment AB may, at times, lie inside then outside Q. Let $L_B = \{[\theta_1, \theta_2], \ldots, [\theta_b, \theta_{b+1}]\}$ be the list of intervals of θ for which AB lies in Q. It is easy to compute L_B by traversing the boundary of Q in clockwise order and testing every edge of Q against AB. Since the segment AB can cross no edge of Q more than twice, this method requires $O(q)$ steps. Similarly, we compute the list L_C defined in the same way with respect to C. Since P imposes an angular difference of α between AB and AC, we must shift

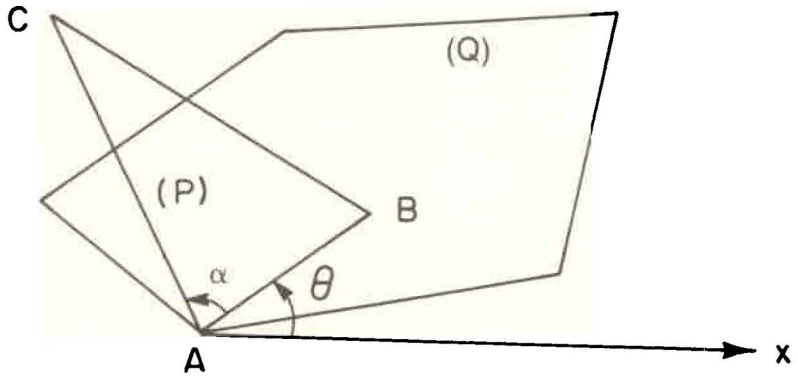

Figure 20. Computing containing placements of a triangle.

the list L_C by α, in order to be able to match it against L_B. Then only, we can compute the intersection of L_B and L_C, observing that this new list corresponds exactly to the containing placements of P. Computing the intersection is similar to merging two ordered lists, and thus requires $O(q)$ time. We can conclude:

THEOREM 14. *It is possible to determine if a triangle can fit into a convex polygon with q vertices, and if it can, report a containing placement, in* $O(q^2)$ *steps.*

5. CONCLUDING REMARKS

In addition to the naive algorithm for the general problem, this work gives two nontrivial algorithms for solving the containment problem restricted to convex polygons. The first preserves the generality of the problem. It relies on two powerful techniques: one is *duality,* which was first introduced in [8] and [2]; the other is the well-known, all-purpose *divide-and-conquer.* The second algorithm restricts the shape of the contained polygon to that of a triangle and uses the notion of geometric *reduction* to limit the set of solution candidates. Both algorithms are quadratic in the size of the containing polygon. This stems from the fact that they both attempt to enumerate all stable containing placements. With this strategy, a quadratic bound appears to be optimal, since it is possible to exhibit a class of polygons Q with q vertices giving $\Omega(q^2)$ distinct stable placements [4]. As a consequence, it is clear that only a nonenumerative method can free itself from this quadratic bound.

Further research on this subject includes devising a nonnaive algorithm for the case where the containing polygon is nonconvex, investigating the

possibility of fast expected-time or probabilistic algorithms, and of course extending the present work to higher dimensions.

ACKNOWLEDGMENT

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA order no. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539.

NOTES

1. Two polygons are similar if they can be made to coincide with each other by uniform enlargement, translation, and rotation.
2. A polygon is said to be simple if any pair of nonconsecutive edges do not intersect.
3. *Line*(ab) denotes the infinite line passing through the two distinct points a and b.
4. More precisely, since to every point on the boundary of a convex polygon corresponds another with the same X-coordinate, we define the bottom part of a convex polygon as the part of its boundary containing the points with the smaller Y-coordinates.
5. All indices are taken modulo p.
6. Recall that the standard method relies on a co-routine strategy, starting with any segment ab, with a on $CU^{(1)}$ and b on $CU^{(2)}$, then proceeding with a traversal of $CU^{(2)}$, starting from b, so as to increase the angle (aX, ab) as much as possible. When the maximum has been reached, the same procedure is carried out, now permuting the roles of $CU^{(1)}$ and $CU^{(2)}$. We iterate on this process, until no angle can be increased. Proving that the algorithm runs in linear time is straightforward.
7. Relating to an index j means an instruction of the form $[t, j, \ldots]$.
8. That is, there exist $3p + 1$ intervals partitioning $[0, 2\pi]$, over each of which the restriction of f is a function of F.
9. In the following, we will refer to duplicates as copies, that is, k occurrences of x in the list give $k - 1$ duplicates.

REFERENCES

1. Bentley JL, Ottmann T: Algorithms for reporting and counting geometric intersections. *IEEE Trans Comp* C-28(9):643–647, 1979.
2. Brown KQ: Geometric transforms for fast geometric algorithms. PhD thesis, Carnegie-Mellon University, Pittsburgh PA, 1979.
3. Chazelle BM: Computational geometry and convexity. PhD thesis, Yale University, New Haven CT, 1980. Also available as Technical Report CMU-CS-80-150, Carnegie-Mellon University, Pittsburgh PA, 1980.
4. Chazelle BM: unpublished manuscript, Carnegie-Mellon University, Pittsburgh PA, 1981.
5. Graham RL: An efficient algorithm for determining the convex hull of a planar set. *Info Proc Lett* 1:132–133, 1972.
6. Manacher GK: An application of pattern matching to a problem in geometrical complexity. *Info Proc Lett* 5:6–7, 1976.

7. McCallum D, Avis D: A linear algorithm for finding the convex hull of a simple polygon. *Info Proc Lett* 9:201–206, 1979.
8. Preparata FP, Muller DE: Finding the intersection of n half-spaces in time $O(n \log n)$. *Theoret Comp Sci* 8:45–55, 1979.
9. Shamos MI: Geometric complexity. *Proc 7th Annual SIGACT Symp*, pp 224–233, 1975.
10. Shamos MI, Hoey D: Geometric intersection problems. *Proc 17th Annual Symp on the Foundations of Computer Science*, pp 208–215, 1976.