ON THE REACHABILITY REGION
OF A LADDER IN TWO CONVEX POLYGONS

Minou Mansouri

School of Computer Science
McGill University, Montreal

August 1986

A thesis
Submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science.

Abstract

In the present thesis, we solve the problem of computing the *reachability* regions in two convex polygons for the endpoints of a ladder, which is allowed any motion provided that each endpoint remains within the boundaries of its respective polygon.

Using existing algorithms, this problem can be solved in O(n logn) time, where n is the number of polygon vertices. However, by taking advantage of the convexity of the polygons, we can reduce this time complexity and we propose an algorithm linear in the input size.

The computation of these regions, after having determined their existence, is done in two main steps : first the calculation of the *unreachability region* in each polygon, if it exists, then that of the *reachability regions.*

Résumé

Dans la présente thèse, nous calculons les régions *d'accessibilité*, dans deux polygones convexes, pour les points extrêmes d'un segment, auquel tout mouvement est permis avec cependant la contrainte que ces points extrêmes restent chacun a l'intérieur de leur polygones respectifs.

Utilisant les algorithmes existants, ce problème peut être résolu en temps O(nlogn), n étant le nombre de sommets dans les polygones. Or, profitant de la convexité de ces polygones, nous pouvons réduire la complexité en temps et proposons un algorithme linéaire en fonction du nombre de sommets.

Le calcul de ces régions, après avoir determiné leur existence, est fait en deux étapes : d'abord le calcul de la *région d'inaccessibilité* dans chaque polygone, s'il existe, puis celui des *régions d'accessibilité*.

Acknowledgments

I would like to express my deepest thanks to Professor Godfried Toussaint, my thesis supervisor, for his constant encouragment and interest, his guidance, for the many helpful discussions and for stimulating the creativity of the mind, in a lively atmosphere for research.

I am grateful to my friends Roger Hernandez and Stavros Mohlulis who cheerfully helped me with the typing of the manuscript.

I also thank all the professors, students and members of the staff, who helped enrich my studies at McGill University.

Table of Contents

CHAPTER 1

1.1 - Historical background

1.1.1 - The Kakeya problem

One of the first geometrical problems involving the motion of a line segment is the Kakeya problem, [Bes], [Cun], [Sch]. In 1917, the Japanese mathematician S. Kakeya posed the following problem : Let U = AB be a unit line segment in the plane. What is the least possible area swept by U if we were to move U from a position AB back to its original position with its endpoints reversed so that the final position is BA ? Refer to figure 1.1 . Kakeya conjectured that the three-cusped hypocycloid H of figure 1.2 inscribed in a circle of diameter 3/2 , with area $\pi/8$ is the minimum area in which U can be turned. Ten years later however, A. Besicovitch established that the switching of the endpoints of the segment U= AB can be done within an arbitrarily small area.

1.1.2 - More recent motion problems

Since then, other types of problems have arisen, all leading to a more interested study of the theory of movement in general and various instances of it in particular. Research in areas such as robotics , computer graphics, VLSI, image processing and artificial intelligence has stimulated considerable interest in the theoretical aspect of the existing problems and in particular, attention and importance has been given to the computational complexity of the problems.

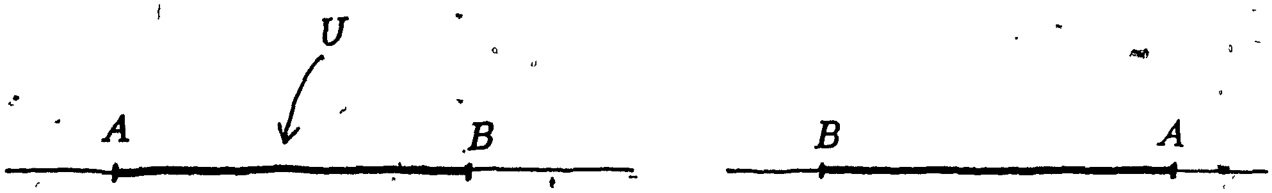


FIGURE 1.1

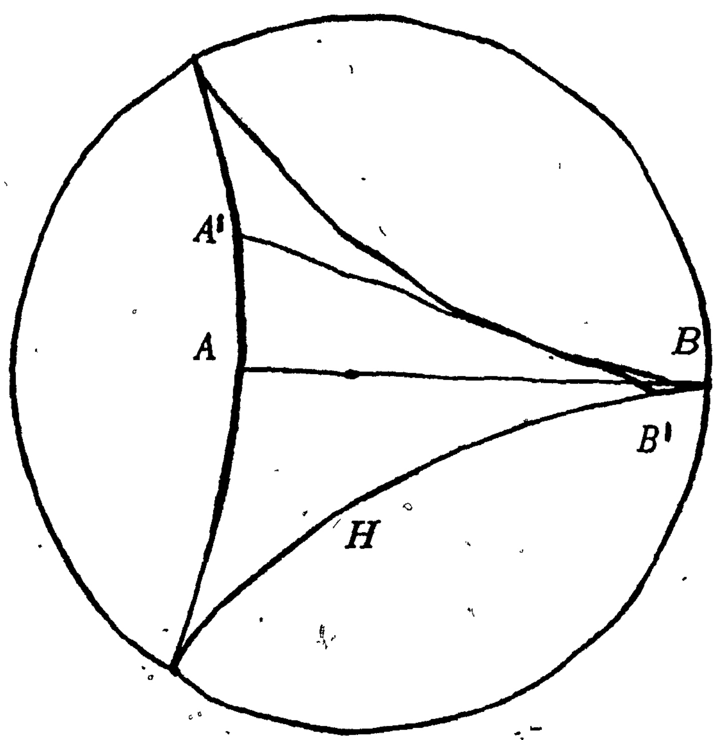


FIGURE 1.2

The algorithms for the motion problems often require and use results from the areas of computational geometry and graph theory. Examples of classical algorithms are the computation of the convex hull, triangulation, intersection detection , Voronoi diagrams, visibility graphs, point location and shortest path.

In contrast to *static geometry,* where the objects are inherently fixed and without mobility, there is also *kinetic geometry.* Remaining in the context of computational geometry, the word kinetic lends to different interpretations. One could be that the solving of a static geometry problem involves an implicit motion. As an example, imagine Jarvis' march or gift-wrapping process for the computation of the convex hull of a set of points, [Jar]. The other, more direct interpretation, is the attribution of an explicit movement to the geometrical objects of the problem. And a large class of problems , called *motion planning* problems come in the latter category. The *motion planning* problem has been addressed in several disciplines and is known by other names in the literature: findpath problem, obstacle-avoidance, collision-avoidance or movers' problem. Many recent papers look at this problem from a computational geometry viewpoint:

* motion planning is thus formulated purely in geometric terms. This allows a deeper study of the inherent mathematical structure of the problems.

* also various asymptotically efficient techniques drawn from the study of algorithms and data-structures are employed. Complexity theory also sheds considerable light on the inherent complexity of the motion planning problem.

For a survey of recent algorithms and complexity results for motion planning together with an emphasis on the computational geometry issues, refer to [Whi]. Also in [Yap1], Chee Yap presents a study of significant theoretical advances in algorithmic motion planning, with an emphasis on two "universal" techniques, the *decomposition* and the *retraction* approaches, respectively, that have been used to solve such problems, [Yap1].

Definition :

The *Mover's problem*, or *Findpath* : given the initial and desired final configurations of an object in 2- or 3-dimensional space , and given a description of the obstacles, determine whether there exists a continuous motion of the object from the one configuration to the other, and find such a motion if one exists.

Various more specific forms of this general definition have appeared in the literature. For example, the objects are sometimes assumed to be polygons or polyhedra and the motions sought might be sequences of pure rotations and pure translations. A variety of words have been used to evoke an image of the object being moved. It has been called a piano, a chair and a sofa for example. The sofa problem, for example, consists in moving a planar figure, the sofa, around a right angle bend in a corridor (see [Mos], [How], [Gol] and [Seb], see also [Str] for the problem of moving a "chair" through a door).

All these words suggest the assumption that the object is inflexible, but the general definition just given allows the possibility that the object consists of more than one part and that these parts are attached to one another in some flexible way, say by revolving or sliding joints. Other possibilities are that the parts of the objects are not attached and can function independently or that they can even change shape as well as configuration as they move.

In case the moving object consists of several independent pieces, the Mover's problem is generally called a *Motion Coordination problem*, see for example [Yap2]. The problem is that of choreographing the motion of disjoint bodies so that , starting at an initial configuration, they attain a goal configuration without ever colliding with themselves or with the obstacles.

In case the only objects are the ones to be moved (i.e. there are no fixed obstacles) and the final configuration is only specified by requiring that the objects be spread out, the problem is called a *Separability problem*. A very nice survey of separability problems is done by Toussaint in [Tou1].

For the even harder case of finding a path while the obstacles are also allowed to move, Kant and Zucker, in [KZ], generalize the path planning problem to one of *trajectory planning* , in a time-varying environment.

For the *Mover's problem*, however, the problem of moving simpler objects than polygons such as a disc or a line segment, also called a *ladder*, a *rod* or a *needle*, has received some attention. We will in this section present some recent results on moving a ladder in the plane and in the next section describe some problems in the area of *Separability*.

Mathematical and algorithmic analysis of the general motion planning problem began in the early eighties in a series of papers by Schwartz and Sharir, [SS1], [SS2] and [SS3]. They showed the possibility of using analytical and topological, rather than purely geometric, methods in motion planning. Using the projection approach, the Mover's problem in [SS1] is reduced to searching for a path in a *graph* that represents the connectivity properties of the space which is all the free legal configurations of the moving object, called FP for free position, that is, positions of the moving object where there exists no collision with the obstacles. Schwartz and Sharir give an $O(n^5)$ algorithm for planning the motion of a ladder where n is the number of line segments composing the boundaries of the obstacles, and in [SS5], they analyse the problem of a rod moving in 3D space. Their work stimulated approaches with a more topological flavor in papers such as [OSY1], [OY] and [Yap2].

For the specific case of a ladder moving past planar obstacles, [OSY1] have improved the $O(n^5)$ algorithm of [SS1] to obtain an $O(n^2 \log n)$ algorithm by applying what they describe as a *retraction* approach, a notion from topology. They obtain $O(n \log n)$ and $O(n^2 \log n)$ algorithms for moving a disc and a line segment respectively past planar obstacles, n being the number of sides in the obstacle boundaries. They do this by using the notion of a Voronoi diagram and its generalisation to 3D configuration space. Then a retraction mapping is applied to this diagram and the Mover's problem is thus reduced to a path search in the diagram. In fact a motion between two positions exists in FP if and only if an appropriately retracted motion exists within the subspace into which FP is mapped. Chien, Zhang, Zhang [CZZ] also analyse the planning of a collision-free path for a rod moving in the ~~plane,~~ among polygonal obstacles, using methods from topology. The rod is allowed translation and rotation. [OSY2] and [OSY3] use generalized Voronoi diagrams for planning the movement of a ladder . General motion planning is reduced to searching in a graph which represents a subset, the skeleton, of a "Voronoi complex". The construction of the diagram yields a motion planning algorithm for the ladder which runs in $O(n^2 \log n \log n)$ time.

Leven and Sharir [LS] give an $O(n^2 \log n)$ motion planning algorithm for a ladder. Their algorithm uses the same general techniques used in [SS1] for the partitioning of the 3D manifold of free positions FP of the ladder into connected components. This technique decomposes FP into simple connected cells, each cell being a vertex in the connectivity graph CG, and establishes adjacency relationships between these cells , reducing the continuous motion planning problem to a discrete graph searching problem. Their algorithm however contains some improvements, such as locally updating the connectivity graph at the critical positions, that make it more efficient than the previous algorithms . Recently, Sifrony and Sharir in [SiS] also exhibited an $O(n^2 \log n)$ algorithm for the same problem, which runs more efficiently when the obstacles are not too cluttered together.

Hopcroft, Joseph and Whitesides however, consider a different type of problem. They deal, not with a single line segment but with an assemblage of them. In [HJW1],[HJW2] they show that the problem of folding a carpenters rule is NP-complete although solvable in pseudo-polynomial time by dynamic programming. They are in [HJW3] concerned with the motion of linkages from the computational complexity point of view. A planar linkage consists of rigid rods that are free to rotate about joints at their endpoints. Each joint connects two or more rods and some joints are fastened to the plane. An interesting result is that a planar linkage , that can model a robot arm for example, can be constrained to stay in the interior of a bounded polygonal region by the addition of a polynomial number of new links.

We will now highlight some interesting results in a different instance of the movers' problem, the *separability of sets.*

1.2 - Separability of polygons

1.2.1 - Previous work on the movable separability of sets

One important subset in the wide class of problems involving motion is that of the *separability of sets* under different types of motion . The movable separability problems are primarily concerned with the idea of separating one or several geometrical objects away from a set and studying the possibility and the methods of doing so. Although it is difficult to precisely define the class of problems that come in the category of separability, they differ in general, from the typical collision avoidance and path planning problems encountered in robotics. The goal is to spread the objects far apart . In this case a precise final configuration is not really specified since one simply wants to detach parts of a "puzzle" allowing different types of motion such as *translation, rotation* , *sequential, simultaneous* , (some puzzles cannot be solved by sequential movement of their parts but by a simultaneous motion , each part having its own direction and velocity), and using geometrical properties of the bodies such as *convexity, monotonicity, star-shapedness* and so on. A study in breadth of the movable separability of sets is done by G. Toussaint and is clearly presented in [Tou1]. We will here present some problems in the plane and a few well established results, some classical, others more recent, before describing in section 1.2.2 the original problem that triggered the problem which is the subject of study of the present thesis.

Consider a set of n *isothetic* rectangles in the plane whose sides are parallel to the x and y axis. Consider the problem of translating the entire collection by some vector with the constraints that every rectangle is moved *sequentially* and that at no time during the process do we allow collisions to occur between a pair of objects. Guibas and Yao have shown that given n rectangles and a direction l, a translation ordering always exists and that it can be computed in O(nlogn) time. This also holds for the more general case of convex polygons,[GY] . Later Ottmann and Widmayer proposed a simpler algorithm with the same complexity to solve the same problem, [OW]. The more general types of problems consider other types of polygons and other types of motion besides simple translation and deal with the notion of interlocking of polygons, see for example Sack and Toussaint in [Tou2] [ST] and [TS] and Chazelle & al [Chai].

Some separability problems can be expressed as queries such as given a subset P' of a set P of convex elements, compute all the directions in which P' can be translated away from P , without colliding with the members of $P - P'$, or given an ordering on the polygons , find all directions of translation that admit this ordering. Refer to [MT]. A simple query is, for example, given an object in a set of convex polygons , what are all the directions of translation for this object to translate away from the set ? One way of solving this query involves the computation of an ordering for (possibly overlapping) intervals on the perimeter of a circle. The same query for the three dimensional case involves finding the union of (possibly intersecting) polygons on the surface of a sphere. In fact, the three-dimensional equivalent of a number of translation queries , involving visibility in the plane, involve solving problems in a non-euclidean space, namely on the surface of a sphere, [Man].

Recently, Battacharya and Toussaint proposed a linear algorithm for determining the translation separability of two simple polygons, once a triangulation is obtained, [BT]. Two polygons are said to be *separable under translation* if one of them can be translated an arbitrary distance in some fixed direction without intersecting with the other. Their algorithm uses the polygon triangulation algorithm of Tarjan and Van Wyck [TV], which runs in O(n log logn) time.

There is an interesting distinction to make between two types of motion in a set of objects. One is the *sequential* movement (that is one object is moved at a time while the others remain stationary) and the other is *simultaneous*. From there also derives the idea of *interlocking*. For example, the three quadrilaterals of figure 1.3 interlock under a sequence of translations or a sequence of rotations, however can be separated under simultaneous motion . The same observation can be made with any number of such quadrilaterals.

Consider the *monotonicity* property. A polygon P is said to be *monotone* or *monotonic*, in a direction d, if it can be partitioned into two subchains, such that for each subchain, the orthogonal projection onto a line parallel to d, perserves the

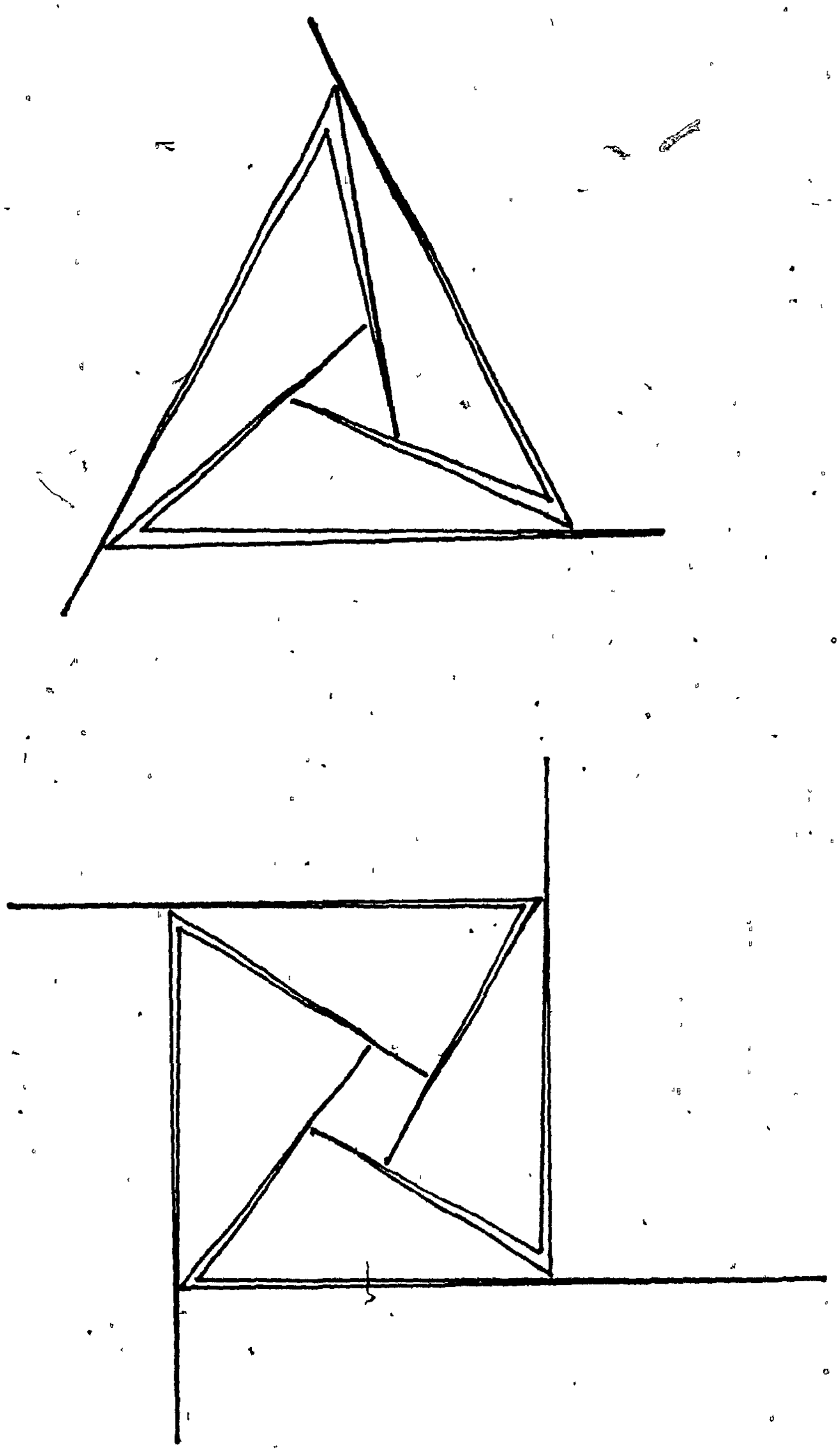


FIGURE 1.3

ordering of the points. Considering *monotone* polygons, ElGindy and Toussaint [TE] have proved the following theorem :

theorem : given two polygons P and Q *monotone* in the directions d and t respectively, then P and Q are separable with a single translation in at least one of the two directions $d + \pi/2$, $t + \pi/2$. And the direction of separability can be determined in O(n) time.

What about three monotone polygons, and four ? Notice that the quadrilaterals of figure 1.3 are also monotonic. Toussaint [Tou2] and Dawson [Daw1] have shown independently, that three monotone polygons can be sequentially interlocked, see figure 1.4, but that they are separable under simultaneous translations, and that four can interlock even under simultaneous motion, [Daw1], see figure 1.5 .

One class of polygons that present interesting properties are *star-shaped* polygons. A polygon P is said to be *star-shaped* if it contains a convex region, called the *kernel* (possibly reduced to a single point), from which no part of P is hidden from a guard, if he were to stand on any point inside the kernel. It is a well known result that two star-shaped n-gons are always movably separable with a single translation and that a direction for separating them can be determined in linear time. This is due to the linear time computation of the kernel of Lee and Preparata, [LP]. The above statement suggests that two star shaped polygons, P and Q, can be separated by translating both of them simultaneously in some pairs of direction with respect to an arbitrary fixed point in the plane. In fact it is sufficient to guarantee that the relative motion between P and Q is correct. Let K(P) and K(Q) be the respective kernels of P and Q. Let a and b be any pair of

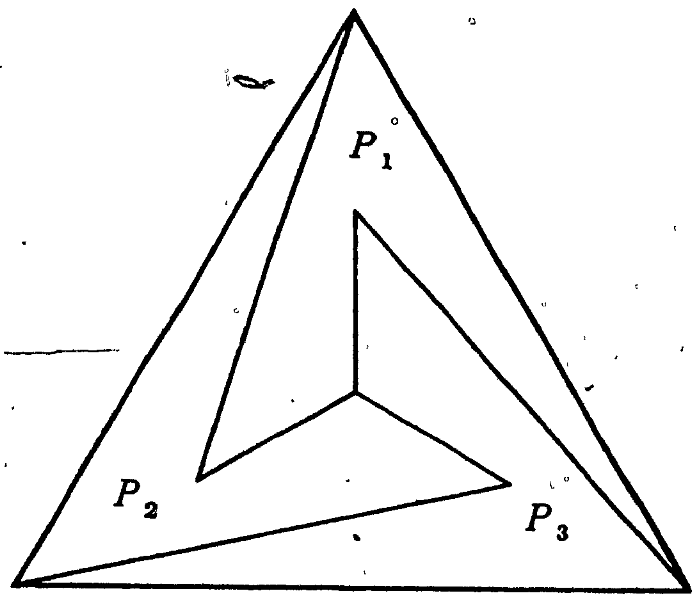


FIGURE 1.4

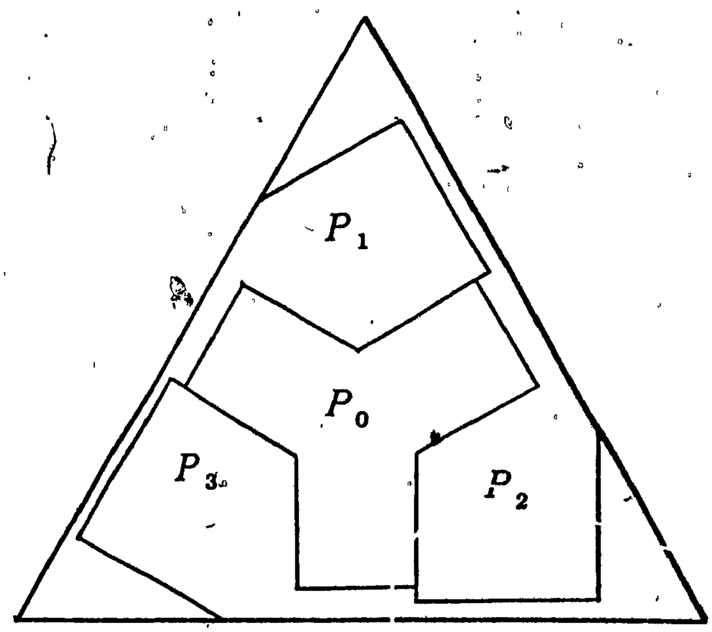


FIGURE 1.5

points in the plane such that the line $L(a, b)$ going through a and b, intersects $K(P)$ and $K(Q)$. Let x be any reference point in the plane, and consider the vectors xa, xb and ab, in figure 1.6 . We can now see that if we translate P and Q in directions xa and xb with velocities proportional to the magnitudes of xa and xb respectively, the correct relative motion between P and Q is maintained. Different pairs of points (a, b) only change the relative velocity of separation.

Dawson has proved in [Daw2] that in any finite collection of three or more convex bodies in the plane, intersecting at most in their boundaries, there exists at least three elements which are movable. This however does not lead to easy generalization in spaces of arbitrary dimensions, as the example of the twelve tiles in figure 1.7 demonstrates it. These convex objects are interlocked under any type of motion. For star-shaped polygons however, Dawson has shown that any collection of them are always separable under simultaneous translation.

Theorem :

Let $P = \{ P1, P2, \ldots , Pn\}$ be a set of *star-shaped* polygons. If there exists a set T of translations $T = \{ T1, T2, \ldots , Tn\}$, $Ti = (D,V)$ (direction and velocity), such that under T every pair (Pi, Pj) $i,j = 1,2,\ldots,n$, of polygons is separable, then P is separable under *simultaneous* translation.

If we translate each Pi by the vector Ti we easily see that the *relative* motion between every pair of polygons is maintained. Refer to figure 1.8.

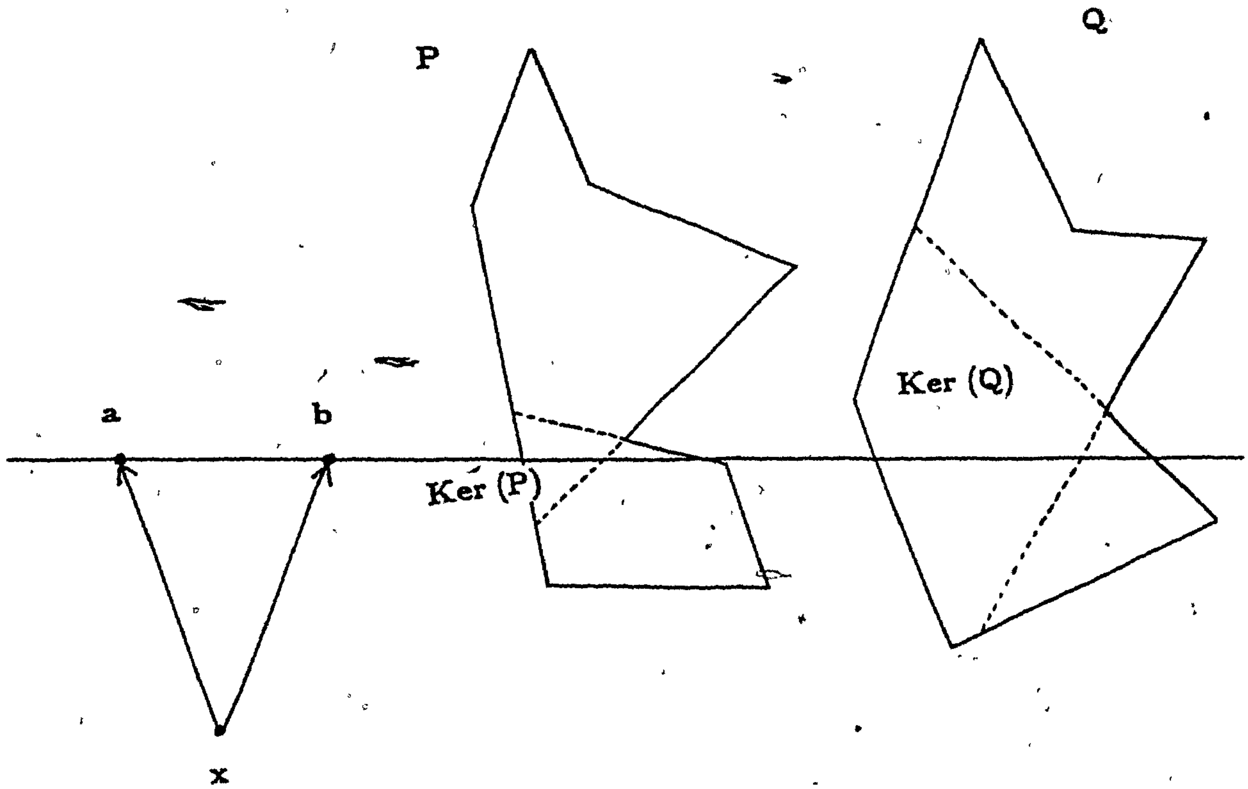


FIGURE 1.6

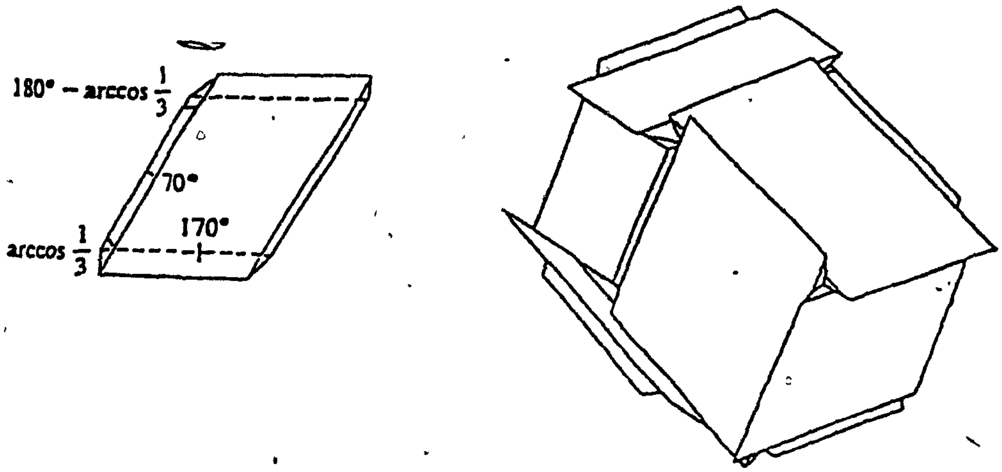


FIGURE 1.7

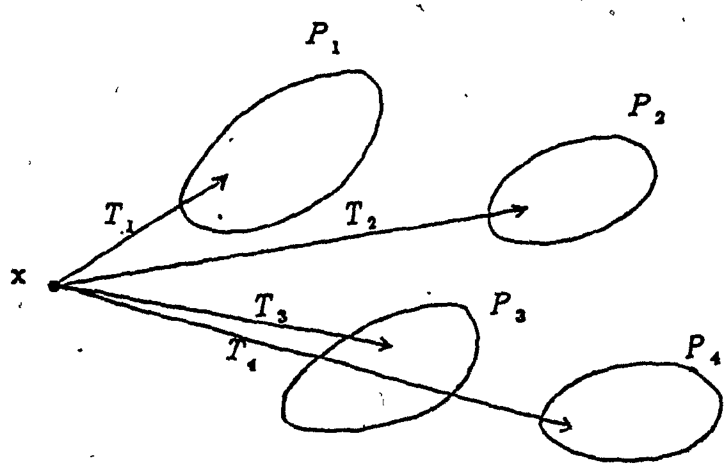


FIGURE 1.8

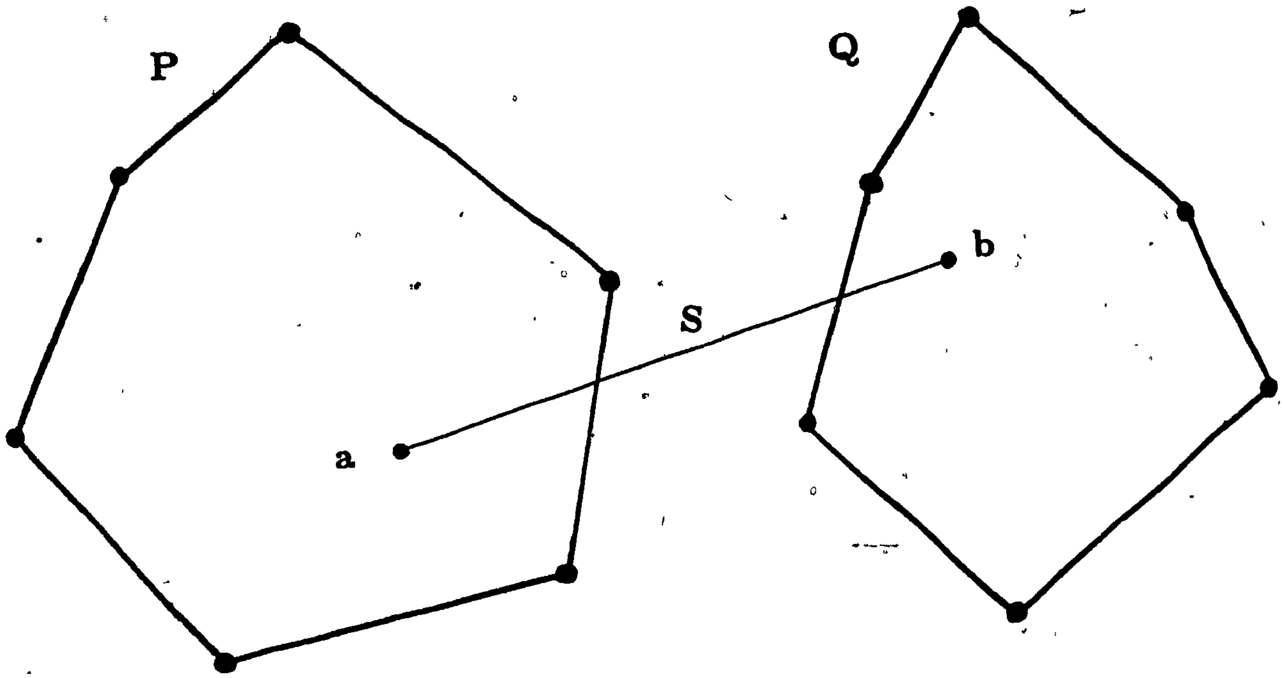


FIGURE 1.9

1.2.2 - Stating the original problem

Let there be two star shaped polygons SP and SQ with kernels K(SP) and K(SQ). And let there be two points $a \in$ Ker (SP) and $b \in$ Ker (SQ). The vector ab determines a *direction* of separation for SP and SQ but also a *velocity*. We are interested in finding all the pairs of points p $\in$ Ker(SP) and q $\in$ Ker(SQ) such that $\| pq \| = \| ab \|$. In other words what regions inside the two kernels determine a given velocity of separation of the two polygons ?

1.3 - Problem Statement

Stated more generally and independently, the problem is the following : Given two convex polygons P and Q and a line segment $S = [\, a,\ b \,]$ of length r, calculate the regions inside P and Q if they exist , such that S can be placed in P and Q with the constraint that the endpoint a lies within the boundaries of P and the endpoint b within those of Q, see figure 1.9 for an illustration. What is the reachability region for endpoints a and b ?

1.4 - The approach taken and the structure of the remaining chapters

To solve this problem, we will first compute the unreachability region for the two polygons. This involves computing the intersection of circles of equal radius about each of the vertices of the polygons. We first present in chapter 2 an existing algorithm to compute this intersection. Then, in chapter 3 ,we show how this problem can be solved by an algorithm that has the advantage of being generalisable to computing the intersection of circles of arbitrary radii. In chapter

4 we compute the reachability regions in the two polygons. For this we have to calculate the intersection of two convex figures that may have arcs of circles as part of their boundaries. And in chapter 5, we exhibit the complete algorithm for solving the problem stated in the previous section. Finally we close with open problems in the last chapter.

CHAPTER 2

2.1 - Brown's algorithm

In [Brow], Kevin Brown exposes an algorithm for intersecting n circles of arbitrary radii, which we describe very briefly, for its beauty and simplicity. The algorithm runs in O(nlogn) time.
Brown uses an involutory inversion transform, which maps a circle passing through the center p of the inversion, to a line that doesn't pass through p and vice-versa. It also transforms any sphere that passes through the center of inversion to a plane not passing through it. Consider figure 2.1. If the n circle in the plane share a common boundary point P , we choose P to be the center of the inversion transform and computing the intersection of the n circles will thus be equivalent to computing the intersection of the half-planes which can be done in O(nlogn) time, see figure 2.2.

In the general case however, when the circles don't intersect at a common point we do the following . Let the circles lie on a plane L . Choose an arbitrary point P not in L. For each circle c, there is a *unique* sphere that passes through point P and that intersects the plane L at circle c. We can thus represent the n discs in the plane L by n balls whose spherical boundaries share a common point P and refer to figure 2.3. Inversion about point P transforms the n spheres to n planes. The intersection of the n discs is therefore represented by the intersection of the n half-spaces, which can be computed in time O(nlogn), using Preparata and Muller's algorithm, [PM].

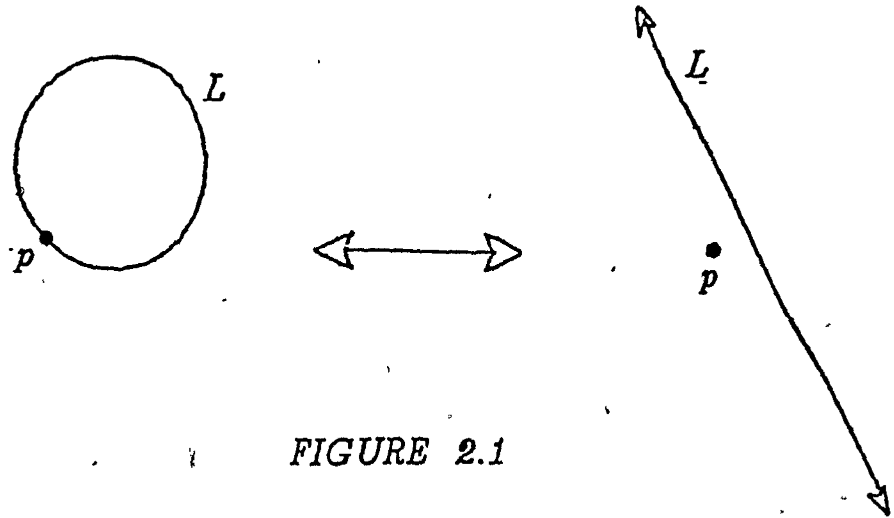


FIGURE 2.1

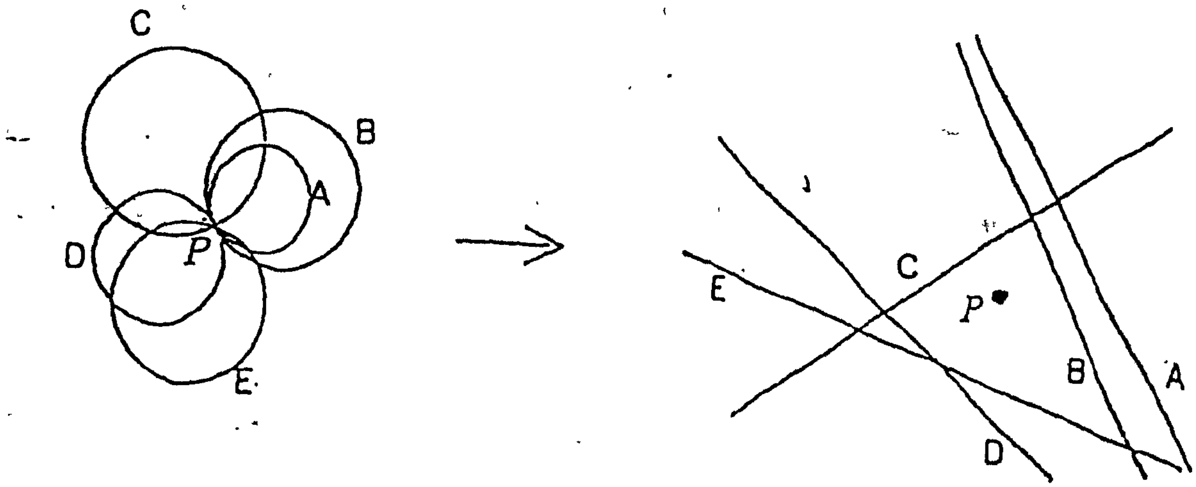


FIGURE 2.2

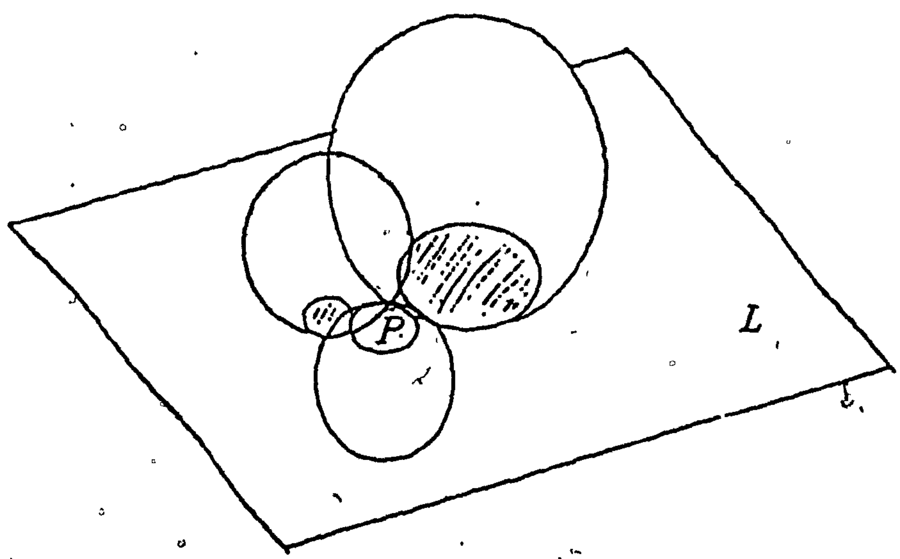


FIGURE 2.3

2.2 - Melville's algorithm

2.2.1 - Problem statement

Melville, [Mel], in the context of finding the minimum spanning circle, (msc), for a set of points in the plane, encounters the following problem :

Given a convex polygon $P = \{ p_1, p_2, \ldots, p_n \}$ and a radius r, what is the region formed by taking the intersection of n circles of radius r, about each of the n vertices of P ?

It is certain that a non-empty intersection exists since the radius r is not given but determined by the distance from the centroid of P to the furthest vertex of P.

2.2.2 - Overview of the rolling algorithm

The *rolling* algorithm is an approximation algorithm which computes a convex region which is certain to contain the center of the msc. The area of this region may be made as small as desired allowing the location of the center to be approximated more and more accurately. Let c^* and r^* be the exact center and radius of the msc. The idea is the following :

- choose an initial center c_0, taken to be the *area centroid* of the input polygon . The area centroid of a convex figure has the following physical interpretation : if the figure were to be cut out of sheet metal, it would balance on a pin point located under the area centroid. We can triangulate the convex polygon and take the centroid to be the weighted sum of the centroids of each triangle.

- compute the maximum distance from c_0 to a vertex , and take this distance to be the radius r_0 of a first spanning circle.

- then calculate the intersection region Dr_0 , of circles of radius r_0 about each of the polygon vertices. We repeat this process by choosing c_1 to be the area centroid of Dr_0, compute r_1 to be the maximum distance between c_1 and the polygon vertices and compute the next region Dr_1 which is nested in Dr_0.

The complete algorithm consists of several iterations of the above procedure and generates a decreasing sequence of radii :

$r_0 > r_1 > r_2 > \dots >= r^*$

and a corresponding sequence of nested convex regions :

$Dr_0 >= Dr_1 >= \dots >= Dr^*$

The interesting part in this algorithm is to show how to compute Dr_i in linear time. We will in the next section describe in detail the calculation of Dr_i, and show that it can be obtained in time linear to the number of the input (convex) polygon vertices.

2.2.3 - Computing Dr_i in linear time

Figure 2.4 shows the Dr region for some $r > r^*$. The idea is the following : imagine that the polygon vertices are poles and that a metal ring of radius r encircles the polygon. The ring is free to roll around the poles. As the ring makes one trip around, the center of the ring will trace out exactly the perimeter of Dr. Let $r >= r^*$ be an upper bound on the radius of the msc . A vertex x of the polygon is a *contact point* at radius r means that there is a radius-r spanning circle through x. We want to identify quickly the radius-r contact vertices. Therefore computing Dr will produce them in counterclockwise order, as a sub-sequence of the input sequence. We therefore need a sufficient condition for discarding points that do not contribute to Dr i.e. those that cannot be contact points. Let p_i, p_{i+1}, p_{i+2} be three consecutive vertices of the convex polygon P.

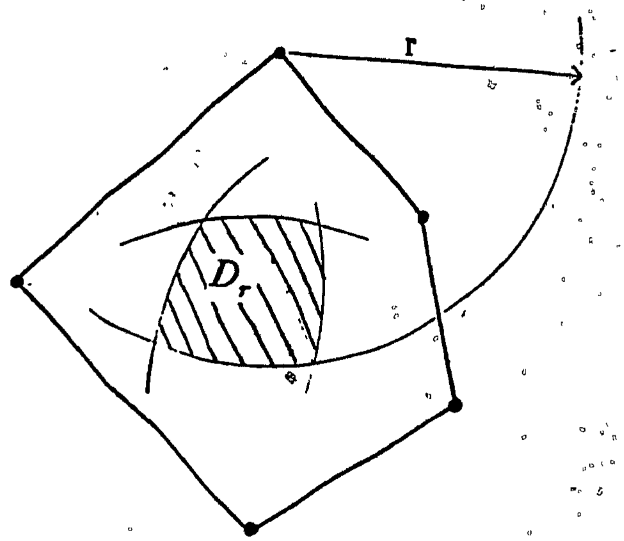


FIGURE 2.4

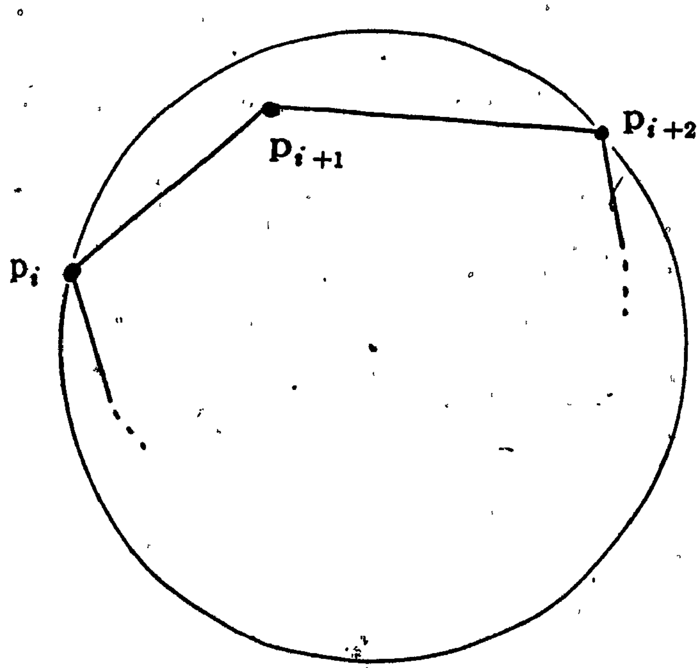


FIGURE 2.5

Let *circ* (r, p_i, p_{i+2}) denote the radius-r circle through p_i and p_{i+2}. Notice that there may be two such circles. We take the one which has its center on the opposite side of the line through p_i and p_{i+2} as p_{i+1} is. Then p_{i+1} is not a radius-r contact point. Refer to figure 2.5. Intuitively, the curvature of the circle is greater than the curvature of the polygon boundary between vertices p_i and p_{i+2}, and p_{i+1} must fall inside. Also, *circ* (r, p_i, p_{i+2}) need not be a spanning circle of the entire polygon.

The algorithm to compute Dr is as follows : Let c be the center of a spanning circle. Find the furthest vertex to c , call it *A1* and let r be this distance and therefore the radius of a spanning circle C. We want to find a pair of consecutive contact points at radius r. C goes through *A1*. To find the next contact point *A2*, imagine C swinging about vertex *A1*, clockwise. The first vertex touched by C will be *A2*. Let s be the center of the radius-r circle through *A1* and *A2*. The wedge , or the angle formed by (c, *A1*, s) is the smallest possible. Refer to figure 2.6.

Let *succ* (p_i) be the successor of vertex p_i in clockwise order. To find the next contact point, we could repeat the procedure, using *A2* as a pivot and choose among *succ*(*A2*), *succ*(*succ*(*A2*)) etc. the one that touches first the swinging circle. It would take $O(n^2)$ time to find all the contact points. Instead, Melville describes a linear algorithm that yields all radius-r contact points after one clockwise trip around the polygon. The idea is the following.

We keep track of the clockwise successor of each vertex , in an array called *succ*. Thus *succ* $[p_i]$ is p_{i+1} in the original polygon. There are also two stacks **S** and **DP**. **S** contains the polygon vertices p_i such that an arc of Dr is on the radius-r circle about p_i. **DP** contains the vertices of Dr, that is the center of the circle that goes through two consecutive vertices of **S**. The stack **S** first contains the two contact points *A1* and *A2*. In a loop, the algorithm keeps adding a point

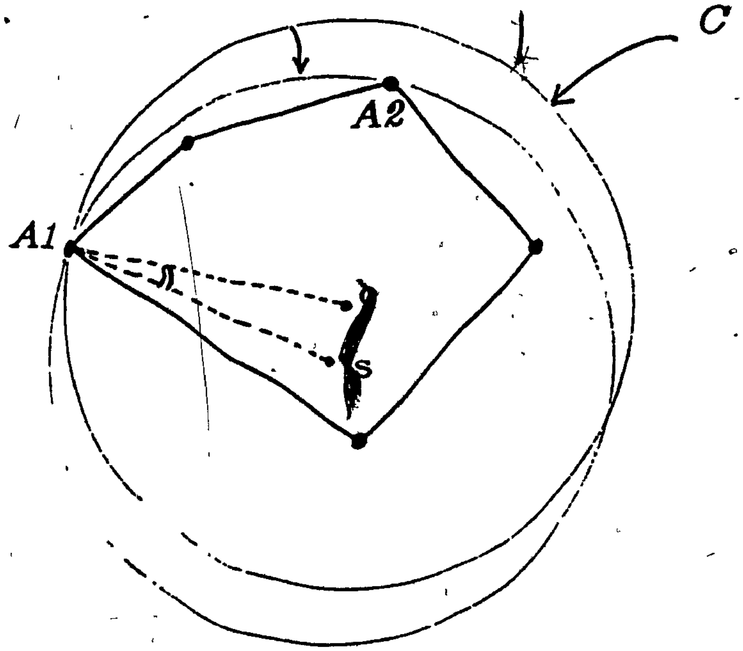


FIGURE 2.6

p_{i+1} to the stack S and the center of *circ* (r, p_{i-1}, p_i) to **DP** as long as p_{i+1} is inside *circ* (r, p_{i-1}, p_i) and that there are still vertices to be visited.

But if p_{i+1} is outside *circ* (r, p_{i-1}, p_i) it means that p_i is inside *circ* (r, pi-1, p_{i+1}) and that p_i is not a contact point and therefore can be discarded, so we pop the stacks **S** and **DP** and update the list of successors by letting p_{i+1} be the successor for p_{i-1}. We then retreat counterclokwise, popping the stack, until $succ(p_k)$ is again inside *circ* (r, p_{k-1}, p_k). The retreat must terminate because at worst , we will back up to *circ* (r, *An-1*, *An*) which is a spanning circle. Once *succ* (p_k) comes back inside, we again start advancing clockwise. Of course , *succ* (p_k) is not necessarily a contact point and we may later back up over this current *succ* (p_k).

We now give a pseudo Pascal description of the algorithm :

Input :

- Vertices of a convex polygon P = { $p_1, p_2, \ldots, p_n$ } , stored in an array.
- a radius r (we suppose that we have already computed this radius as being the distance of the centroid of P to the furthest vertex of P).

Output :

Intersection region Dr, of circles of radius r , about each of the n vertices of P.

Data structure :

- P is stored in an array [1 .. n] of vertices .
- *Succ* is an array [0 .. n] of integer. It stores a circular linked list of indices into P, the active polygon vertices.
- S is the stack of contact points i.e. the vertices p_i such that an arc of Dr is on a radius-r circle about p_i .
- DP is the stack containing the vertices of Dr, that is the center of the circle that

goes through two consecutive vertices of **S**.

Algorithm :

```
initialize :
    for i := 0 to (n - 1) do succ [ i ] <--- i + 1 ;
    succ [n] <--- 1 ;
    S <--- ( A1, A2 ),  {suppose we have found these first two
                                  contact points as described earlier}
     k <--- 2,  index  of starting vertex.

While s[k] < > s[1] do   { when s[k] = s[1] we have come to the first
                                   contact point and we stop }
     begin

       while P[ succ [ s[k] ] ] inside circ(r, s[k-1], s[k]) and s [ k ] < > s [ 1 ] do
           begin
             push onto DP, the center of circ( r, s[k-1], s[k] )
             push onto S, succ [ s[k] ]
             k <--- k + 1
           end;

       if s[k] < > s[1] then

         begin
           z <--- succ [ s[k] ]
           while P[z] outside circ (r, s[k-1], s[k]) do
               begin           { start backtracking }
                 succ [ s[k-1] ] <--- z
                 pop S
                 pop DP
                 k <--- k - 1
               end
         end
     end.
```

2.2.4 - The analysis

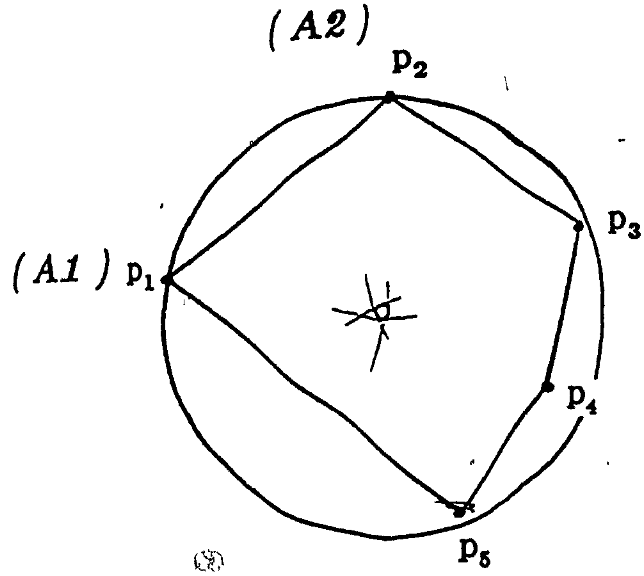
The correctness is proved by the fact that if p_{i+2} is outside the circle *circ* (r, p_i, p_{i+1}) then p_{i+1} does not contribute to Dr since it will be inside *circ* (r, p_i, p_{i+2}). We now want to show that the algorithm requires O(n) time to select the radius-r contact points from a convex n-gon. When advancing clockwise, the algorithm always stacks a point it has not considered before. Since there are at most n vertices, the total cost for advancing is O(n). Now consider the retreating action , in which the algorithm backs up past points it has already placed on the stack. The test to determine whether the algorithm should backup is O(1), since it requires only checking that one point is inside or outside of a known circle. Whenever the algorithm backs up, it eliminates a point as a possible contact point. Since at most O(n) points may be eliminated, the total cost of retreating is bounded by O(n).

2.2.5 - An example

Consider figure 2.7. Let *A1* be p_1. The first vertex *A2*, in clockwise order, hit by the swinging circle about the pivot p_1 will be p_2. So p_1 and p_2 will be the first two elements of **S**. We then advance clockwise and test whether p_3 is in *circ* (r, p_1, p_2). Yes it is , so we push (p_1,p_2) in **DP**, push the vertex p_3 in S and go to the next vertex which is p_4. Let ((p_j,p_k) be short for *circ*(r, p_j , p_k)). Is p_4 in (p_2,p_3) ? Yes, and we push (p_2,p_3) in **DP**, push p_4 in **S** and go to see p_5. At this stage the stacks **S** and **DP** contain (from bottom up) : p_1, p_2, p_3, p_4 and (p_1,p_2), (p_2,p_3) respectively. We continue, is p_5 in (p_3,p_4) ? No, so we backtrack and pop p_4 from **S** and (p_2,p_3) from **DP**. Is p_5 outside of (p_2,p_3) , no so we stop the backtracking and go forward . Is p_5 in (p_2,p_3), yes so we push p_5 in **S** and (p_2,p_3) in **DP**. Then, we test : is p_1 in (p_3,p_5) ? (notice that since we have popped p_4 from **S**, the successor of p_3 is not p_4 anymore but p_5). Yes it is , so we push p_1 in **S** and (p_3,p_5) in **DP**. We test the next one : is p_2 in (p_5,p_1), yes, so we must also push (p_5,p_1) in **DP** and we stop since p_1 is the first element of the stack. The contact points are therefore p_1,p_2,p_3,p_5 in **S** together with the vertices of Dr in **DP**.

2.2.6 - Comments on Melville's algorithm

An important thing to notice in Melville's algorithm is that the radius of the circle is not given but rather is a function of the input polygon. The fact that a spanning circle exists guarantees a non-empty region Dr. Therefore, his algorithm



Succ : active vertices

0	1	2	3	4	5
1	2	3	~~4~~ 5	5	1

S : stack of contact points

1	2	3	4	5	6
1	2	3	~~4~~ 5	1	

DP : vertices of D_r

1	2	3	4	5	6
(1,2)	~~(2,3)~~ (2,3)	(3,5)	(5,1)		

Integer i in array stands for vertex p_i

FIGURE 2.7

does not really answer the query: given a convex polygon *and* given a radius r, compute the intersection of the circles about the polygon vertices, if it exists, and answer no if it doesn't. Also, his algorithm may involve backtracking. Finally, an important restriction on this method is that it does not easily extend to the general case of computing the intersection of circles of arbitrary radii, about polygon vertices.

The algorithm we propose in the next chapter, is also linear in its time complexity but has the advantages of taking as input any convex polygon and any radius, reporting if no intersection exists and yielding it, if one does. Moreover, it can also be easily modified to handle the general case, that is when the circles about the vertices have each their own radius.

CHAPTER 3

3.1 - Introduction

Given a convex polygon $P = \{ p_1, p_2, \ldots , p_n \}$ and a line segment $A = [a_1, a_2]$ of length r, with the constraint that a_1 be inside P, we would like to compute the region of the plane, that is not reachable by a_2, if a_1 remains in P. Let CH be the convex hull of the n circles of radius r about each of the n vertices of P. The region of the plane outside CH is unreachable by a_2. There may also be such a region inside CH. If it exists, we prove that it is the intersection of the n circles and give a linear algorithm to compute it. Also we will show how our algorithm for computing this intersection can be generalized to the case where each circle, around a vertex, has its own radius.

3.2 - Intersection of circles of equal radii

3.2.1 - Preliminary results

3.2.1.1 - The line segment case

Before computing the unreachability region for the polygon case, let us solve a simpler version of the problem :

Question :

Consider a static line segment $S=[s_1, s_2]$ of length l in the plane , and a line segment $A=[a_1, a_2]$ of length r , such that a_1 is constrained to remain on S and can slide between s_1 and s_2, while a_2 is free to rotate about a_1. See figure 3.1. We want to determine the unreachable region U for a_2, that is the locus of points, such that : for any point p of U, there is no point q on S such that d(p, q) = r, where d(p, q) denotes the euclidean distance between points p and q .

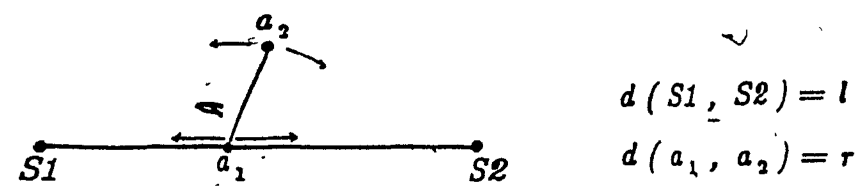


FIGURE 3.1

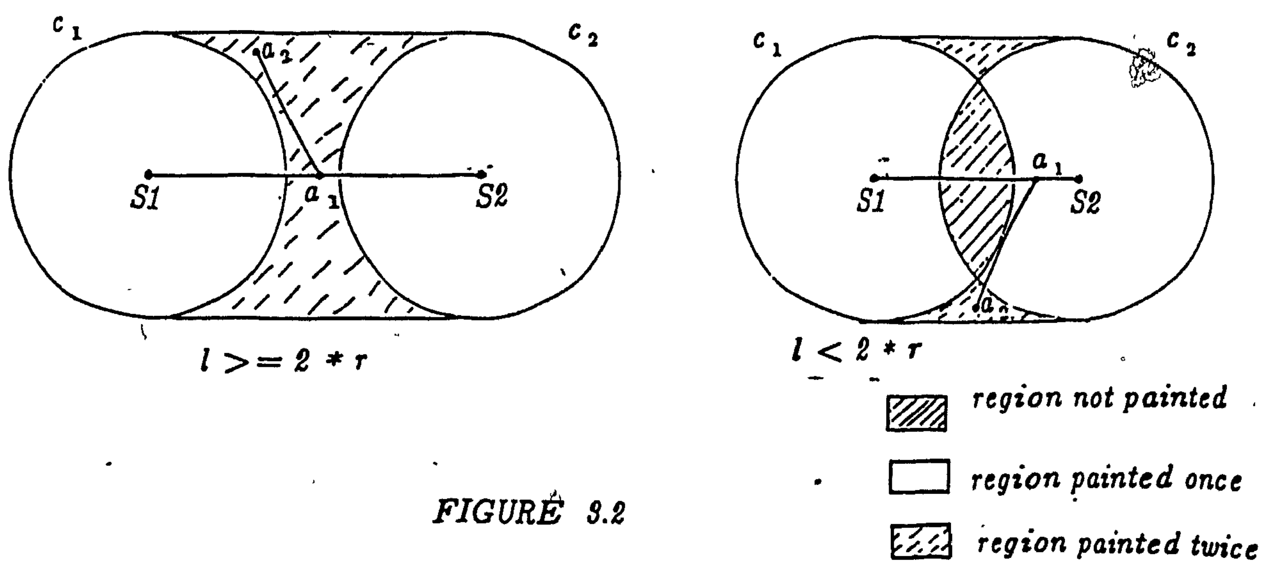


FIGURE 3.2

Idea :

Consider the convex hull of the two circles c_1 and c_2 of radius r and about s_1 and s_2 respectively. It has the shape of an *éclair*. See figure 3.2. The region of the plane "outside" CH(c_1,c_2) is obviously an unreachable region for a_2. But there may also be such a region inside CH(c_1,c_2).

* If l >= 2 * r , then the entire *éclair* is accessible to a_2. As a proof imagine the following: Place a_1 on s_1. The accessible points to a_2 are all the points on the perimeter of c_1. Imagine c_1 to be a circular *painting brush.* Translate a_1 from s_1 to s_2. The accessible points also translate from the boundary of c_1 to that of c_2, thus painting the entire *éclair*. Notice that the *hourglass* region inside the *éclair* but outside c_1 and c_2 will be painted twice.

* If l < 2 * r , then there will exist an unreachable region for a_2 which will be the strict interior of $c_1 \bigcap c_2$. It is easy to see that this region remains unpainted, and the two curved triangular regions, inside the *éclair* and outside both c_1 and c_2, will be painted twice.

3.2.1.2 - The polygon case

We will first state the problem, then prove a few results, before presenting, in the next section, the algorithm.

Problem statement :

Consider a convex polygon P as defined previously, and a line segment A = $[a_1, a_2]$ of length r for which a_1 is constrained to remain *inside* or *on* the boundary of P. What is the unreachable region for a_2 ?

We are interested in finding the unreachable points inside CH (c_1, c_2, ... , c_n). From now on we will assume that all two adjacent circles intersect. The case of two disjoint adjacent circles will be handled by a simple test in the final algorithm.

Lemma 1 :

The region, if it exists, that is outside each of the n *éclairs* of the n edges and inside P, is reachable by a_2, a_1 remaining inside P.

Proof :

Take any point p inside this region. Draw a half-line L from p such that L's direction is perpendicular to an edge e of P and cuts e at a point x. Refer to figure 3.3. Let y be the intersection point of L with the "inner" *éclair* boundary. Then d (p, x) > d (y, x) = r. Therefore, by placing a_2 at p, we can always find a point a_1 on the line segment (p, x) such that the segment (a_1, a_2) is inside P. QED

Lemma 2 :

The unreachable region for a_2, inside CH(c_1, c_2, ... , c_n), is the strict interior of the intersection of all the c_i's, $i = 1, 2, \ldots, n$.

Proof :

Let U be the unreachable region *inside* CH(c_1, c_2, ... c_n) and CH be CH(c_1, c_2, ... , c_n).

$* \; (\bigcap_{i=1..n} c_i) \subseteq U$:

for every point p in ($\bigcap_{i=1..n} c_i$) , there is no point q on the boundary of P such that d(p, q)= r. So there is no q in the interior of P such that d(p, q) = r, therefore p is in U.

$* \; U \subseteq (\bigcap_{i=1..n} c_i)$:

for every point p in U, there is no point q in P such that d(p, q)=r, so there is no point q on the boundary of P such that d(p, q)=r. Since p is inside CH, and that

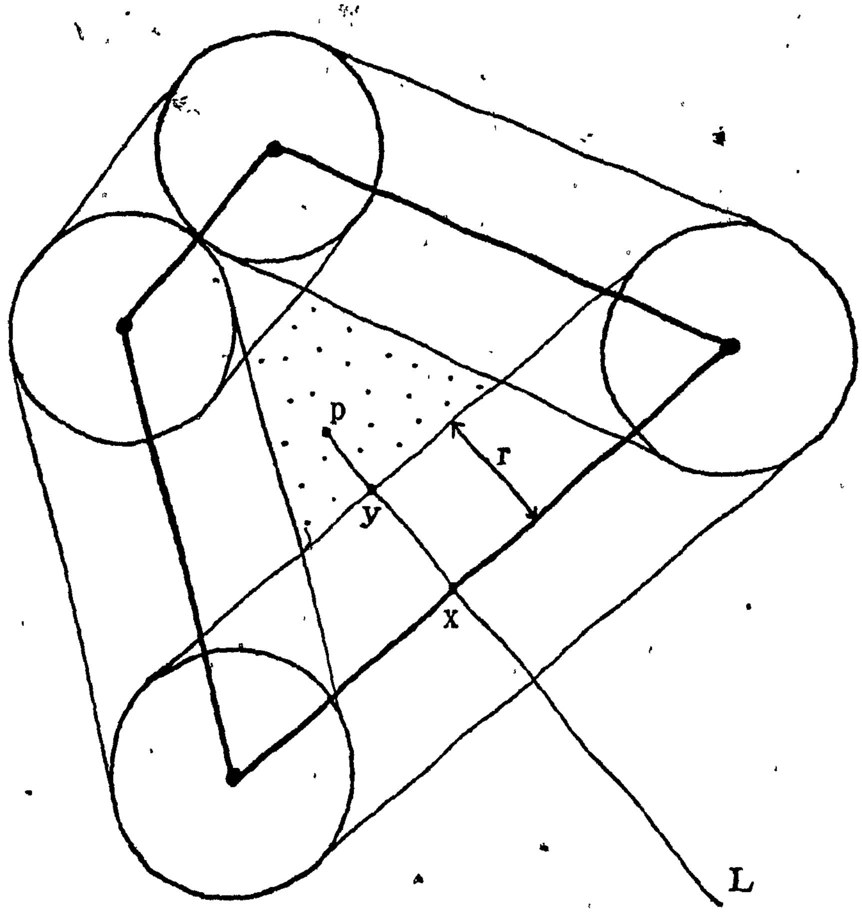


FIGURE 3.3

it is inside all n lunes, therefore p is in ($\bigcap_{i=1\ n} c_i$). QED

Definition 1 :

Let P $= \{ p_1, p_2, \dots , p_n \}$ be a convex polygon, where the p_i's are the vertices specified in terms of cartesian coordinates and given in counterclockwise order. Let c_i be the circle about vertex p_i of radius r. See figure 3.4 .

* The *lune* of two circles c_i and c_j, noted lune (c_i, c_j), is $c_i \bigcap c_j$.

* The *lune bissector* is the line segment bissecting the lune and joining the two intersection points that are on the circle boundaries.

* The lune of edge $e_i = (p_i, p_{i+1})$ is lune(c_i, c_{i+1}). The *lune head*, lh_i , of edge e_i , is the endpoint of the lune bissector which is to the left of (p_i, p_{i+1}). Recall that P is given in counterclockwise order. The *lune tail*, lt_i , is the other bissector endpoint.

Good and bad arcs :

We will draw two types of arcs between lune heads according to their relative position. Consider figure 3.5 . By convexity of P, p_{i+2} must be in the region of the plane that is to the left of (p_i, p_{i+1}) and of (p_1, p_2) and to the right of (p_{i+1}, p_1). This region may be bounded or not. The angle $< p_{i+2}, p_i, p_{i+1} >$ is in the range $]\ 0\ , \pi\ [$ and the lune bissector of $(p_{i+1}, p_{i+2}) = e_{i+1}$ makes with $(p_i, p_{i+1}) = e_i$, an angle in the range $]\ \pi/2\ , 3 * \pi/2\ [$.

We draw a *good arc* on c_{i+1}, if lh_{i+1} is on the arc (lh_i, y) (going counterclockwise).
We draw a *bad arc* on c_{i+1}, if lh_{i+1} is on the arc (x, lh_i) (going counterclockwise).

Angle between two consecutive good arcs :

Refer to figure 3.6 . Suppose we have two adjacent good arcs on c_{i+1} and c_{i+2}. We call the angle between two good arcs a *closed angle*. A good arc joining

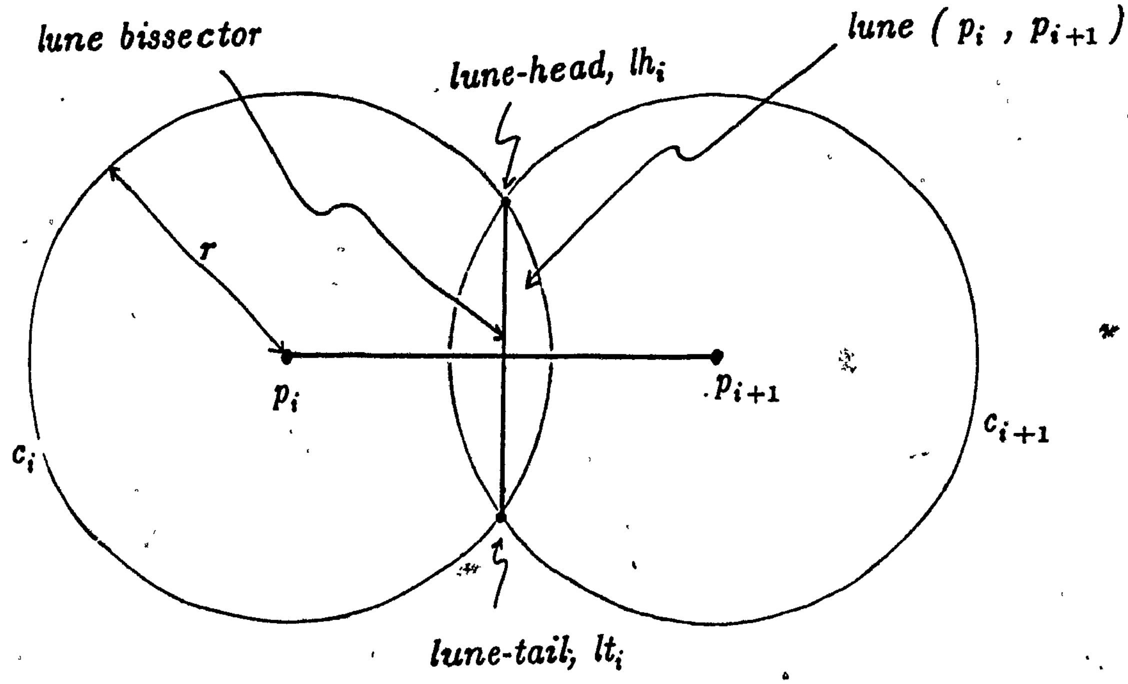


FIGURE 3.4

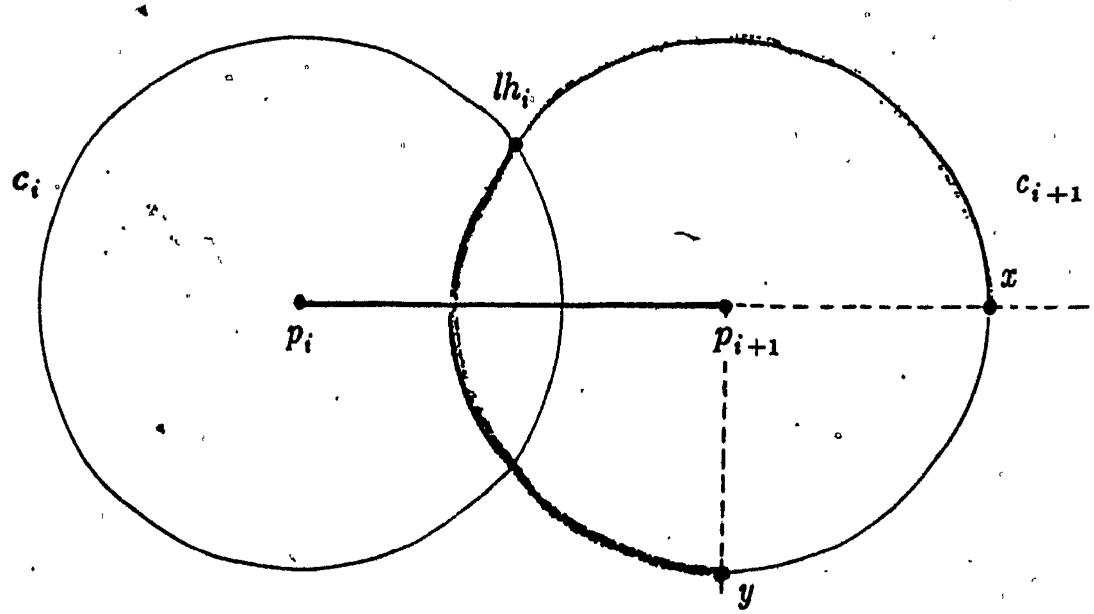
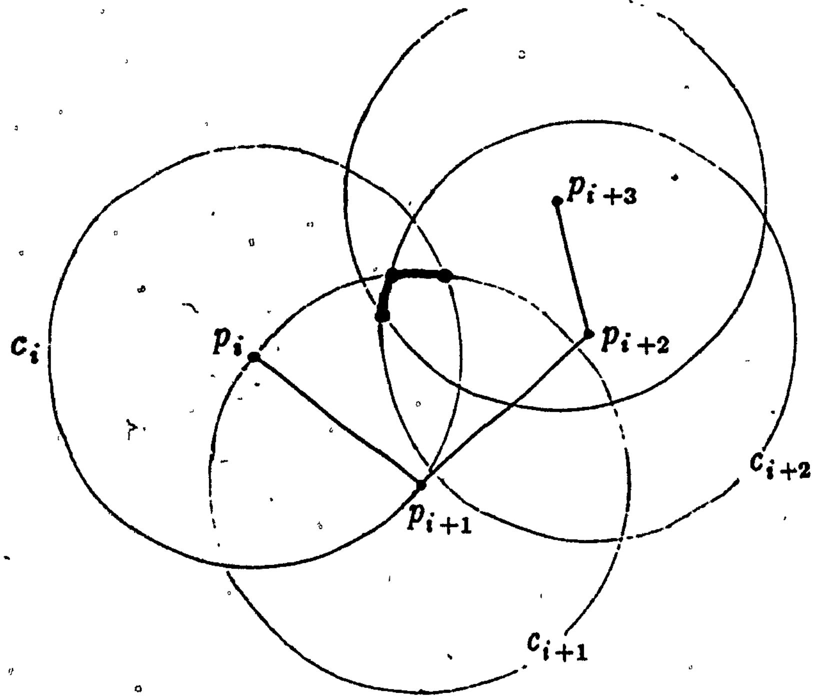
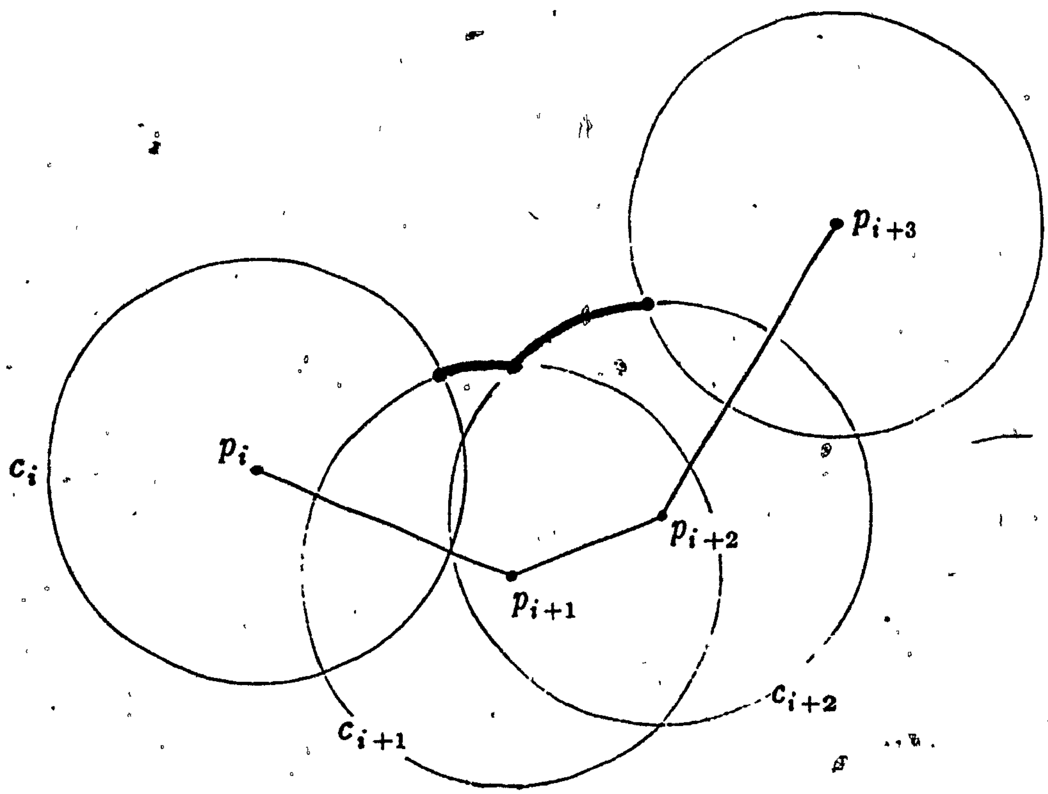


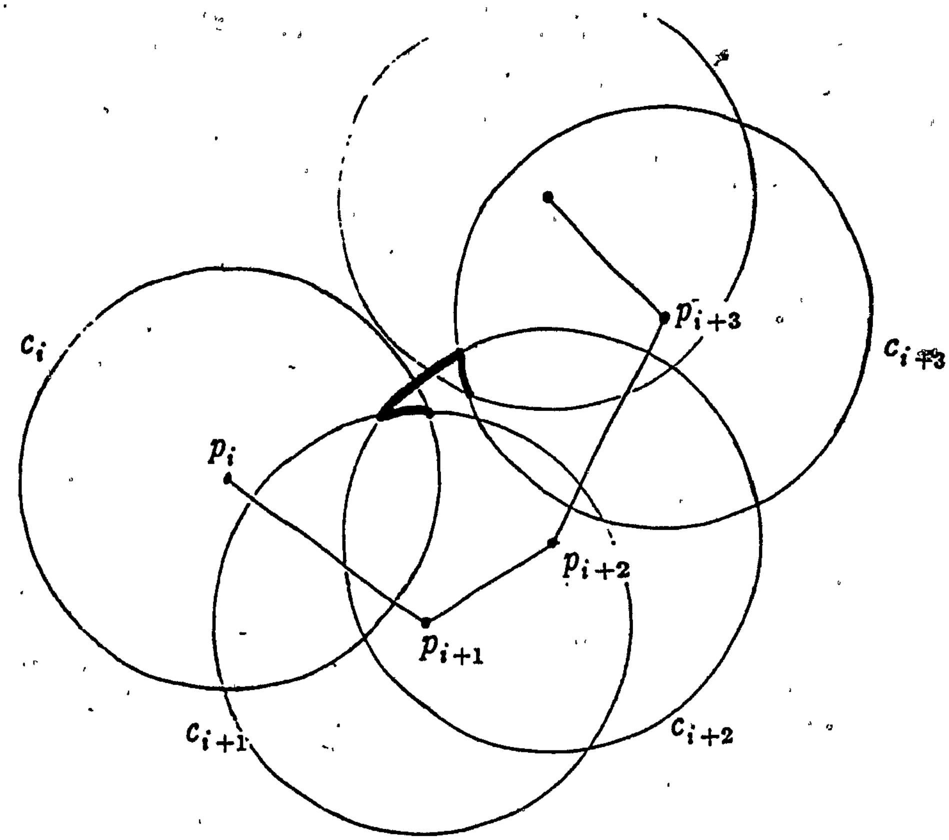
FIGURE 3.5



a – *closed* angle between two good arcs



b – *open* angle between two bad arcs



c — *semi-open* angle between a good and a bad arc

FIGURE 3.6

lh_i to lh_{i+1} is to the right of the line segment $[lh_i, lh_{i+1}]$.

Angle between two consecutive bad arcs :

Suppose we have two adjacent bad arcs on c_{i+1} and c_{i+2}. We call the angle between two bad arcs an *open angle*. A bad arc joining lh_i to lh_{i+1} is to the left of the line segment $[lh_i, lh_{i+1}]$.

Angle between a good and a bad arc :

We call the angle between a good and a bad arc a *semi-open angle*. The position of an arc, relative to the line segment joining its two endpoints, changes in going from one arc to another of a different sort.

Lemma 3 :

The boundary of the intersection of n circles, if it exists, *cannot* consist of any bad arcs.

Proof :

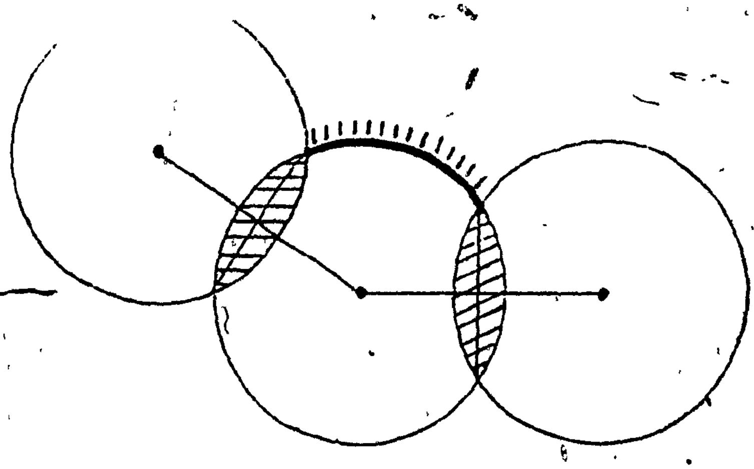
The intersection of n circles in the plane is a convex region. Having one or several bad arcs involves open or semi-open angles thus violating the notion of convexity, i.e. we could find a pair of points inside the region such that the line segment joining them is partly outside it. QED

We define four types of relation between two adjacent lunes: *disjoint* lunes, lune *touching* lune, lune *inside* lune and *lili-pad*. Refer to figure 3.7 . We define a *pattern* to be the closed sequence of arcs linking adjacent lune heads.

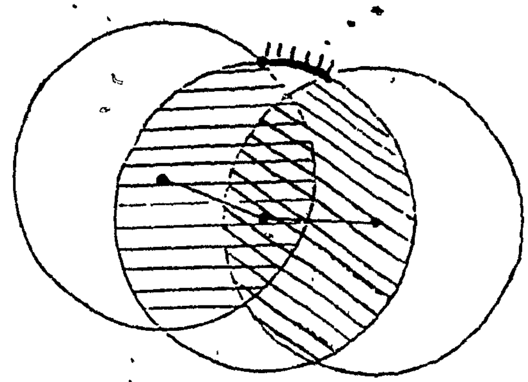
Lemma 4 :

A pattern composed of only good arcs is *simple*.

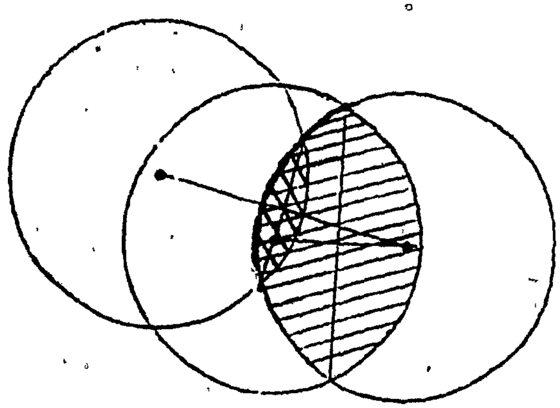
FIGURE 3.7



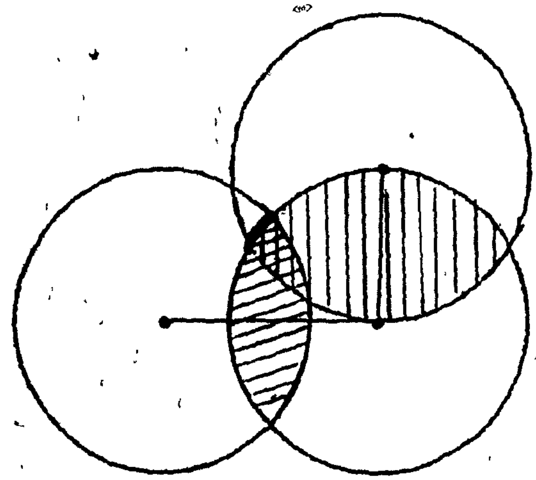
disjoint lunes



lune touching lune



lune inside lune



lily-pad

Proof :

No two non-adjacent arcs intersect: suppose that there are two non adjacent arcs that intersect. Since the pattern is a continuous path and a closed curve by construction, consider the situations in figure 3.8 :

obviously, we cannot have 1 because it is not one single closed curve. Also we cannot have 2. because at points a and b there is a change in the curvature direction which cannot happen with arcs of the same nature. Remains 3 : every vertex of the pattern is a lune head. To every lune head corresponds a lune bissector and an edge of the polygon . Going counterclockwise, the sequence of lune heads between a and b and that between c and d correspond to two polygonal chains that "cover" the same *portion* or *cone* in the plane.In other words , the polygon corresponding to this pattern is not simple, which contradicts our hypothesis.

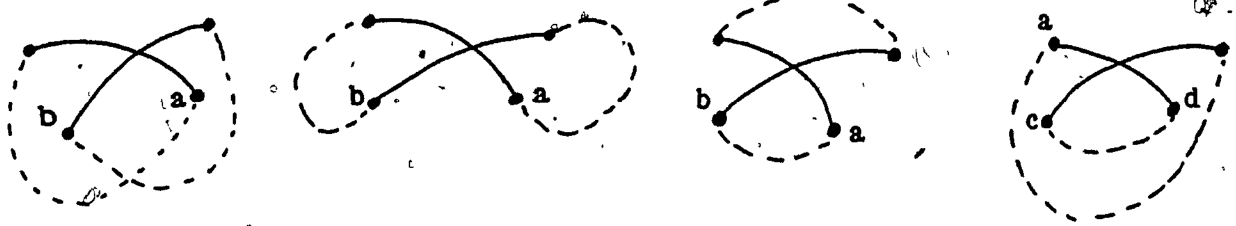
Lemma 5 : A pattern composed of only good arcs is *convex.*

Proof :

Suppose it is not . Then we can find two points p and q such that the line segment joining them is partly outside the region. See figure 3.9 . We know by the previous lemma that the region is simple. There must be a path going counterclockwise from a to b. Since we have a simple figure, it would imply that at one point in this path from a to b there is a change in the curvature direction which is impossible with arcs of the same sort. QED

Lemma 6 :

A pattern is non convex and has at least one bad arc *if and only if* there exists a circle c, corresponding to an arc in the pattern, such that c does not entirely contain the pattern,



1st situation *2nd situation* *3rd situation*

FIGURE 3.8

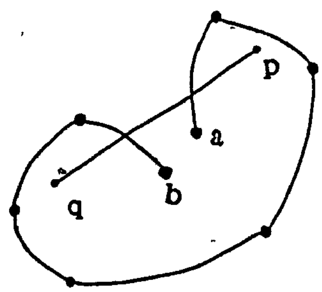


FIGURE 3.9

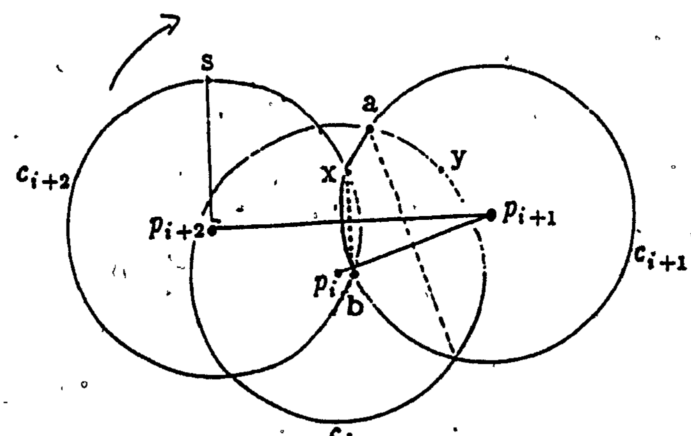


FIGURE 3.10

Proof :

only if : since the pattern is not convex it has either an open angle or a semi-open angle (or both). open angle : see figure 3.6-b, there exists circle c_i such that lune head lh_{i+1} is outside c_i. semi-open angle : see figure 3.6-c, there exists circle c_{i+1} such that lune head lh_{i+2} is outside c_{i+1}.

if : there exists a circle c corresponding to an arc *a* of the pattern such that part of the pattern is outside c. Take any point p on *a* (except its endpoints) and any point q on that region of the pattern which is outside c. The line segment (p, q) will be partly outside c, therefore "outside" the curvature of the arc *a*. Hence the pattern is not convex and contains at least one bad arc. QED

Corrollary 1 : A pattern is convex *if and only if* the circle corresponding to any arc of the pattern contains this pattern entirely.

Remark :

There is an obvious analogy between a convex polygon (intersection of half-planes) and a convex pattern (we prove in lemma 6 that the latter is the intersection of circles). Any edge e of a convex polygon P is such that P lies entirely in a half-plane defined by e. Any arc *a* of a convex pattern is such that the pattern lies entirely inside the circle corresponding to *a*. For a convex polygon, a straight half-line (circle of radius infinity) partitions the plane , for a convex pattern, a circle of "small" radius does.

Lemma 7 :

If there is a good arc from lh_i to lh_{i+1} on circle c_{i+1} such that lune(c_i, c_{i+1}) is either entirely contained in lune (c_{i+1}, c_{i+2}) or it entirely contains it, then in going from lh_{i+1} to lh_i (in counterclockwise order) the chain of arcs will have at least one bad arc. In other words a "lune *inside* lune" generates a *bad* arc.

Proof :

Consider figure 3.10 . Lune head a is outside c_{i+2}. By lemma 6 the pattern will have a bad arc. QED

Corrollary 2 : If a pattern has good arcs only then every pair of adjacent lunes forms a *fan* .

Lemma 8 :

There is a bad arc on c_i *if and only if* lune(c_{i-1},c_{i+1}) is entirely inside or entirely outside c_i .

Proof :

Consider figure 3.11 for an illustration. Let p_i , the center of c_i , be on the line bissecting the segment [p_{i-1}, p_{i+1}]. Just imagine moving c_i on this line from bottom up. As long as c_i entirely includes or excludes lune (c_{i-1}, c_{i+1}), the lune-heads lh_{i-1} and lh_i are on the boundary of c_i in clockwise order, which gives a bad arc. But when c_i intersects lune (c_{i-1}, c_{i+1}), then the two lune heads will be in counterclockwise order , which gives a good arc.

Corrollary 3 : There is a good arc on c_i *if and only* if c_i intersects lune (c_{i-1}, c_{i+1}).

Lemma 9 :

If a pattern has k arcs and k good arcs only, then it is the intersection of the k corresponding circles.

Proof:

Arc (i, i+1) is on circle c_{i+1}. See figure 3.12. Call R the region inside the pattern and I the intersection of the k circles. R is a convex region by lemma 5.

* $R \subseteq$ I : for every point p of R, p is inside all the k lunes, by the previous

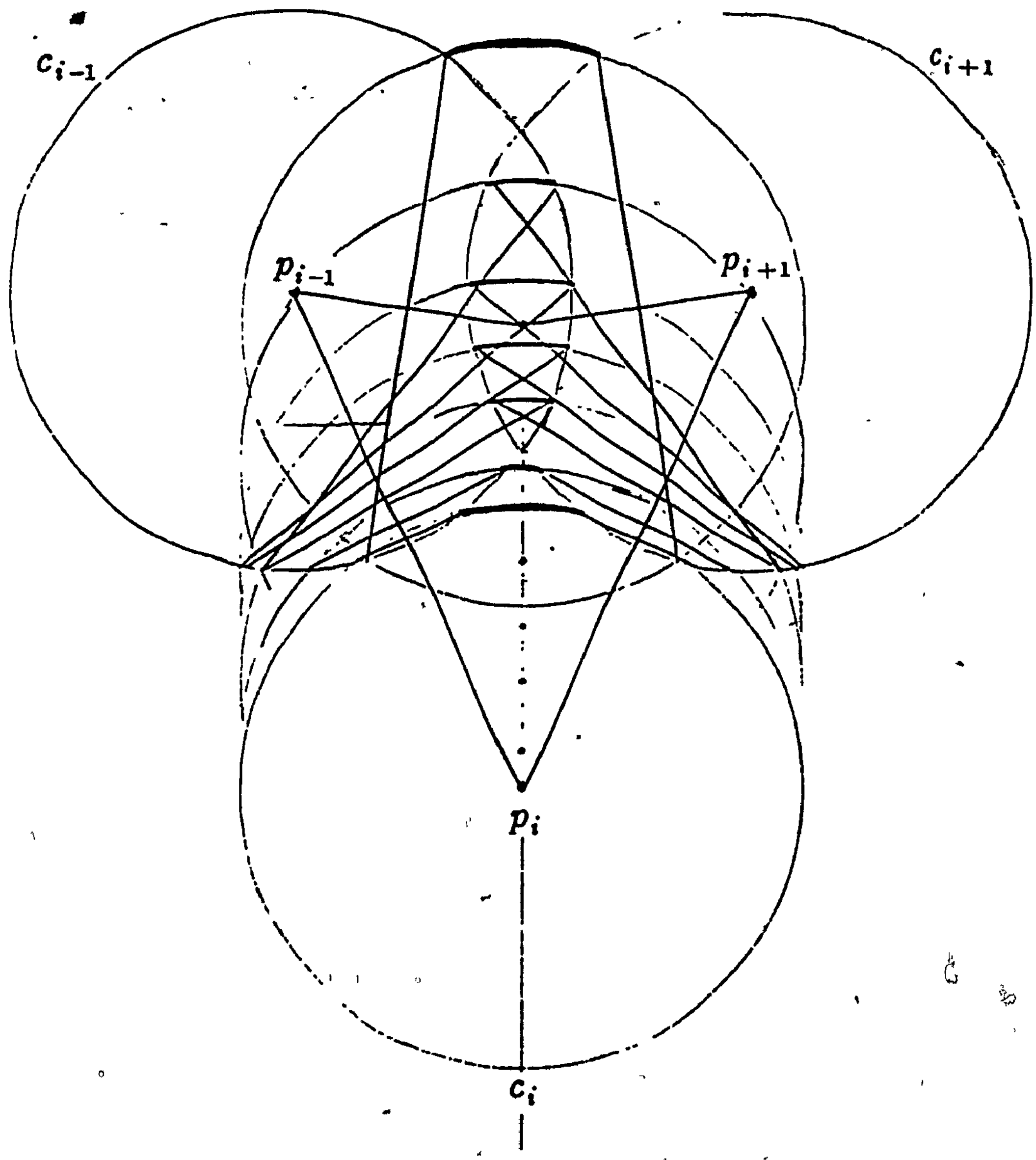


FIGURE 3.11

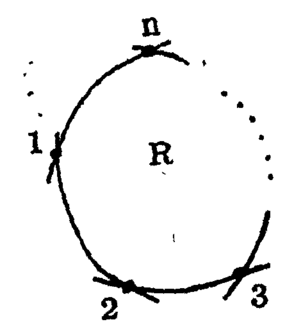


FIGURE 3.12

corrollary and because all lunes are fan types, so p is inside all the ci's. Therefore p is in I.

* $I \subseteq R$: for every point p of I, p is in all lunes,

--> p must be to the left of arc 12 and

--> p must be to the left of arc 23 and ....

--> p must be to the left of arc k1.

Therefore p is in R. QED

Lemma 10 :

Given a pattern, removing a bad arc and updating the pattern can be carried out in O(1) time.

Proof :

After deleting a bad arc on c_i, we update the pattern as follows:

1- compute lune head (c_{i-1}, c_{i+1}). Suppose it exists.

2- draw arc on c_{i-1} from lune head (c_{i-2}, c_{i-1}) to lune head (c_{i-1}, c_{i+1}).

3- draw arc on c_{i+1} from lune head (c_{i-1}, c_{i+1}) to lune head (c_{i+1}, c_{i+2}). QED

Lemma 11 :

Diameter (P) $<$ 2 * r *if and only if* every pair of circles intersect.

Proof :

Let p_k and p_l be such that $d(p_k, p_l) =$ diameter(P).

only if : diameter(P) $<$ 2*r, so $d(p_k, p_l) < 2*r$ and by transitivity $d(p_i, p_j) < 2*r$ for $(i, j) <> (k, l)$, therefore $c_i \cap c_j <> \emptyset$ for all $i, j = 1, 2, \ldots, n$.

if : $c_i \cap c_j <> \emptyset$, for all $i, j = 1,2,\ldots n$, therefore $d(p_i, p_j) < 2 * r$ for all i, j and in particular $d(p_k, p_l) =$ diameter(P) $< 2 * r$. QED

Definition 2 : Let $P = (p_1, p_2, \ldots, p_m)$ and $Q = (q_1, q_2, \ldots, q_n)$ be two convex polygons. Let c_i be the circle of radius r about vertex p_i and let s_j be a circle of radius r about vertex q_j. Assume that k circles , $1 < k <= m$, form the non empty intersection of all the c_i's. We say that $(\bigcap_{i=1\ m} c_i) \sim (\bigcap_{j=1\ n} s_j)$ if and only if $(\bigcap_{j=1.\ n} s_j)$ can be obtained from $(\bigcap_{i=1.\ m} c_i)$ after one translation and one rotation. The symbol $\sim$ is read "is congruent to".

Definition 3 : Two patterns on m and n circles respectively are said to be *equivalent* when one of the two following conditions holds :

- $(\bigcap_{i=1\ m} c_i) = (\bigcap_{j=1\ n} s_j) =$ empty set
- $(\bigcap c_i) \sim (\bigcap s_j)$

Lemma (Helly, 1923) :

If F1, F2, ... , Fn are convex subsets of the plane such that every three of them have a point in common, then they all have a point in common.

Observation :

If n circles have an empty intersection then there is at least one triplet with an empty intersection, that is there is at least one circle which is outside at least one lune (in the case that all pairs of circles intersect).

Remarks :

1- Suppose we have k circles, $c_1, c_2, ..., c_k$, having a non empty intersection I and such that each of these k circles contributes to this intersection. In other words the corresponding pattern has k good arcs. Adding a circle c_{k+1} such that I is completely contained inside c_{k+1} does not change the intersection, that is $c_1 \bigcap c_2 \bigcap \cdots \bigcap c_k = c_1 \bigcap c_2 \bigcap \cdots \bigcap c_k \bigcap c_{k+1}$. c_{k+1} is called a *redundant* circle.

2- Let c_1 and c_2 be two circles such that $c_1 \bigcap c_2 = \emptyset$. Adding any number of circles to c_1 and c_2 does not change anything and the intersection in every case remains empty.

3- Let there be k circles such that all pairs of circles intersect. Suppose there is a circle c_1 which is entirely outside the lune of two circles c_2 and c_3. Hence ($\bigcap_{i=1\ k}$ $c_i) = \emptyset$, and adding any number of circles to c_1, c_2 and c_3 does not change the non existence of the intersection which remains empty. Circles added to an initial set of circles, whose intersection is empty, are also called redundant circles. Therefore, we obtain *equivalent* patterns by adding or deleting ***redundant*** circles.

Lemma 12 :

Given a pattern with a bad arc on c_i such that lune (c_{i-1}, c_{i+1}) is entirely inside c_i, the pattern obtained after deleting c_i and updating remains *equivalent* to the preceeding one.

Proof :

1st case :

If the intersection of the n circles is not empty , then it must be entirely contained in every lune and in particular inside lune (c_{i-1}, c_{i+1}) hence inside c_i . So c_i is redundant and removing it does not change the intersection and its existence.

2nd case :

If the intersection of the n circles $c_1, c_2, \dots, c_n$ is empty then we must show that the c_i we delete is a redundant circle. We can test in linear time whether diameter(P) $> 2 * r$ in which case we stop. We thus consider the case diameter(P) $< 2 * r$. In this case no two circles are disjoint. Also for all c_i's, lune (c_{i-1}, c_{i+1}) exists otherwise c_{i-1} and c_{i+1} are disjoint which is contrary to our assumption. Suppose, by contradiction, that c_i is not a redundant circle. Then deleting it creates an intersection I. So I must be inside lune (c_{i-1}, c_{i+1}), therefore strictly inside c_i. Therefore a non empty intersection should have existed before deleting c_i, which contradicts the two assumptions that there was no intersection and that c_i was not redundant. (There is another intuitive proof with the three furthest circles). QED

Note :

This lemma also holds in the general case : If there is a circle c_i which contains entirely the lune of two other circles c_k and c_l, then c_i is redundant and can be deleted.

3.2.2 - The Algorithm

We now present a linear algorithm to compute the intersection of n circles c_i , for $i = 1, 2, \ldots, n$, of radius r whose centers are the vertices of a convex polygon $P = \{ p_1, p_2, \ldots, p_n \}$.

Notation : I is the intersection of the n circles.

diameter (P) is realized by (p_k, p_l)

ca is the total number of arcs in the pattern

cg is the number of good arcs in the pattern.

begin

Step 1 : If diameter(P) is

$> 2 * r$: stop, $I = \emptyset$.

$<= 2 * r$: - for all circles c_i do :

If c_i contains entirely lune(c_k, c_l), delete c_i.

If c_i is outside lune(c_k, c_l) : stop, I is empty.

- Draw pattern and in the process :

* update *cg* and *ca*.

* for each bad arc on c_i, if lune(c_{i-1}, c_{i+1}) is outside c_i : stop, I = $\emptyset$.

Step 2 : If $ca <> cg$ then for every bad arc on c_i do :

1- If lune(c_{i-1}, c_{i+1}) is inside c_i then

* delete c_i

* update pattern and counters

else stop, I = $\emptyset$.

2- If $cg < 2$, stop , I = $\emptyset$.

Step 3 : If $ca = cg$: the pattern is the intersection I.

end.

3.2.3 - The Analysis

Correctness : In step 1 , if diameter(P) is $> 2 * r$ then c_k and c_l are disjoint and I is empty. If it is $<= 2 * r$ then we draw the pattern and deleting redundant circles does not change the intersection by lemma 12. In step 2, we delete safely, by lemma 12, redundant circles. In step 3, the pattern composed of only good arcs *is* the intersection, by lemma 9. The algorithm correctly computes the intersection.

Complexity : In step 1 , the diameter of a convex polygon can be computed in linear time. The complexity of drawing the pattern is also linear in the number of vertices. For each bad arc, testing the position of the lune of its two neighbours takes constant time. In step 2 there are at most n bad arcs, and for each of them , deleting a c_i and updating takes constant time by lemma 10. In step 3 , if there are only good arcs then we simply enumerate in O(n) time the already ordered list of lune heads and arcs of the pattern and obtain the intersection of the n circles. The algorithm correctly computes the intersection of n circles of equal radius whose centers are the vertices of a convex polygon in O(n) time.

It is easily seen that Ω (n) is a lower bound since each of the n circles may contribute to the intersection.

3.3 - Intersection of circles of different radii

In this section, we analyse the general case of computing the intersection region of n circles of arbitrary radius , about vertices of a convex polygon.

Problem statement : Given n circles of arbitrary radius, whose centers are the vertices of a convex polyon P, compute their intersection.

3.3.1 - Preliminary results

We will first give a few lemmas before exhibiting the algorithm and its analysis.

Lemma 13 :

Given two consecutive circles on a convex polygon, c_i of radius r_i and c_{i+1} of radius r_{i+1}, about vertices p_i and p_{i+1} respectively, such that $c_i \bigcap c_{i+1} = c_i$, deleting c_{i+1} does not change the intersection of the set of circles.

Proof :

case 1 : the intersection of the n circles is not empty, then it must be inside c_i, therefore strictly inside c_{i+1} which is then a redundant circle.

case 2 : the intersection of the n circles is empty, we show that deleting c_{i+1} does not create an intersection. If there is no intersection then there is at least a triplet with empty intersection. Now we delete c_{i+1} such that c_i is completely inside c_{i+1}. If we get an intersection then all triplets have non empty intersection , then all n-1 circles have non empty intersection, therefore this intersection must be completely inside c_i. And adding c_{i+1} shouldn't change it because c_{i+1} contains c_i therefore it contains the intersection , this means that the n circles have a non empty intersection which is contrary to our assumption. QED

Lemma 14 : let c_k and c_l be the circles with the two largest radius r_k and r_l. If the diameter of P is strictly greater than $(r_k + r_l)$ then I is empty.

Proof :

let p_d and p_e realize the diameter. If $\{ c_d , c_e \} = \{ c_k , c_l \}$ then the claim is evident otherwise $\text{diam}(P) = d(p_d , p_e) > r_k + r_l$, since $r_d + r_e < r_k + r_l$ then $d(p_d , p_e) = r_d + r_e + L$, for some $L > 0$, therefore c_d and c_e do not intersect and I is empty. QED

3.3.2 - The Algorithm

begin

step 1 : take the two circles with the two largest radius, r_k and r_l.

If diameter (P) $> r_k + r_l$ then stop, I = empty set.

otherwise

begin

* draw the pattern and in the process :

test1 : if two adjacent circles don't intersect stop, I = empty set

test2 : for each bad arc on c_i , if lune (c_{i-1}, c_{i+1}) is out of c_i

then stop , I = empty set ,

test3 : if two adjacent circles have an inclusion : delete the including circle and update the pattern.

* set *ca* to n.

end

step 2 : if $ca <> cg$ then

for every bad arc on c_i do :

1- if lune (c_{i-1}, c_{i+1}) is inside c_i then

* delete c_i

* update pattern and counters

else stop, I = empty set.

2- if $cg < 2$ then stop, I is empty.

step 3 : if $ca = cg$ then the obtained pattern *is* the intersection.

end.

3.3.3 - The Analysis

Correctness : In step 1 , if the diameter test is true , then the overall intersection is empty by lemma 14. Otherwise we draw the pattern and deleting redundant circles does not change the intersection by lemma 12 and 13. In step 2 , by the same lemmae we safely delete the redundant circles. In step 3 , if the pattern consists of good arcs only, by lemma 9, it is the intersection of the n circles. Therefore the algorithm correctly computes the intersection of the n circles.

Complexity : Step 1 , the diameter test can be performed in linear time. While drawing the pattern we advance counterclockise and every time we add an arc to the pattern it corresponds to a circle that hasn't been visited yet. The pattern will contain n arcs at most. Making tests 1 , 2 and 3 is each time O(1). If test 3 is true though, we may have to update the pattern (delete including circles, update arcs) which is O(1), and if a circle is to be deleted there is only one test done. We delete at most the number of arcs the pattern has so far. So the total cost of this step is linear. In step 2 , there are at most n bad arcs, we have no inclusion of adjacent circles. Deleting and updating one arc is O(1) by lemma 10. For step 3 , if only good arcs remain, we enumerate, in order, the arcs and the vertices of the pattern, which takes linear time. Therefore the algorithm correctly computes in linear time the intersection of n circles of different radii whose centers are the vertices of a convex polygon.

CHAPTER 4

4.1- Introduction

Given two disjoint convex polygons P=$\{p_1, p_2, \ldots, p_n\}$ and Q=$\{q_1, q_2, \ldots, q_m\}$ with n and m vertices respectively and given a line segment S = $[a, b]$ of length r, also called a *ladder*, a *needle* or a *rod*, we want to compute the regions in Q that b can reach with the constraint that a lies within the boundaries of P, see figure 4.1. Testing whether a reachability region *exists* for point b in polygon Q, will be a step of the final algorithm, described in the next chapter. In this chapter we will therefore assume that such a region exists in Q and concentrate on the algorithm to compute it. This is done in two main steps. The two problems we will solve are best visualized in figures 4.2-a and 4.2-b. Let CP denote the *convex hull* of n circles of radius r, with centers the n vertices of P respectively, and let UP denote their *intersection* . We will assume UP is not empty. Recall from the previous chapter that the reachable region for point b when a remains inside P is CP - UP, where the symbol - denotes the set difference. In other words the unbounded region of the plane outside CP is unreachable as , well as UP which is inside CP. Since we assumed that a reachability region exists then CP and Q must have a non-empty intersection Q$'$. Finding Q$'$ will be the first part of this chapter. Also since UP exists, intersecting UP with Q$'$ and obtaining Q$''$ will be the second part of this chapter. The reachable regions for point b will be RQ = Q$'$ - Q$''$.

4.2 - The first problem

4.2.1 - Problem statement

We want to compute the intersection region between CP and Q, denoted by Q$'$. Let us first expose the possible situations according to the assumptions made

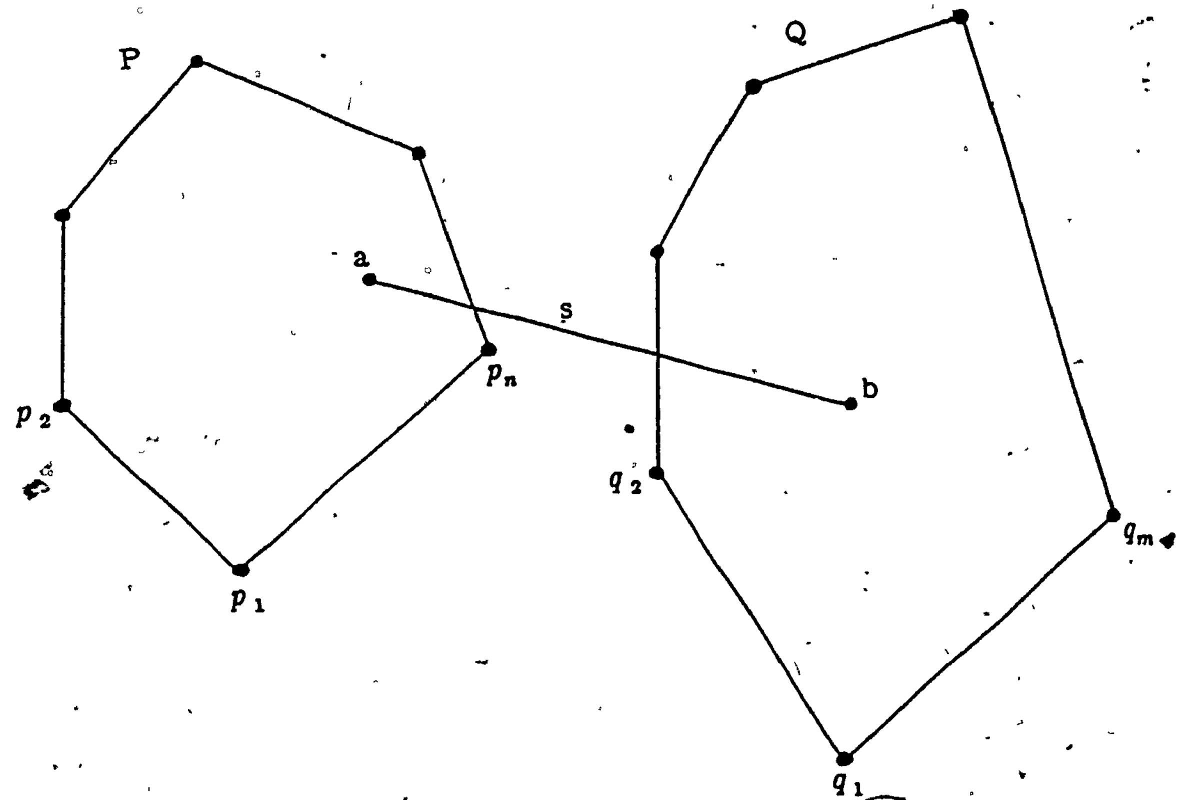


FIGURE 4.1

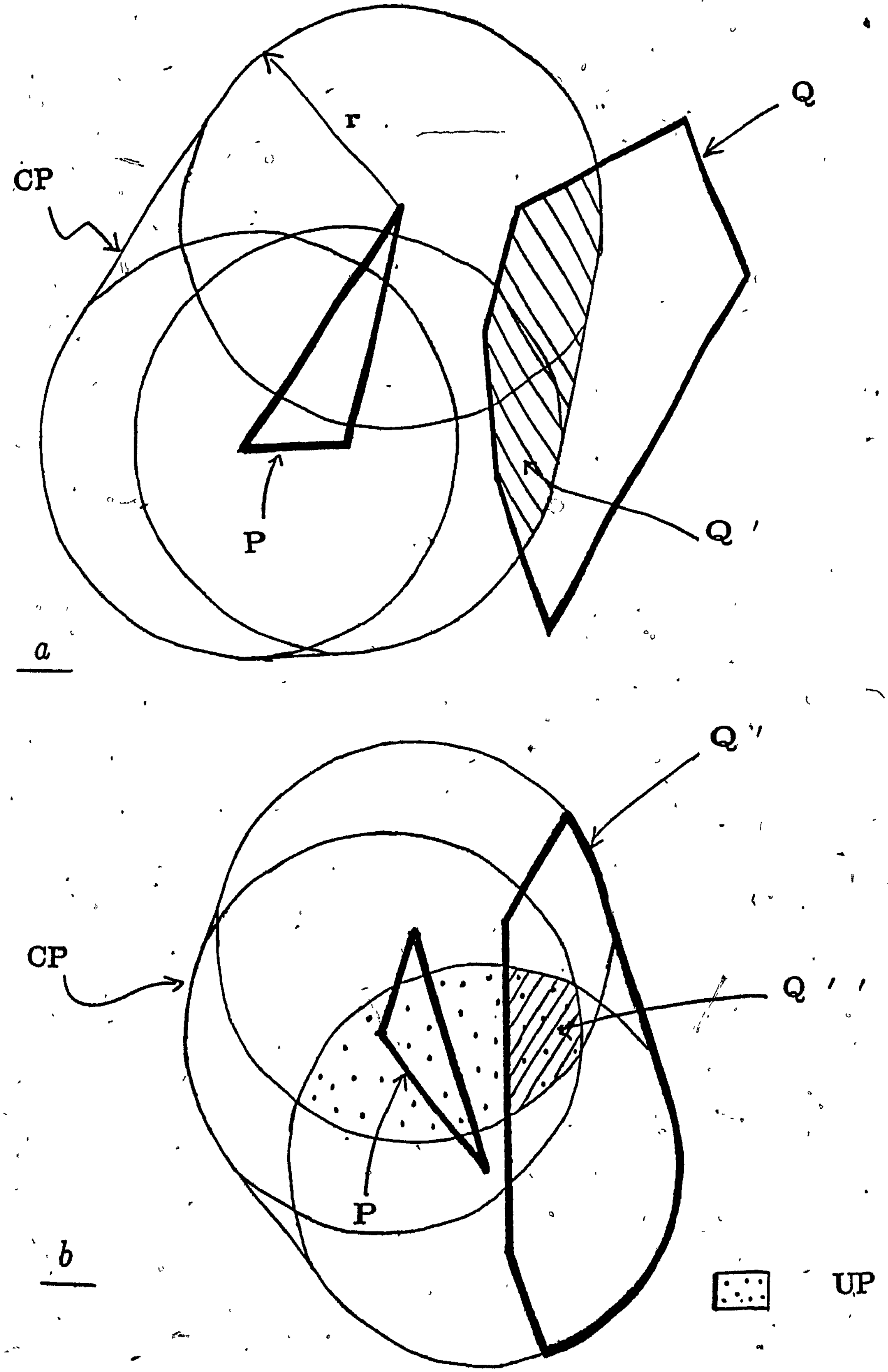


FIGURE 4.2

previously. Consider figure 4.3 . Figure 4.3-c cannot occur since we assumed that a reachability region exists in Q. Also figure 4.3-d is impossible because it implies that Q contains P , and we have assumed P and Q to be disjoint , Therefore only the first two cases are valid under the above assumption and notice that in the first situation Q is contained in CP - P.

4.2.2 - General description of the algorithm

Refer to figure 4.4 for the following definitions.

Definition 1 : *old and new edges*

An old edge is an edge of CP. A new edge is an edge in CP′ replacing an arc of CP, by a segment joining the two arc endpoints. If P has n vertices, then CP′ has n old and n new edges.

Definition 2 : *a dome*

The region under an arc of CP but outside the corresponding edge in CP′ is called a dome . CP - CP′ thus gives n domes , each corresponding to an arc.

Definition 3 : *an attic*

Every arc of a new edge is contained in the triangular region which is delimited by:

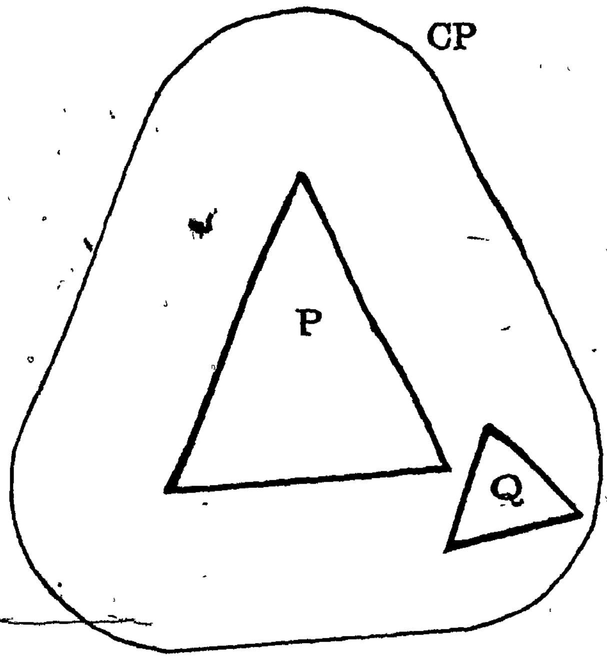
- the new edge.
- and the two lines supporting the two neighbouring old edges.

This region is always bounded and called the pediment, or the attic .

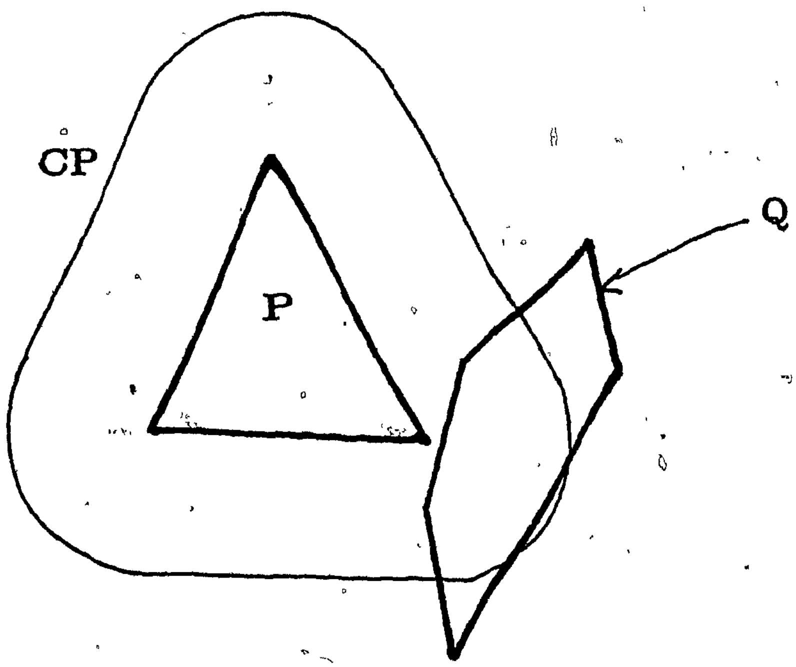
Definition 4 : *an arc region*

It is an unbounded rectangular region delimited by

- a new edge
- and the two half lines perpendicular to this new edge at its endpoints and going

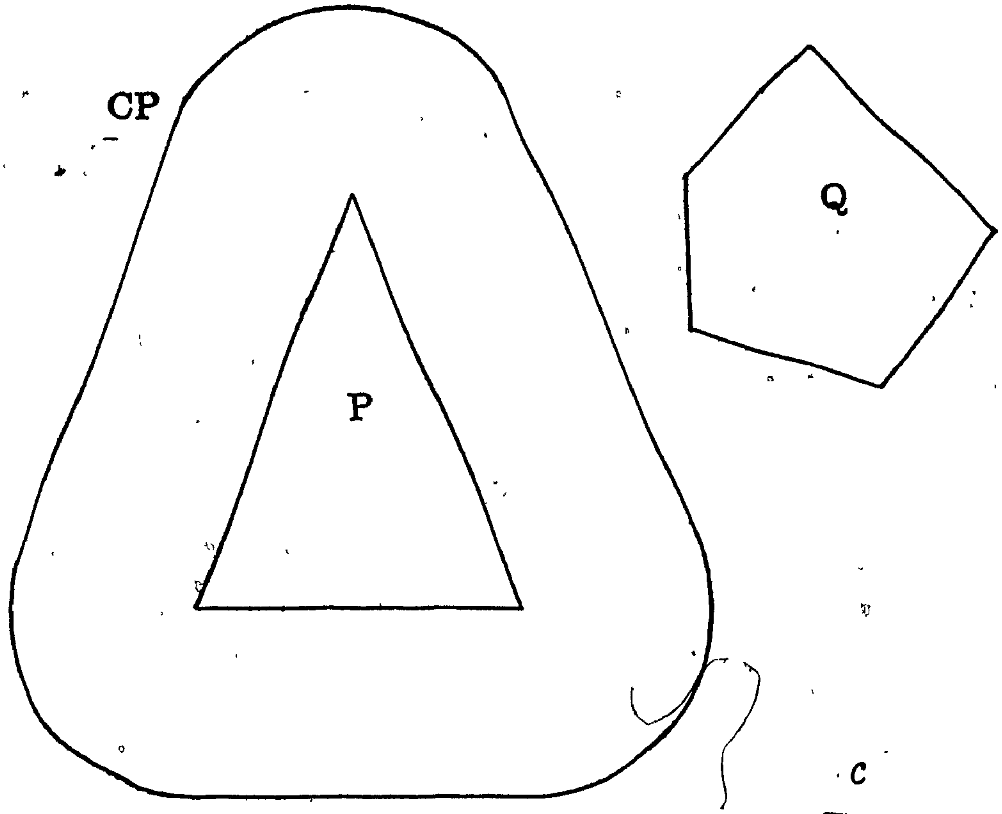


a

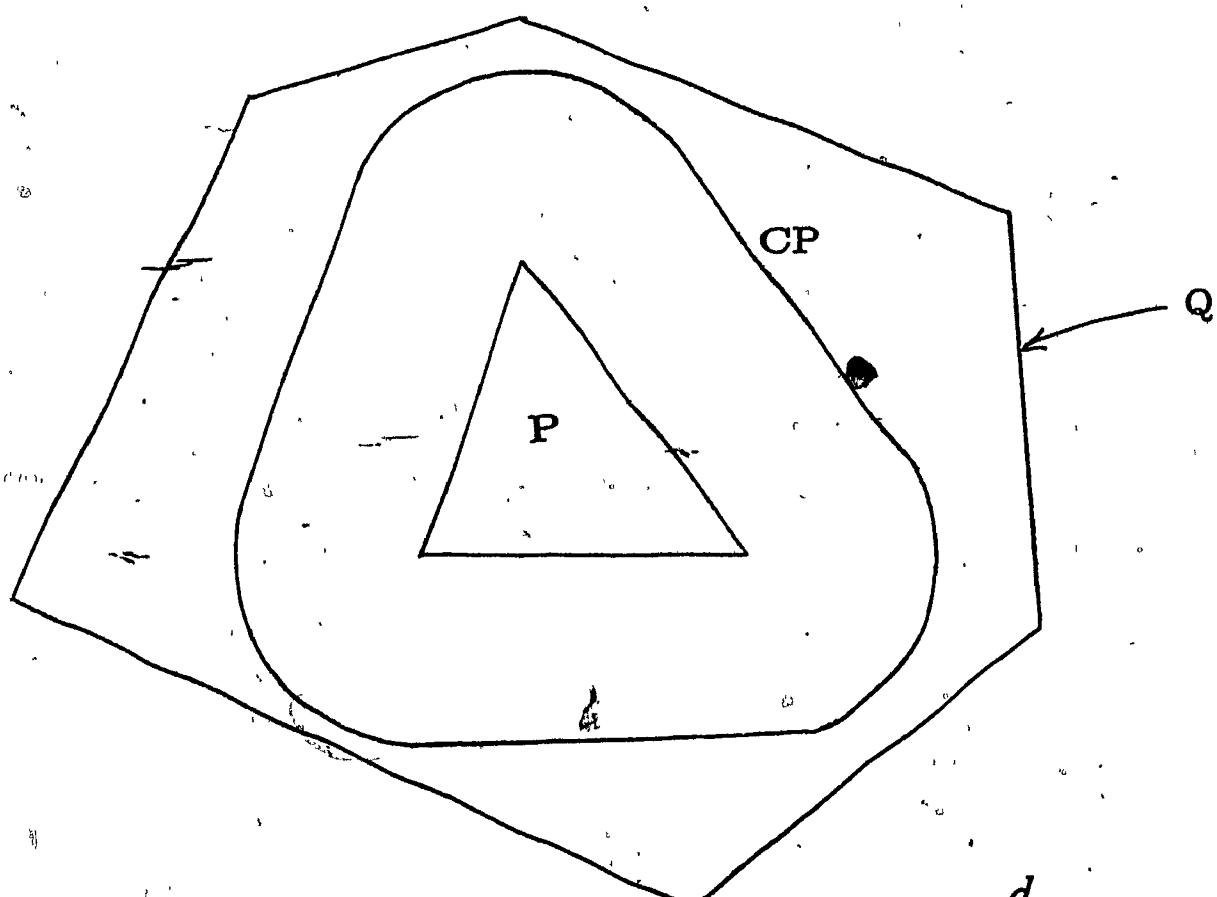


b

FIGURE 4.3



c



d

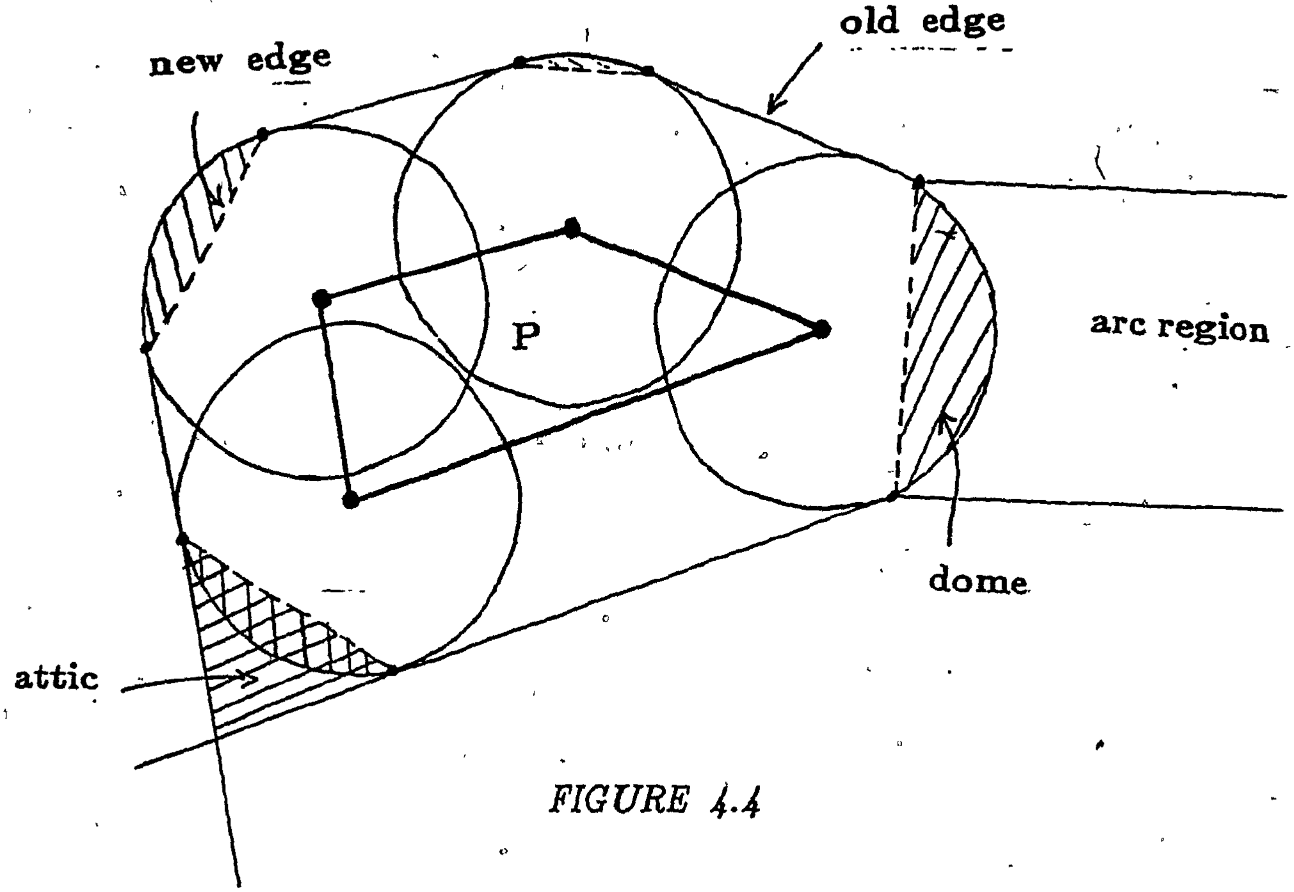


FIGURE 4.4

outwards that is lying in the half plane delimited by the new edge that does not contain the polygon.

The boundary of CP is an alternating sequence of arcs and edges. Therefore a straightforward application of an existing algorithm for intersecting two convex polygons is not sufficient. We could solve the problem of computing the intersection between CP and Q using existing tools. In O(nlog n + klogn) time, where k is the number of intersection points, the algorithm of Bentley and Ottmann, [BO], based on that of Shamos and Hoey, [SH], reports all the intersection points between the set of line segments and arcs. To have a linear running time algorithm though, we first work with two convex polygons, then we reinsert the arcs to update the intersection region.

The two possible situations are then the following :

- Q is entirely inside CP.
- Q intersects CP and some parts of Q are unreachable regions i.e. outside of CP

We now give a general description of the algorithm that computes CP $\cap$ Q before decribing it in full detail in the next sections.

In a first step we replace all the arcs of CP with straight edges and obtain CP'. We determine whether CP' and Q intersect in logarithmic time using Chazelle and Dobkin's algorithm [CD], [Cha1]. We assume that CP' and Q are in a general position, and that the vertices of CP' U Q are distinct. Let I be CP' $\cap$ Q.

We distinguish two cases :

* *If I is the empty set*: then Q is intersecting one dome and only one, by either being strictly inside it or by intersecting its arc. In the latter case the minimum distance between the two polygons determines the arc that Q intersects. This can be computed in sublinear time using Edelsbruner's binary elimination technique.

His algorithm will be briefly described in the next chapter.

* *If I is not the empty set* : then we compute the convex hull of the union of CP' and Q using any of the algorithms [Tou4], [Sha2], [Tou2]. If the convex hull is CP' then Q is included entirely in CP' - P. Otherwise we calculate the intersection region between CP' and Q. For this , several algorithms are available [O'R2], [Sha2], [Tou3], we will use O'Rourke's algorithm for its simplicity , [O'R2], which we also describe in the next section.

We then identify the subchains of Q that are outside CP' , by finding the intersection points lying on the boundary of the polygons. Those subchains possibly intersect arcs of CP. To find those arcs we partition the plane around CP into regions containing arcs (*arc regions*), and regions not containing any. Then, for every arc region we test for intersection between the outer subchain of Q and the arc of the region we are in.

4.2.3 - Preliminary algorithms

4.2.3.1- O'Rourke's polygon intersection algorithm

In this section we present a short description of the algorithm to compute the intersection of two convex polygons P and Q, with m and n vertices respectively. The general idea is the following : two "bugs", one on each polygon boundary, go around P and Q, and an intersection point is detected and calculated every time that they cross each other. These two bugs advance according to some rules, and the key idea is *not to advance on the boundary (either of P or of Q) whose current edge may contain a yet to be found intersection.* We give here an outline of the algorithm and for a more detailed description , refer to the original paper [O'R2].

```
Procedure CONVEX POLYGON INTERSECTION
begin i := j := k := 1 ;   { i and j are the two "bugs" }
repeat
  begin
    if ( edges p_{i-1} p_i and q_{j-1} q_j intersect ) then print the intersection ;
    ADVANCE { either i or j is incremented } ;
    k := k + 1;
  end
until k = 2 * ( m + n );

if ( no intersection has been found ) then
  begin if p_i ∈ Q then P ⊆ Q
     else if q_j ∈ P then Q ⊆ P
         else P ∩ Q = ∅
  end
end.
```

4.2.3.2- Edelsbrunner's minimum distance algorithm

Let P and Q be two convex disjoint polygons with m and n vertices respectively. Let d(P, Q) denote the minimum distance between P and Q.

Edelsbrunner has shown that :

Lemma : If $d(P, Q) > 0$ then there exists $p \in P$ and $q \in Q$ that realize d(P,Q) and such that at least one of them is a polygon vertex.

His algorithm consists in performing a binary search in the list of vertices of P and Q and at every step, eliminate half of the candidates to be considered for the minimum distance. This binary elimination technique yields a sublinear algorithm, as it is proved by the following theorem :

Theorem :

The minimum distance between P and Q , two convex polygons with m and n vertices respectively, along with points p in P and q in Q that realise it , can be computed in O $(\log m + \log n)$ time.

For a detailed desciption of the technique, refer to [Ede].

4.2.4 - Preliminary results

Assumptions : P and Q are disjoint.

CP and Q have a non empty intersection.

We will prove some lemmae and theorems as we give a more detailed description of the algorithm. The context will thus help a better understanding of the results. We now start explaining the algorithm to compute $CP \cap Q$.

We first replace each arc of CP by a straight edge joining the two endpoints of the arc, and obtain CP$'$. Then we detect whether CP$'$ and Q intersect and consider two cases. This intersection I is empty or not.

CASE 1 : $I = Q \cap CP' = \emptyset$:

Then since we have assumed that Q and CP are not disjoint it must be that Q intersects a dome . It may or may not intersect the corresponding arc. The problem now is to find which dome . But before let us prove that Q cannot intersect more than one dome. In fact it intersects *exactly* one dome.

Lemma 1 :

If $Q \cap CP' = \emptyset$ then Q intersects exactly one dome of CP.

Proof :

Since CP and Q are not disjoint then Q must intersect CP. Since CP$'$ and Q are disjoint then Q must intersect at least one dome and no old edges. We must now show that Q intersects at most one dome : Suppose it intersects more than one dome. Since $Q \cap CP' = \emptyset$ then this implies that Q is not convex, which contradicts our initial assumption . QED

We now want to find which dome of CP, Q intersects. We calculate the convex hull of CP' and Q and get two bridges. It has been proved in [Tou3] that in the case that the intersection is not an inclusion, there are exactly two bridges. Several cases arise.

Let us first point out that we cannot have that the inner chain of CP' is one old edge only, in other words the two bridges of CH ($CP' \cup Q$) be connected to the two endpoints of the same old edge. This would simply mean that Q and CP are disjoint which contradicts our initial assumption . See Fig. 4.5-a. As well as we cannot have figure 4.5-b which is the more general case of Q and CP' being disjoint . We assume that none of these two cases can happen.

Now in some situations we can identify quickly which dome Q intersects and then take the corresponding arc to see if it intersects Q . These cases are depicted in figure 4.6.

In case 1 , the inner part of CP' is one new edge only, therefore the dome of that new edge is the seeked dome. We test for intersection points between Q and the arc. If there are none then Q is completely " under " the dome. If there are any , we identify them and construct the region.

In case 2 , the inner chain of CP' is composed of one new and one old edge (that are consecutive). The seeked dome is then the only one i.e. the one corresponding to the new edge.

In case 3 , the inner chain is composed of an alternating sequence of two old and one new edges and again we identify quickly the intersecting dome.

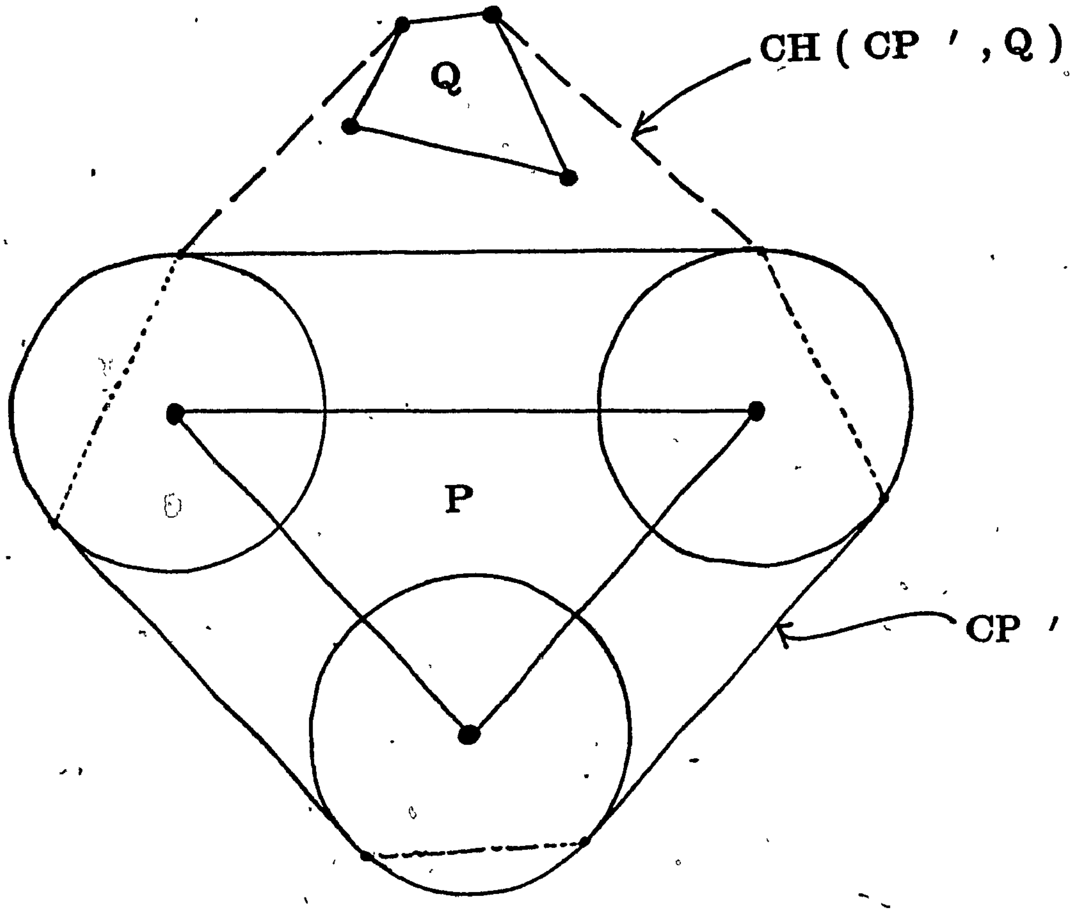
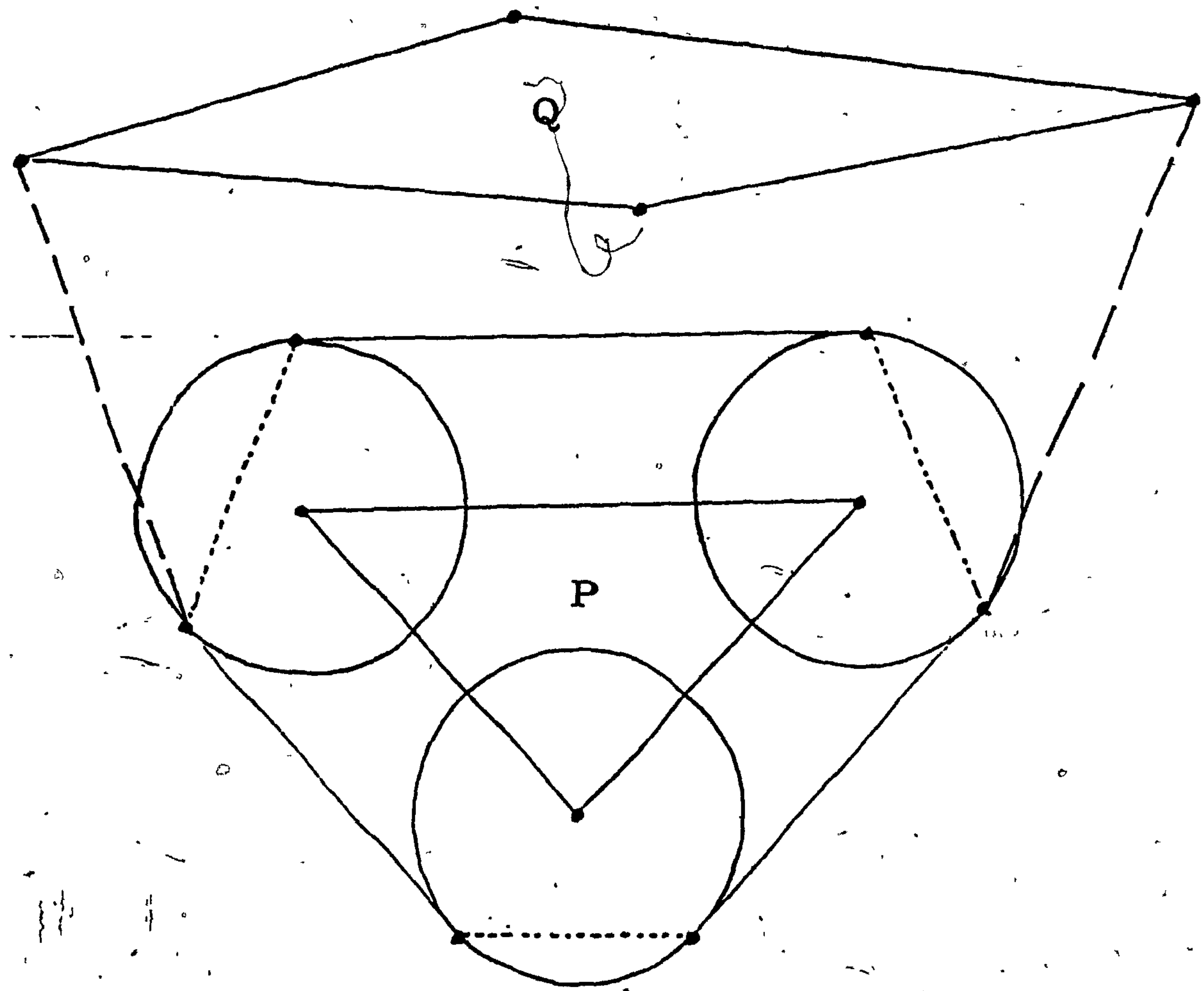
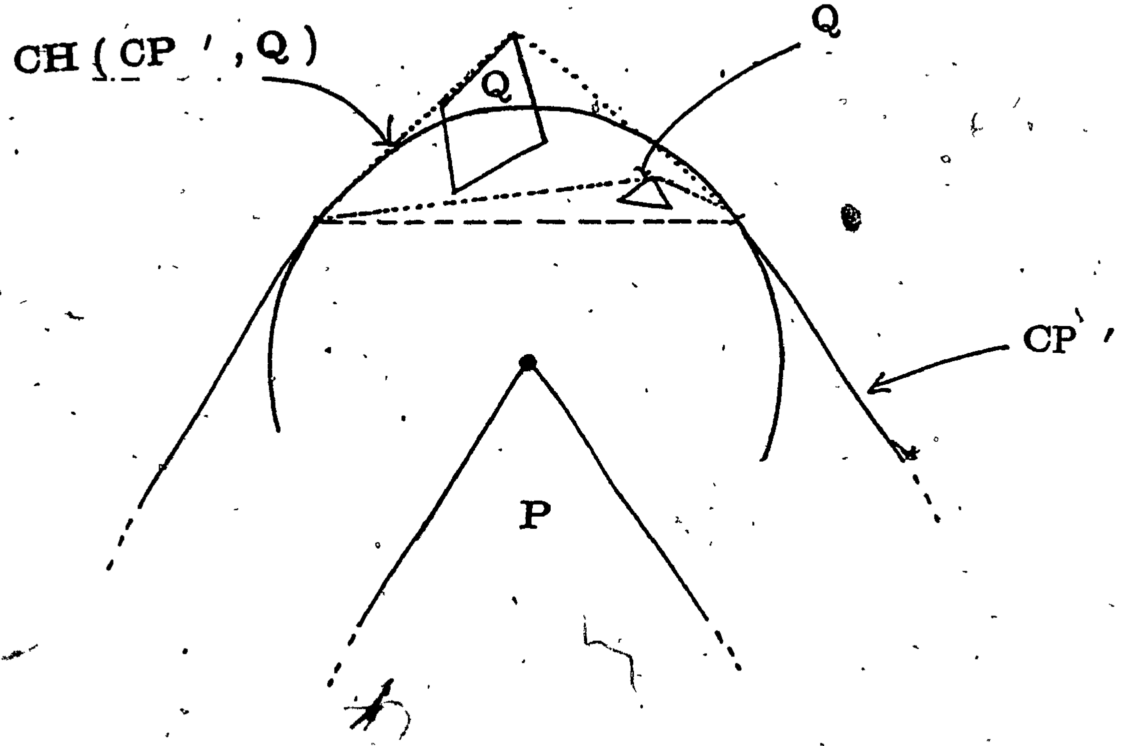


FIGURE 4.5 a

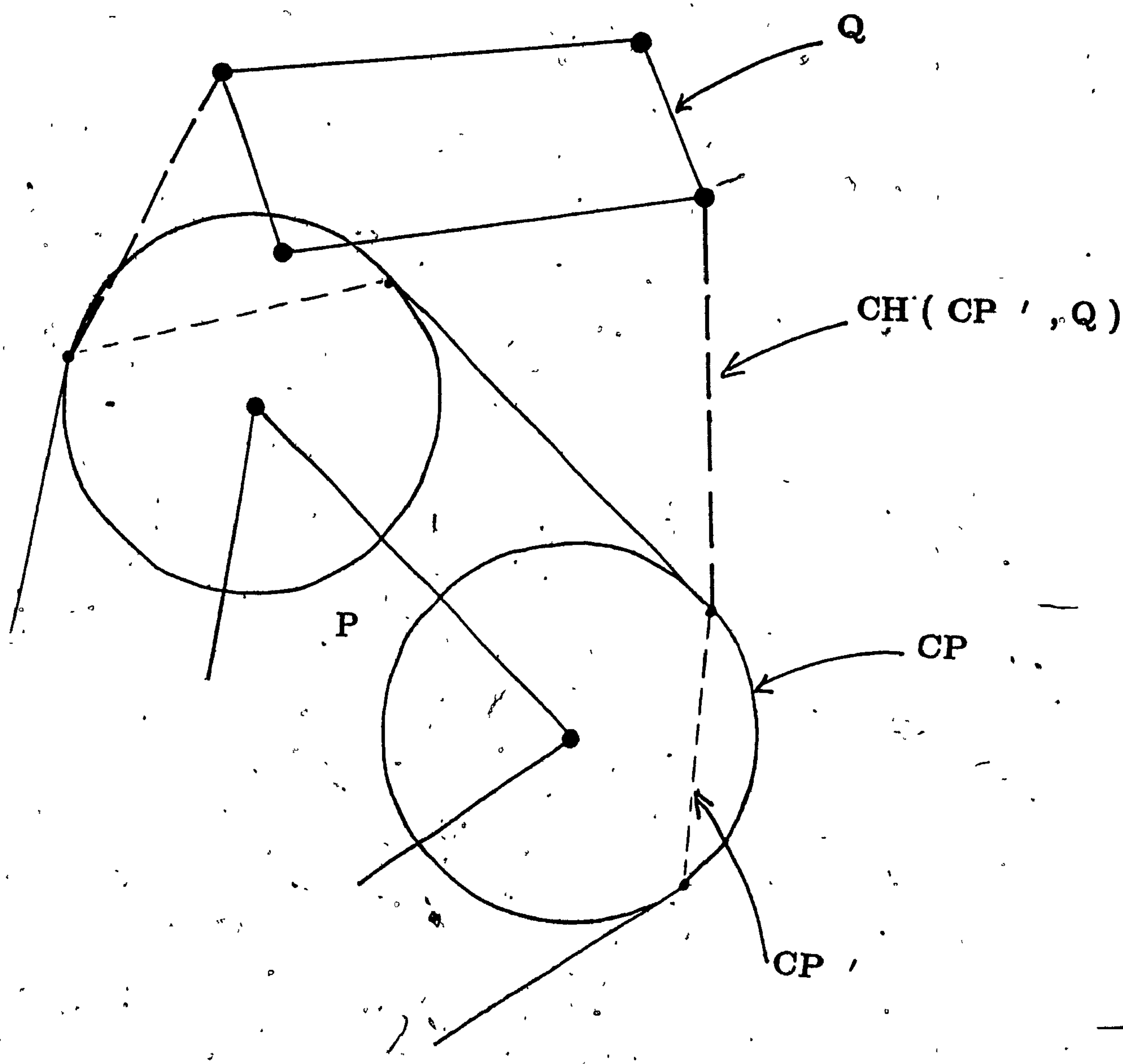


b

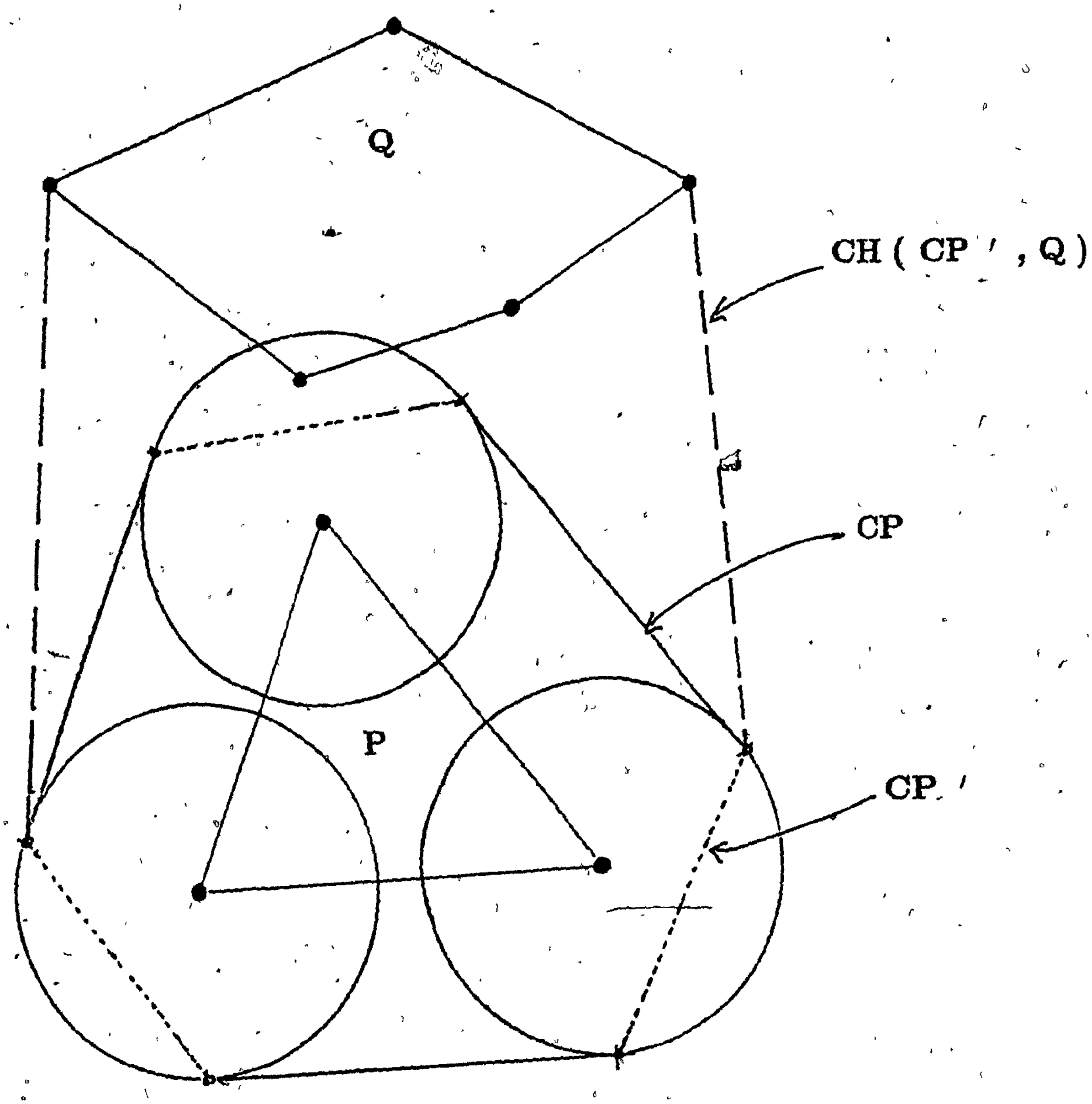


case 1

FIGURE 4.6



case 2



case 3

In other than these cases , the inner chain of CP' has more than one new edge and hence more than one dome are candidate for intersecting Q. And we must determine which dome intersects Q. In this general situation we compute the minimum distance between Q and CP' and this tells us which dome Q intersects. If Q has n vertices and CP' has m vertices then using Edelsbrunner's, [Ede], algorithm, described earlier, we can compute in $O(\log m + \log n)$ time the minimum distance between these two polygons .

We must first say that the minimum distance is realised by either a pair of vertices or a vertex and a point on an edge. We want now to prove that the point on CP' that realizes the minimum distance *determines* the dome and therefore the arc that intersects Q. That point p is either a vertex or a point lying on an edge.

case 1.a : p is a *point* on an edge of CP'

Lemma 2 :

Let A and B be two convex polygons. Let (s, t) of A and B repectively be the two points realizing the minimum distance between A and B. Let L_A and L_B be the two parallel lines tangent to lune (s, t) at points s and t respectively. A and B are on different sides of L_A and of L_B such that the region sandwiched between L_A and L_B is free of any points.

Proof :

This result is also proved in [McT]. Refer to figure 4.7. Suppose , by contradiction, that a point x, say $x \in A$, is between L_A and L_B. Since A is convex the segment [x, s] is inside A. But since [x, s] is not tangent to lune(s, t) it intersects it , which implies that lune(s, t) is not empty. This contradicts the fact that s and t realize the minimum distance between A and B. QED

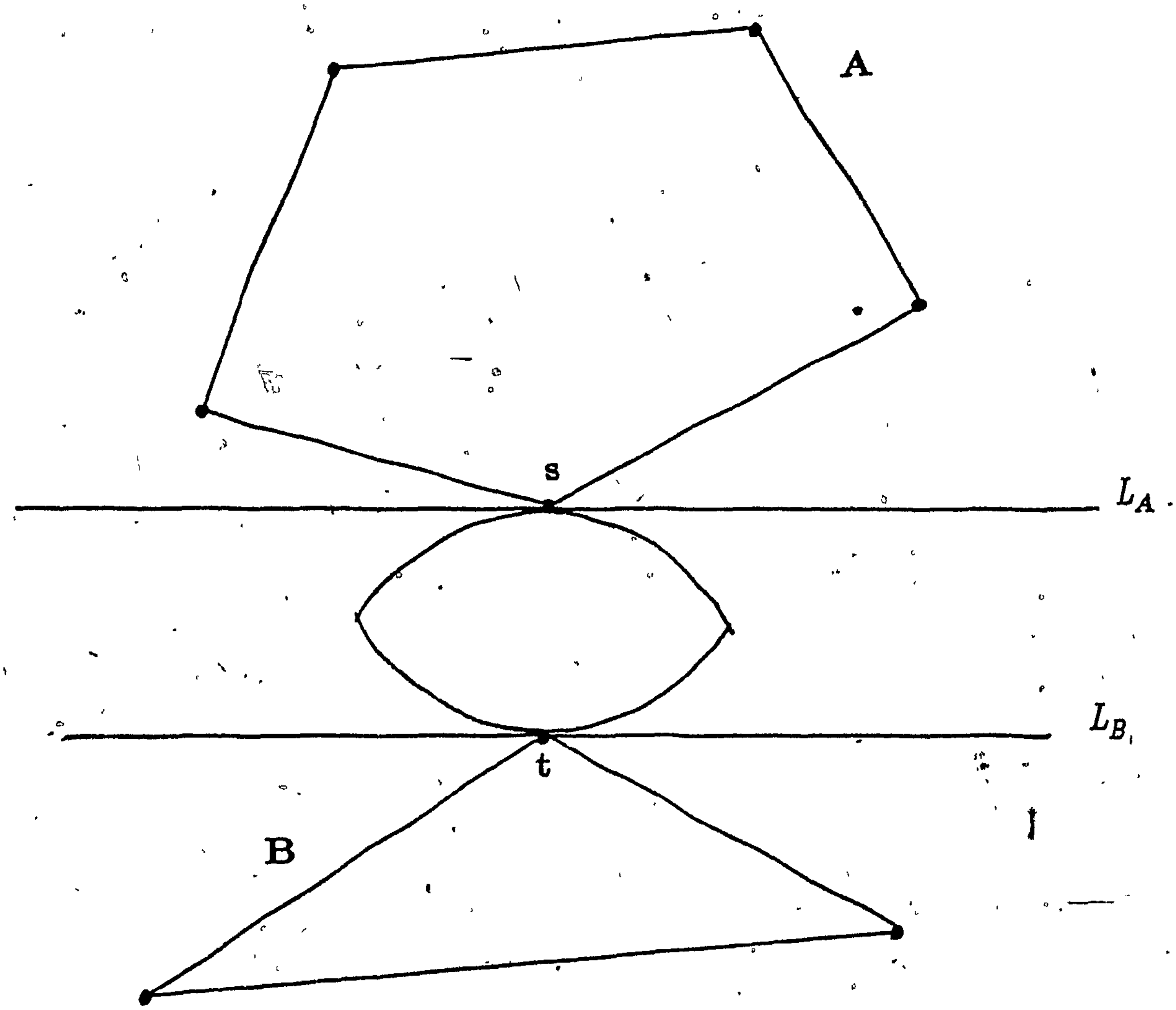


FIGURE 4.7

Lemma 3 :

If the point p of CP′ ,realizing the minimum distance between CP′ and Q, is a point on an edge of CP′ then it must be on a *new* edge.

Proof :

Suppose by contradiction that the point realizing the minimum distance lies on an old edge of CP′. The point in Q realizing the minimum distance must then be a vertex q. Let d = d (p, q) be the minimum distance between Q and CP′. See the constructions in figure 4.8. The lune of radius d and about p and q must be empty because p and q realize the minimum distance. Let L be the line supporting the old edge on which p lies . Since CP is convex it must lie entirely below L. Let L′ be the line parallel to L passing through q . L′ is tangent to lune(p, q) at point q.

Now Q cannot intersect any arc of CP and CP in general, because CP is below L and Q above L' (and if it did it would mean that Q is not convex, a contradiction). This means simply that Q and CP are disjoint which is a contradiction to our initial assumption, QED.

Therefore if the minimum distance is realized by a point on an edge in CP′ this point must lie on a new edge.

We now want to show that the arc corresponding to this new edge is the one intersecting Q.

Lemma 4 :

If the minimum distance between CP′ and Q is realised by a point p on a new edge of CP′, then the dome adjacent to p intersects Q.

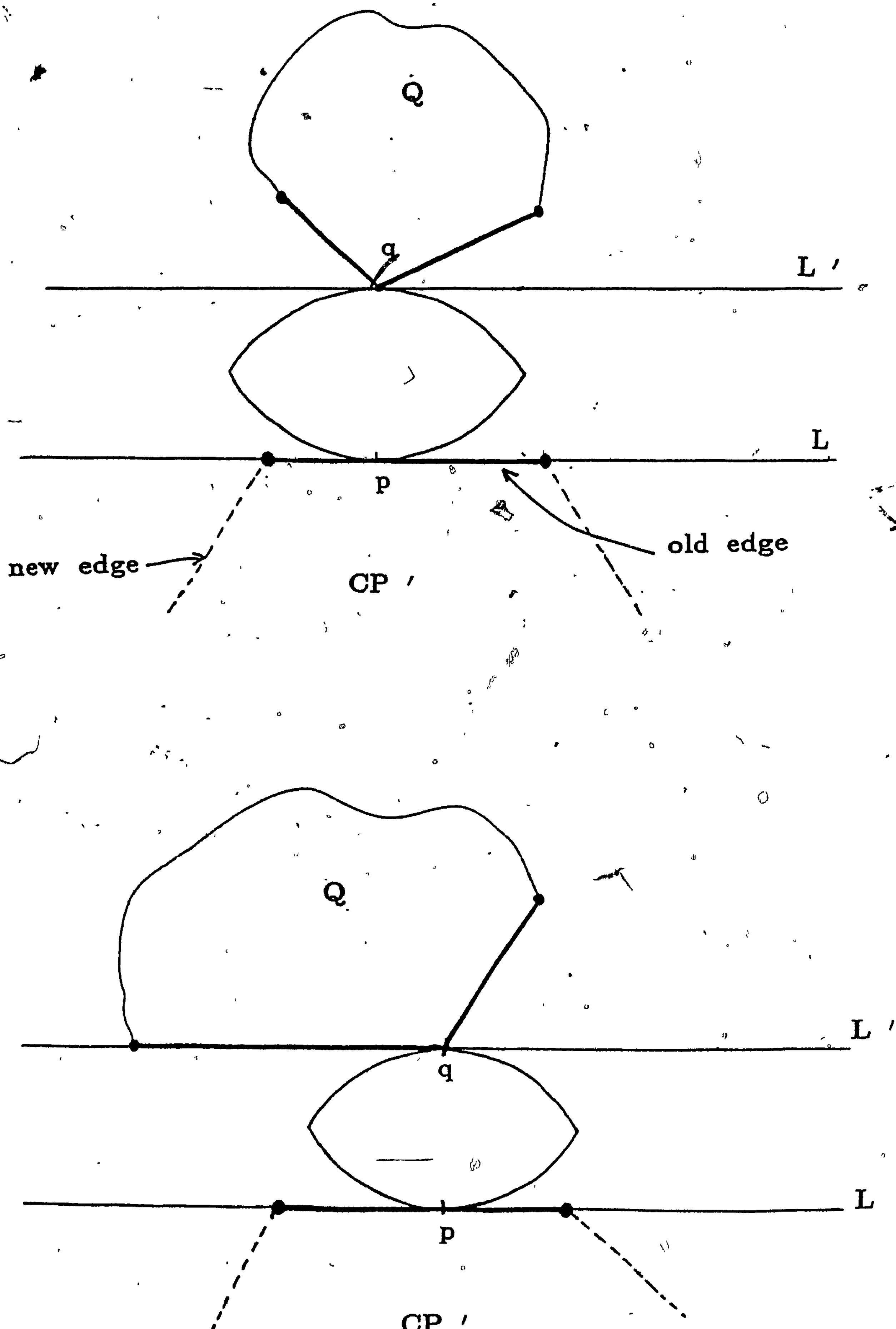


FIGURE 4.8

Proof :

We know that Q and CP′ are disjoint but that Q and CP intersect and that Q intersects exactly one arc. For Q to intersect another arc of CP, Q must come below line L which is impossible due to Q′s convexity. Therefore Q intersects the dome (and the arc) corresponding to the new edge on which p lies and q is under the dome, QED.

case 1.b : p is a *vertex* of CP′

Now we want to examine the case where p is a vertex of CP′. This vertex will have on one side of it a new edge and on the other an old edge. We will show that it is the arc corresponding to the new edge that intersects Q.

The point realizing the minimum distance in Q is either a vertex or a point on an edge and consider figure 4.9.

Now, p is between a new edge e_n and an old edge e_o. Let O be the line supporting e_o. By convexity CP lies at one side of O. So, for Q to intersect an arc of CP, the intersecting part of Q must be :

- at one side of O (the same side CP lies)
- and at one side of L′ (the same side Q lies)

This defines a region in the plane (the intersection of two half planes) where the intersection of Q and CP occurs (we assume O and L′ are not parallel for if they were then Q and CP would be disjoint).

We must now show that Q must intersect the arc corresponding to e_n.

We know that Q must intersect exactly one arc. It is clear that only the triangular region defined by e_n, O and O′ has an intersection with the half plane (delimited by L') in which Q lies, and that therefore Q must intersect the arc corresponding to e_n.

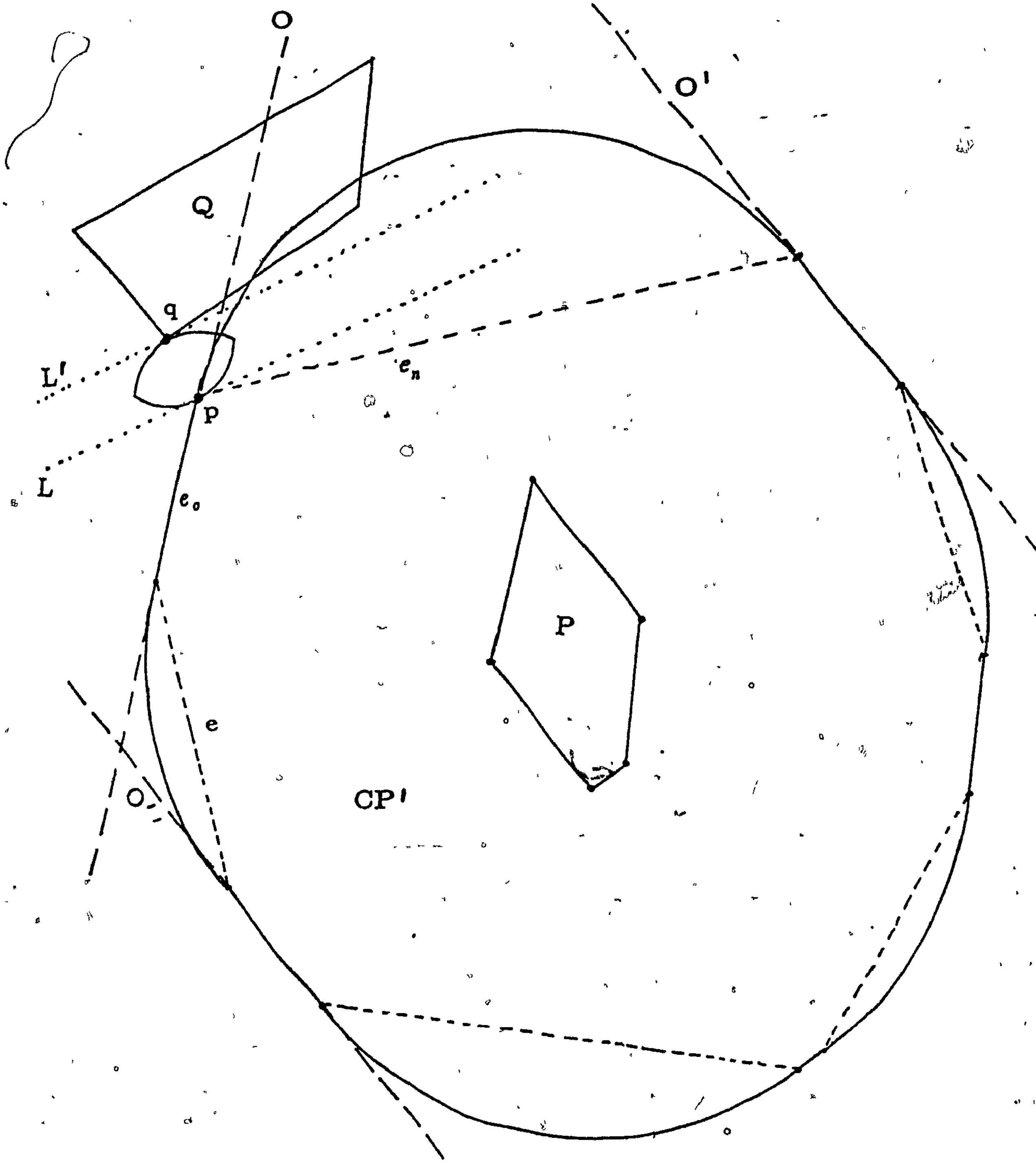


FIGURE 4.9

The other triangular regions do not have an intersection with the half plane of L'. If for example the triangle (e, O'', O) did have an intersection with the half plane above L' it would imply that e_0 will go above L (which is parallel to L') and that lune (p, q) will not be empty, which cannot happen.

Therefore in the case $I = Q \bigcap CP'$ = empty set , Q intersects exactly one dome of CP. We will now examine the case where $I = Q \bigcap CP'$ is not empty.

CASE 2 : $I = Q \bigcap CP' <> \emptyset$

We will now consider the case Q intersects CP' .

In this case we calculate CH (CP' U Q) . If it is equal to CP' then I = Q and Q is strictly in CP' - P. But if the CH is not CP' the set difference CH (CP' U Q) - (CP' U Q) forms k *pockets* each corresponding to an intersection point also called a *bridge point*, on the boundaries of the two polygons. The *lid* of each pocket is called a *bridge* and to each bridge corresponds exactly one bridge point, [Tou3].

We must now consider the subchains of Q that are outside CP', and for each of them from bridge point to bridge point , we see what arcs of CP this subchain intersects. How is this done ? See figure 4.10 for an illustration. We proceed counterclockwise and we start at the first (in counterclockwise order) bridge point. We identify the first (counterclockwise order) arc region and we repeat the following process for every arc region :

Let $L\,1_i$ and $L\,2_i$ be the two infinite half lines delimiting the arc region i, in counterclockwise order. Find the first edge of the Q-chain to intersect $L\,1_i$. We are now in the arc region i. Keep testing for intersection for the following edges with arc i. The first edge cutting $L\,2_i$ signifies that we are out of the arc region i. We repeat this for the next arc region i + 1 . We can keep an array or a doubly linked list

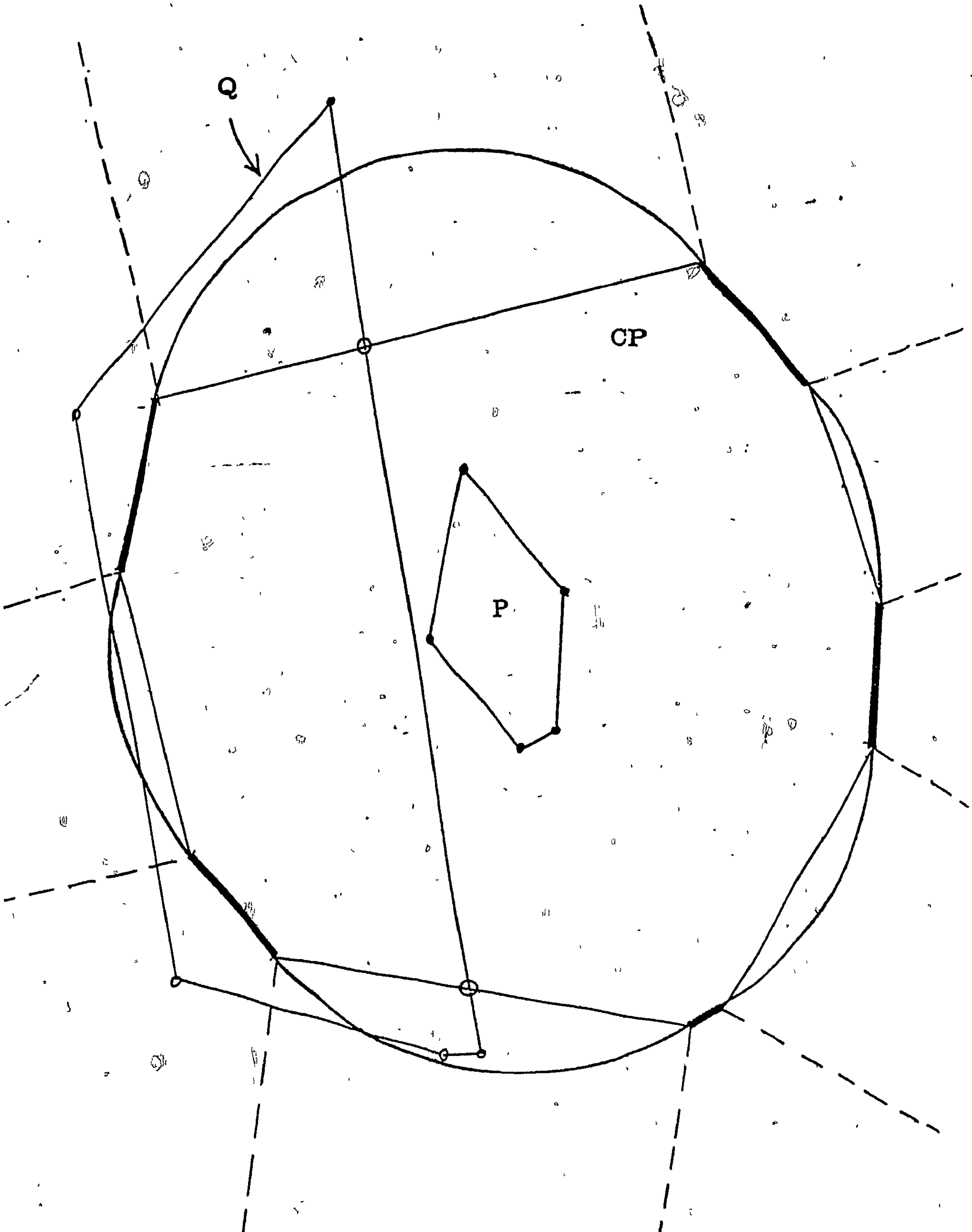


FIGURE 4.10

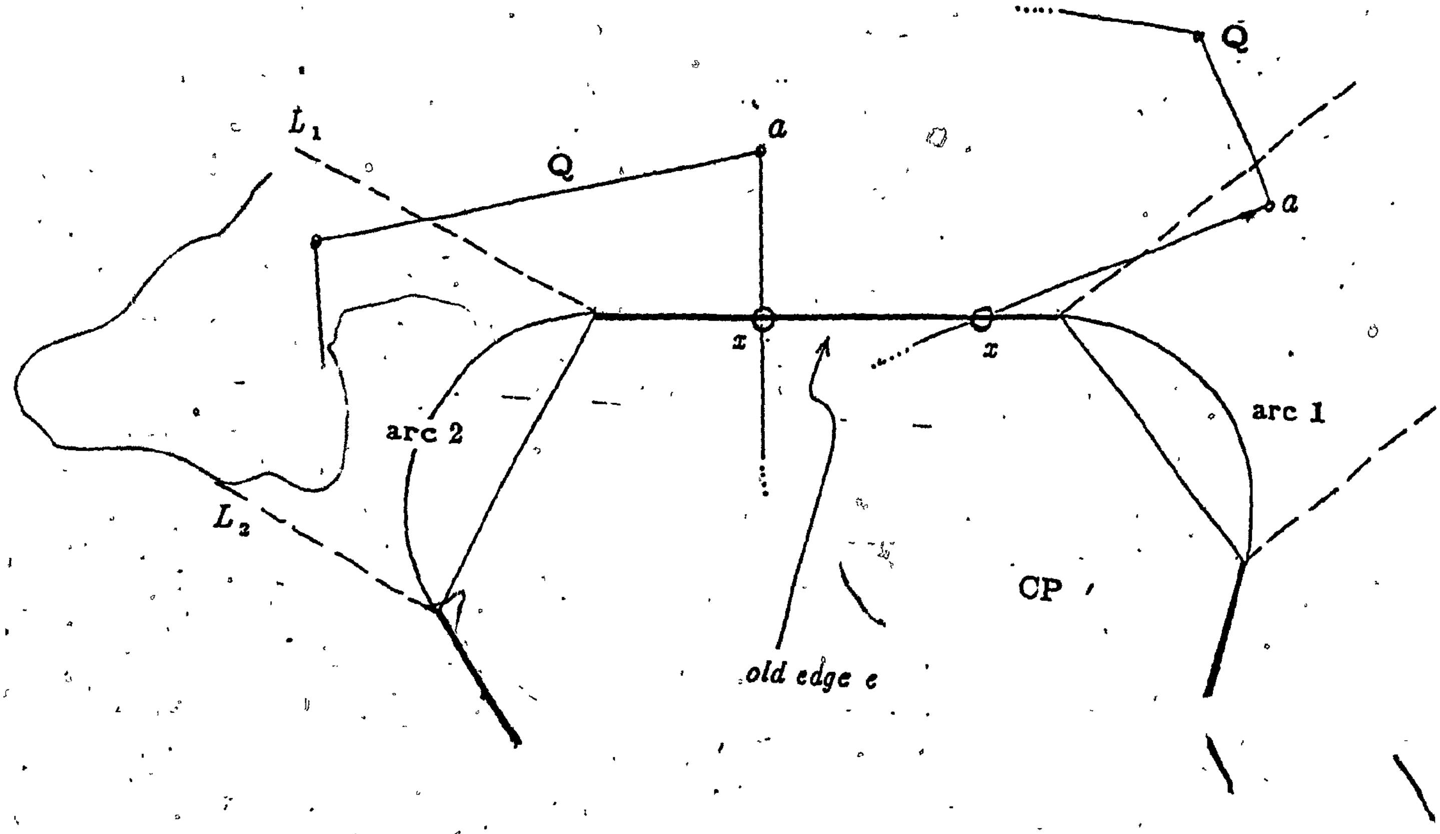


FIGURE 4.11

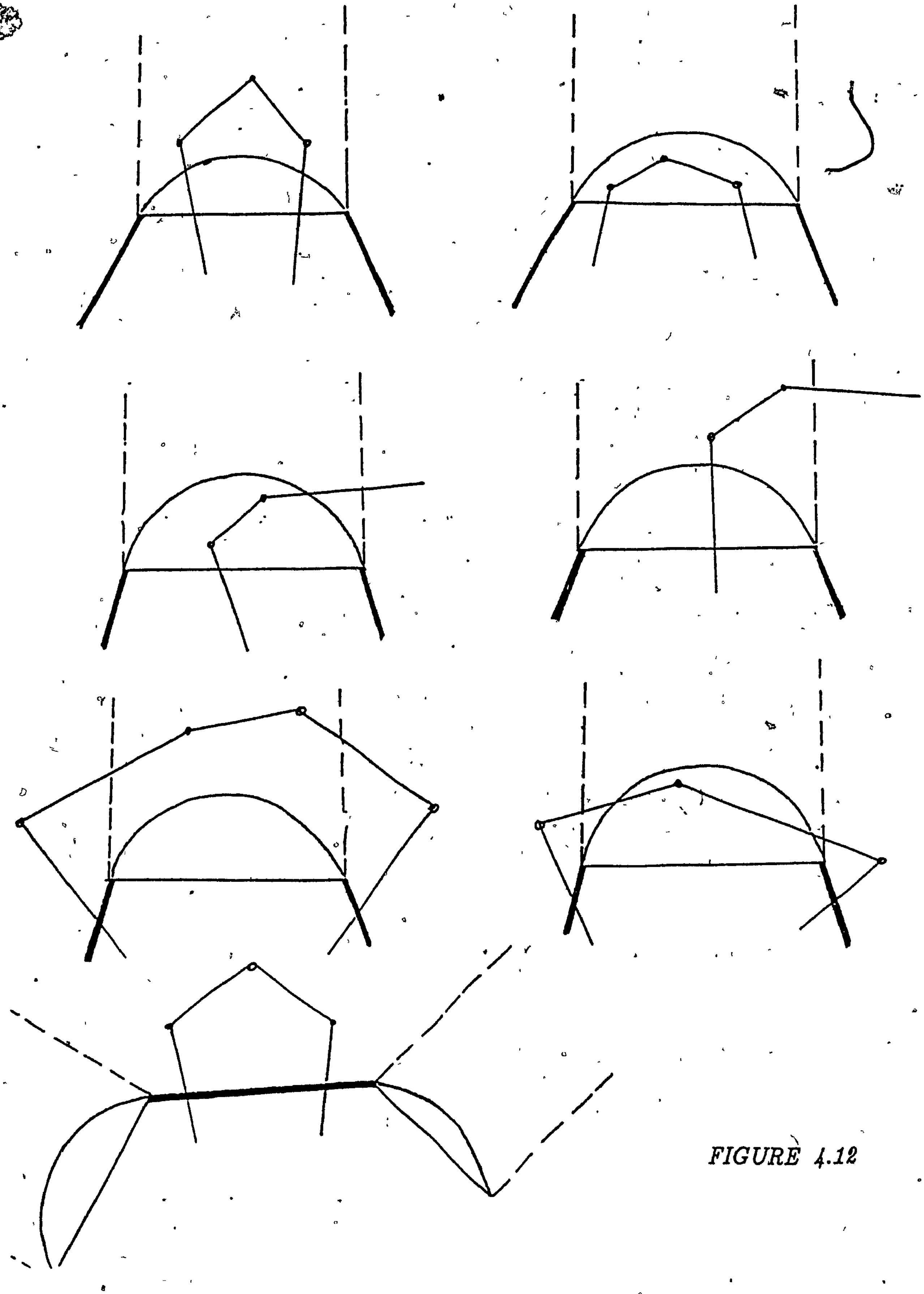


FIGURE 4.12

for the half lines.

Parenthesis :

Let a be the outside endpoint of a Q-edge cutting an old edge e of CP$'$. Point a cannot cross a neighbouring arc and intersect the old edge e at the same time, see figure 4.11. So if we go counterclockwise, starting at the bridge point x on the edge e, to see which arcs the Q-chain intersects, we mustn't worry about previous arcs (arc1) and we can start the testing at the next arc i.e. arc2. Some possible simple cases for Q-chains intersecting CP are depicted in figure 4.12.

4.2.5 - The algorithm

Input : two convex polygons P and Q,

a ladder S = [a, b] of length r.

Output : the intersection Q$'$ of CP and Q.

Assumption : CP and Q intersect.

begin

step 1 : compute CP and replace its arcs with new edges and obtain CP$'$.

step 2 : detect whether CP$'$ and Q intersect

step 3 : if CP$' \cap$ Q $= \emptyset$ then

- compute the minimum distance between the two polygons
- identify the dome corresponding to the point in CP$'$ that realizes the minimum distance with Q ; this dome intersects Q
- find the intersection points between the arc of the dome and Q, if there are any.

step 4 : if $CP' \cap Q <> \emptyset$ then

- identify the bridge points on the boundary
- for every outer subchain of Q , identify the intersection points with the arc in each region.

step 5 : merge the inner chains of CP and Q, determined by the intersection points found in steps 3 and 4.

end

4.2.6 - The analysis

Data structures :

We keep the edges and arcs of each polygon in an array and for each edge we specify if it is a new edge or an old one. Also , we keep an ordered array of the half lines delimiting the arc region.

Correctness :

We have proved in lemma 4, that in the case that Q and CP′ do not intersect , the minimum distance between them determines the dome of CP intersecting Q. We have also proved that for the more general case, for every outer portion of Q every arc region corresponding to it determines the intersection points.

Time complexity :

Computing CP′ directly from P can be done in linear time. In step 2, detecting the existence of an intersection between two convex figures can be done in sub-linear time using Chazelle and Dobkin's algorithm. In step 3, the minimum distance between two convex polygons can be obtained , together with the points realizing it in logarithmic time using Edelsbrunner's binary elimination technique. Then once a dome is identified finding the intersection points with Q , if there are any i is done in at most O(n) time. For the general case in step 4, since we have the outer subchains of Q (they are ordered but this does not really matter) and the corresponding arc regions, updating the intersection region is really a merge of two sorted lists each having a linear number of elements. Step 5 is also a linear merge of two sorted lists, if we leave pointers from the intersection points to the inner and outer chains in both directions. Therefore the total running time of the algorithm is bounded by O(n).

4.3 - The second problem

4.3.1 - Problem Statement

At this stage we have the region Q' in Q, in which the reachability regions are contained. We now encounter the problem of computing the intersection region between Q', obtained in the previous section, and UP. This intersection will be denoted by IQ and the reachability region in Q will be $RQ = Q' - IQ$. See figure 4.13 for an illustration.

4.3.2 - General description of the algorithm

To compute $UP \cap Q'$, we first replace all the arcs of UP, and the possible arcs of Q' with edges to obtain two convex polygons UP' and Q''. We then test for the intersection J between these two, using Chazelle and Dobkin's algorithm, [CD].

If J is empty then two cases arise : either UP and Q' are disjoint, which means that all of Q' is reachable for the endpoint b of the ladder $S = [a, b]$ or UP and Q' intersect in a region IQ and more precisely we prove that Q' intersects one dome of UP.

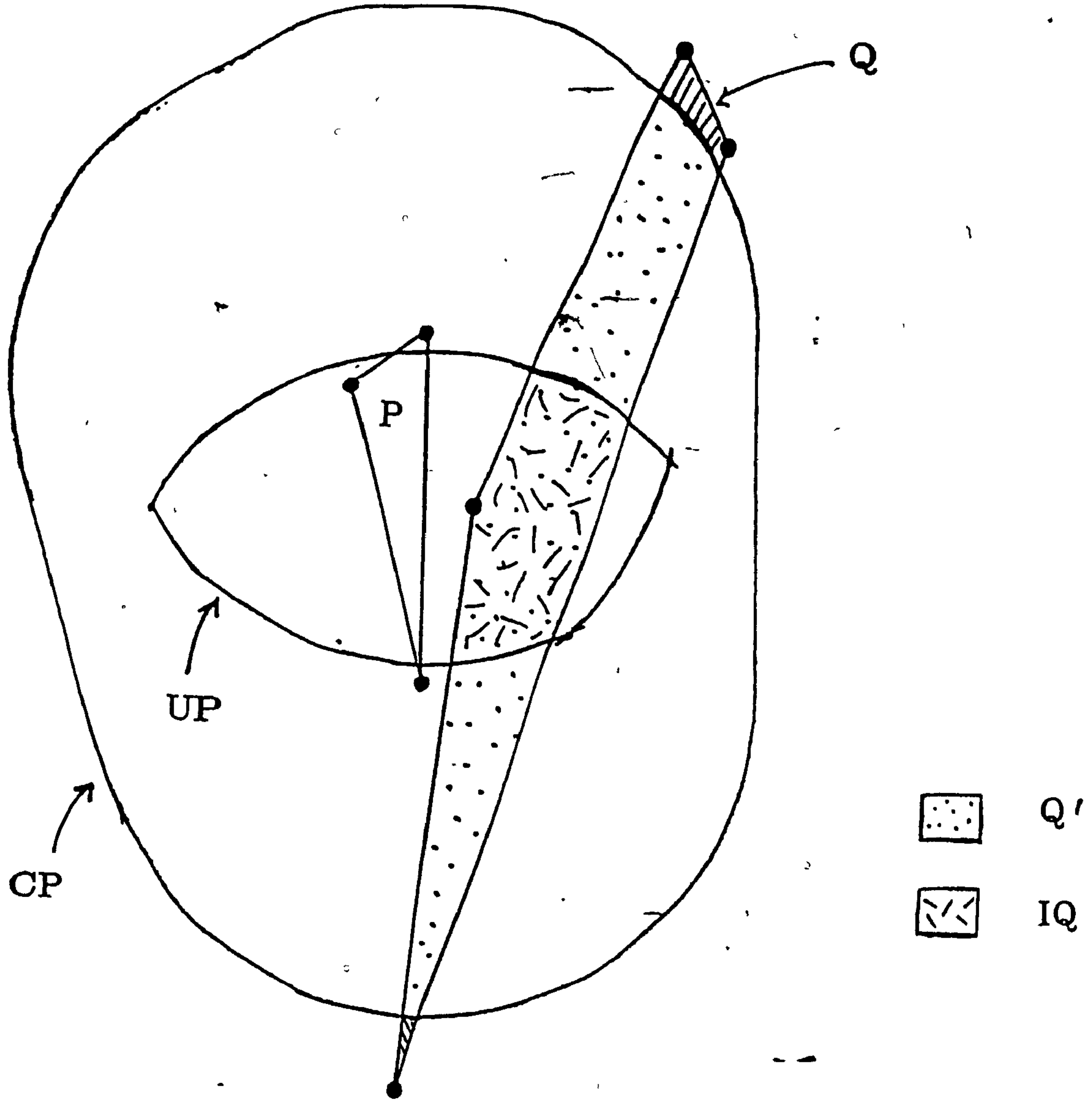


FIGURE 4.13

In the second situation, if J is not empty then to find IQ (we are in fact interested in RQ $= Q'$ - IQ) we partition the plane around UP in arc regions and for subchains of Q' we test for intersection with arcs in these regions. Let us first expose the different possibilities, by considering figure 4.14 . Q' contains the reachable region(s). Figure 4.14-a is not valid under the assumption that a reachability region exists in Q. Figure 4.14-d is also invalid because it means that Q' intersects P (since UP contains the centroid of P) and that therefore Q intersects P. Figures 4.14-b and 4.14-c only are valid. UP consists of arcs only and Q' may or may not have arcs.

4.3.3 - Preliminary results

We will prove a few theorems and make observations as we describe more fully the algorithm.

Facts :

- UP contains the centroid of P, therefore Q can only intersect the parts of UP that are outside P.
- Also an arc of Q' cannot intersect UP because an arc of Q' is an arc of CP and because UP is strictly in CP. Therefore only edges of Q' (that are not edges of CP) can intersect UP.

We replace the arcs of UP and of Q' to obtain two convex polygons UP' and Q'' respectively. We then test whether $J = UP' \cap Q''$ is empty or not. If J is empty we prove , in case 1, that if an intersection exists between UP and Q' then it must be that Q' intersects exactly one arc of UP. If J is not empty , we are in the more general situation treated in case 2.

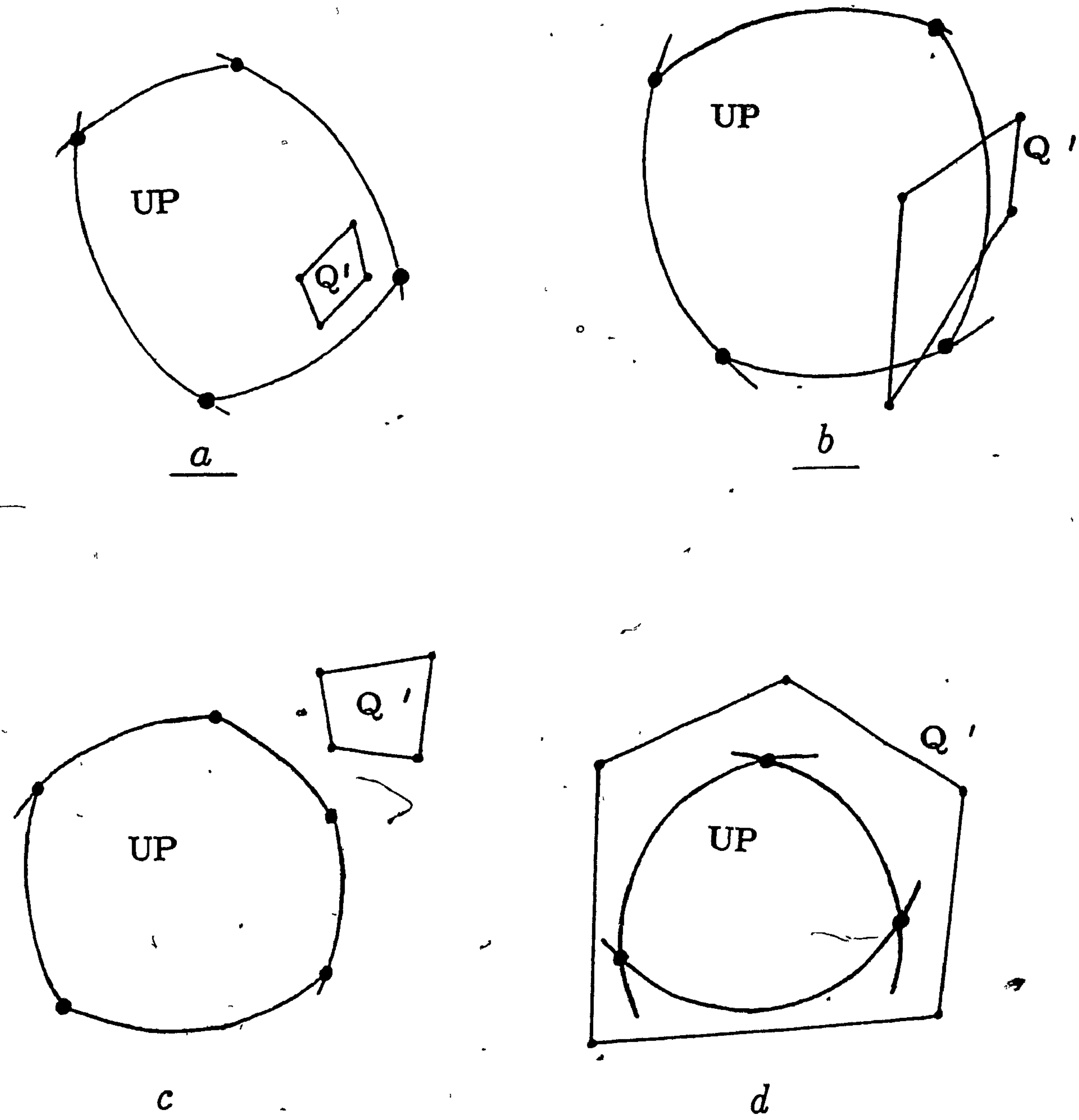


FIGURE 4.14

CASE 1 : $J = UP' \bigcap Q'' =$ empty set

Lemma 5 :

UP' and Q'' are disjoint if and only if UP' and Q' are disjoint.

Proof :

only if : The arcs of Q', if any, are the arcs of CP. UP' and Q'' are disjoint. we know that no arcs of Q' intersects UP (therefore UP') since UP is strictly inside CP. We now bring back the arcs of Q' to get Q'. We know that an arc a of Q' does not intersect UP' since a is a portion of an arc of CP. Since Q'' does not intersect UP', edge e does not intersect UP'. So when we restore the dome of Q' the dome itself doesn't cut UP' for if it did then UP' must have intersected e or a, which is contrary to our assumption . Also UP' cannot be strictly included in the dome . We now prove this statement. Suppose UP' were strictly inside a dome. UP contains the centroid of P and more precisely UP' contains it. Therefore UP' and P must have some intersection and cannot be disjoint, therefore part of P must also be in the dome, and since the dome is part of Q (the original Q) this implies that P and Q intersect which contradicts the initial assumption of two disjoint polygons.

if : straightforward, since Q'' is contained in Q'. QED

Corrollary : UP' and Q'' intersect if and only if UP' and Q' intersect.

We can prove using a similar reasoning the following theorem and derive its corrollary :

Lemma 6 : UP and Q'' are disjoint if and only if UP and Q' are disjoint.

Corrollary : UP and Q'' intersect if and only if UP and Q' intersect.

Lemma 7 : If UP' and Q'' are disjoint, and if UP and Q' intersect, then Q' intersects exactly one arc of UP.

Proof : Q' cannot be completely under a dome because then it would be unreachable which contradicts our initial hypothesis. Since Q' and UP intersect and that the boundary of UP is composed of arcs, Q' must intersect at least one arc of UP. Suppose Q' intersects more than one arc of UP. Let a be one such arc, and refer to figure 4.15. Let L_a be the line supporting the edge of UP' corresponding to arc a. All of UP' is at one side of L_a and arc a is on the other side. Q'' does not intersect UP' but intersects a . Therefore the interior of Q'' must be on the same side of L_a as arc a is. For Q'' to intersect other arcs as well , Q'' would have to be non convex which is a contradiction. QED

We now want to find which arc of UP intersects Q' , if any such arc exists. In any case we also know that if there exists an intersection , it cannot be with an arc of Q' (or a new edge of Q' i.e. an edge of CP) but with a real edge of Q'.

We calculate the minimum distance t realized by u in UP' and v in Q'', using Edelsbrunner 's algorithm. And let L_u and L_v be the two parallel lines tangent to lune (u, v) at points u and v respectively.

case 1.a : u is a *point* on an edge of UP'

If u is a point on an edge e_u of UP' , then v must be a vertex of Q''. To edge e_u corresponds a dome d_u and an arc a_u . We test whether v is below or above the dome. Notice that all of UP except a_u is below L_u .

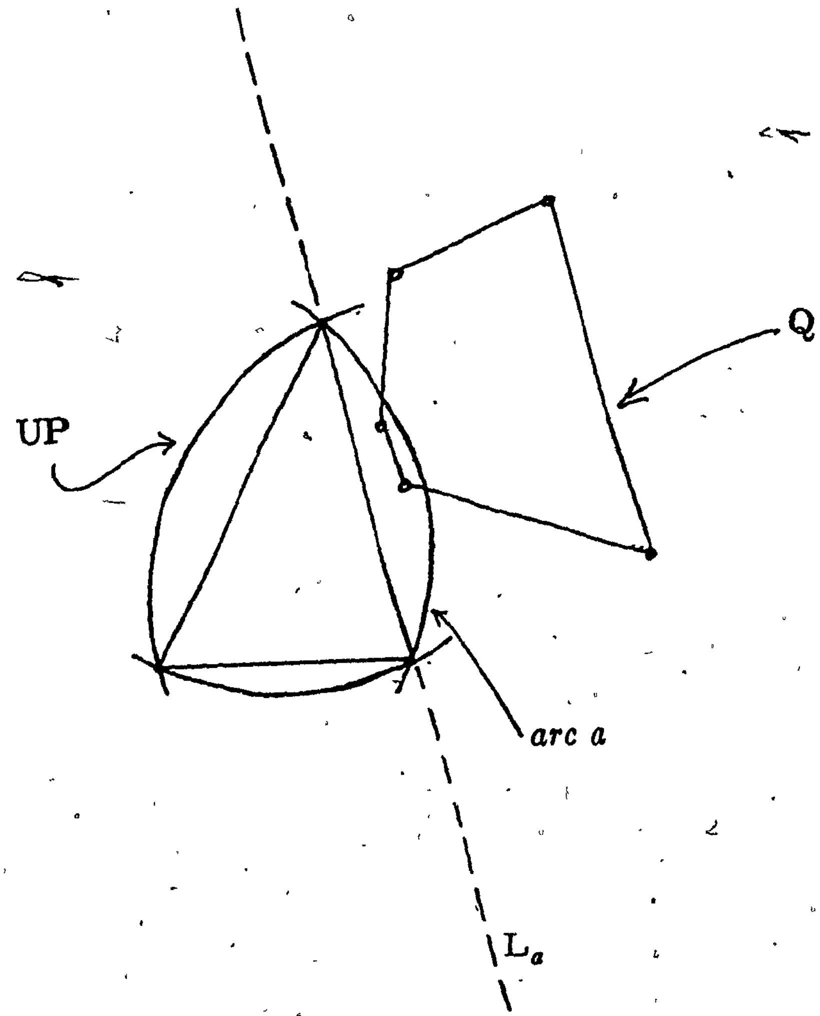


FIGURE 4.15

If v is above d_u, that is outside the curvature of the arc a_u, then we can easily see that Q'' and UP do not intersect, since only a_u is above L_u. If v is below the dome then a_u is the arc intersecting Q'', and going up from v on its left and its right edges, we can find the two intersection points.

case 1.b : u is a *vertex* of UP'

If u is a vertex of UP', then v can either be a vertex or a point on an edge of Q''. If v is a vertex of Q'' (consider figure 4.16) then let a_{u-1} and a_u be the two arcs of UP adjacent to u, in counterclockwise order. If L_v cuts none of a_{u-1} or a_u then Q'' and UP do not intersect, therefore Q' and UP don't intersect. If L_v cuts any of a_{u-1} or a_u then this arc may cut Q'. To see if it does we simply check the edges of Q', starting at v, in counterclockwise direction if the arc is a_{u-1}, and in cw direction if it is a_u. The reasoning is similar if v is a point on an edge of Q''.

CASE 2 : J = UP' $\cap$ Q'' < > empty set

In this case Q' may intersect more than one arc in UP. We find the bridge points after having CH (UP' U Q''). We build arc regions for UP, no need to build them for Q' since no arc of Q' intersects UP. In fact, as we look for intersection points, we cut off the reachable regions of Q'.

Suppose that we have computed the bridge points between Q'' and UP'. In a first step we must reinsert the arcs of Q'. For this we look at the new edges of Q''. If a new edge of Q'' does not intersect the boundary of UP' then replacing it with its arc does not change this (i.e. the arc will not cut UP'). In such a case this new edge is outside UP', it cannot be completely inside UP' because then the corresponding arc of CP, will be inside UP and intersect UP which cannot happen

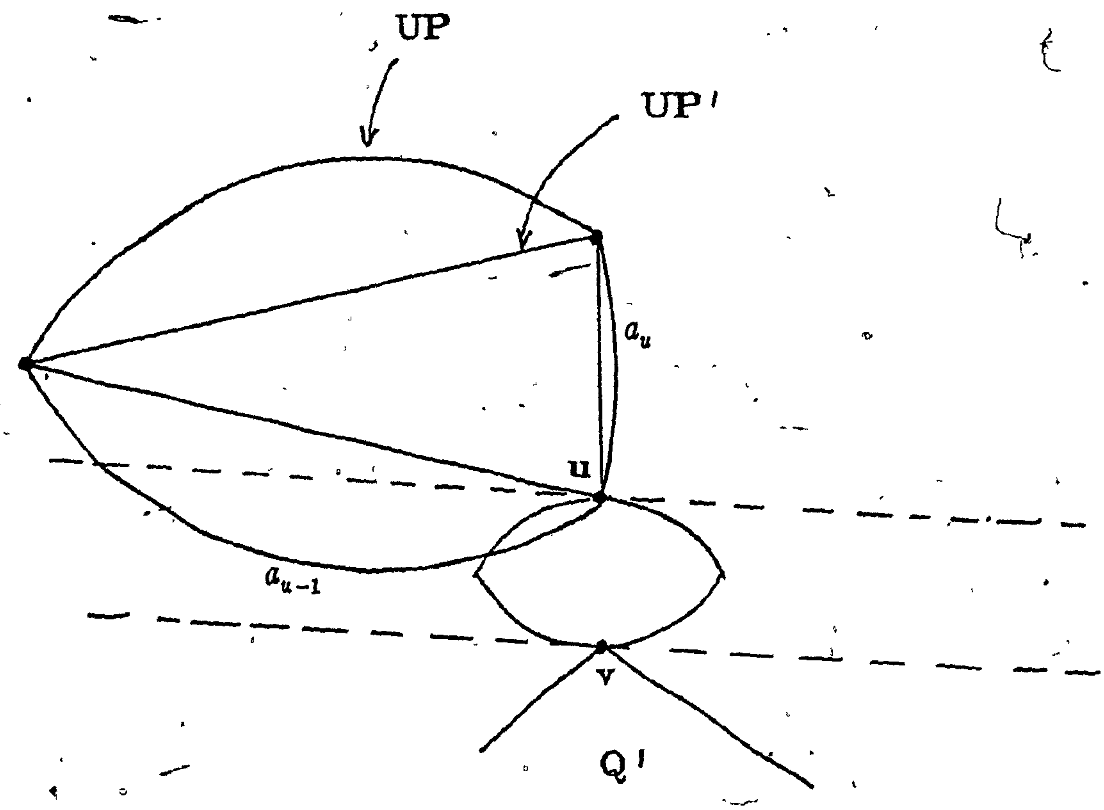


FIGURE 4.16

as we mentionned it before.

Lemma 8 : If a new edge e of Q'' intersects boundary (UP'), we say that it must be cut twice exactly.

Proof :

See figure 4.17 for an illustration. We know that when a segment intersects a convex polygon it intersects it in at most two points. Let us show that e will be cut more than once : e is a new edge , suppose by contradiction that it cuts boundary(UP') in one point only. Then an endpoint of e must be inside UP'(if the intersection point is a vertex of UP' it counts for two points) and the endpoint of a new edge is a point (possibly the endpoint) on an arc of boundary(CP). This is a contradiction since boundary(UP') is strictly inside boundary(CP), QED.

Between the two bridge points on a new edge e, there is an outer subchain of UP'. By replacing e with its corresponding arc a, all the subchain of UP' will be under the curvature of a. Therefore two bridge points pop off. So the two adjacent outer subchains of Q'' become a single outer subchain of Q'.

Once we have replaced all the new edges of Q'' with arcs of Q'. We test for intersection points with arcs of UP by going from bridge point to bridge point and from one arc region to the following.

4.3.4 - The algorithm

input : - a convex figure UP that is the intersection of n circles of radius r , about vertices of a convex polygon P

- a convex figure Q' whose boundary may contain arcs of CP, obtained from the previous algorithm in section 4.2.5.

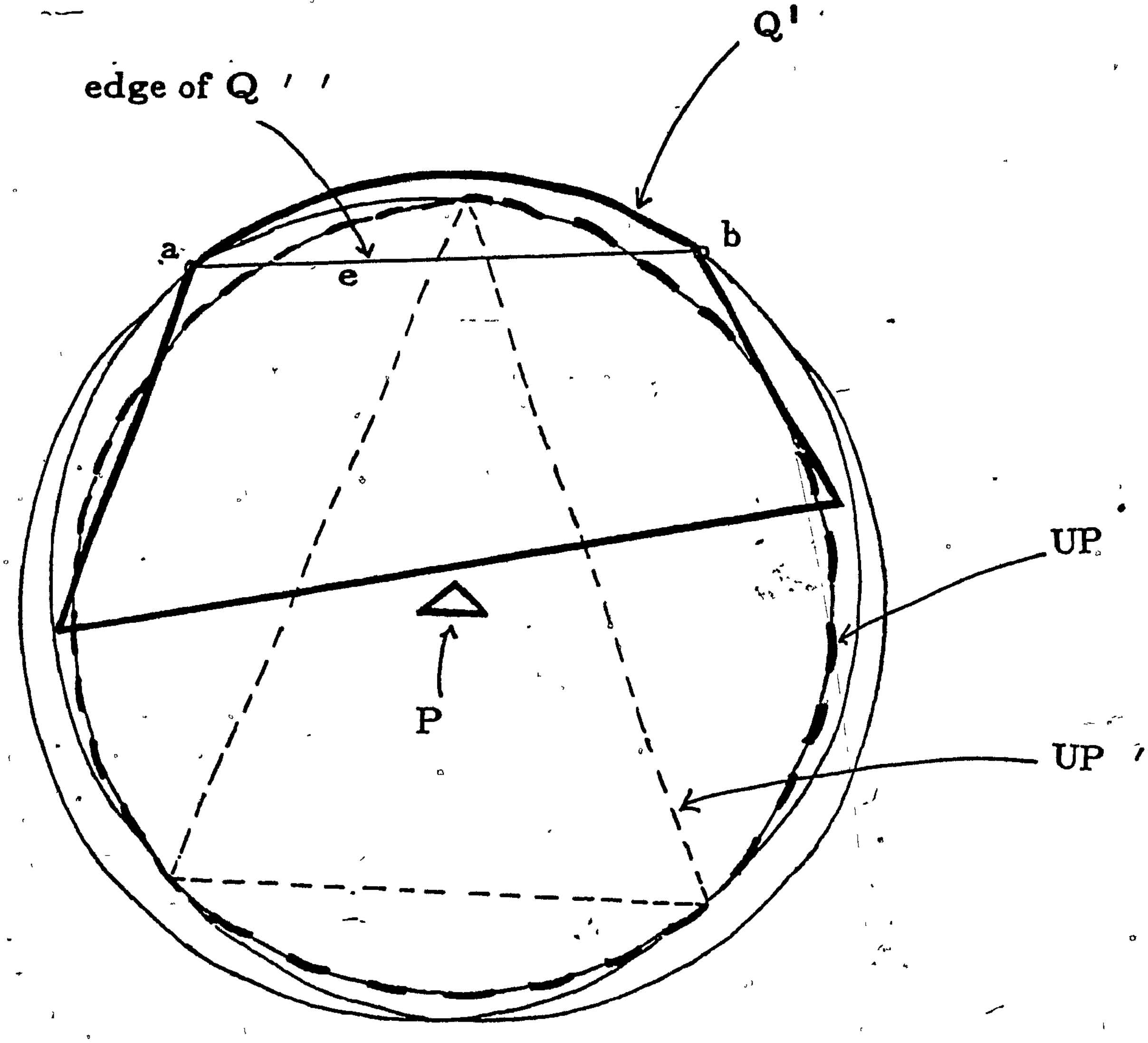


FIGURE 4.17

output : the reachability region for endpoint b of a ladder $S = [a, b]$,

$Q' - (UP \cap Q')$

Algorithm :

begin

step 1 : replace arcs of UP and of Q' to obtain UP' and Q''

step 2 : test whether $UP' \cap Q''$ intersect

step 3 : if $UP' \cap Q' = \emptyset$ then

- compute the minimum distance between UP' and Q''
- test whether the arc of UP, corresponding to a vertex of UP' realizing the minimum distance, intersects Q'', if so compute the intersection points , otherwise UP and Q' are disjoint.

step 4 : if $UP' \cap Q'' <> \emptyset$ then

- compute $CH(UP' \cup Q'')$
- replace new edges of Q'' with its arcs
- build arc regions around UP'
- for every outer subchain of Q', go from one arc region to the next and identify the intersection points

step 5 : merge the two lists of UP and Q' and take only the regions determined by the intersection points, found in steps 3 and 4, and the outer chains of Q'.

end.

4.3.5 - The analysis

data structure :

An array to keep track of vertices and edges (or arcs) of the two polygons. After having computed the bridge points, we can for each edge keep track of these intersection points. An array or a doubly linked list to keep track of the boundaries of the arc regions.

correctness :

We have proved that in step 3 the minimum distance determines the intersection points for the case that UP′ and Q′′ are disjoint. If this is not the case in step 4, we have proved that reinserting arcs of Q′ and between two bridge points for an outer subchain of Q′, and testing each arc in consecutive arc regions yields the intersection points and thus we already cut off the parts of Q′ that are not in UP and get up to n non convex reachability regions.

time complexity :

Step 1 is linear . Step 2 is logarithmic using Chazelle and Dobkin′s algorithm. In step 3 we compute the minimum distance in log(m + n) time using Edelsbrunner's algorithm. Testing for intersection between one arc and one convex figure is at most O(n). In step 4 , the CH can be computed using the rotating callipers of Toussaint in O(m + n) time . Replacing arcs and building arc regions is a sequential process feasable in O(n) time. We use a linear running time algorithm to compute the bridge points in sorted order . And since for every outer subchain of Q′ we know which arc region we are in , the testing for intersection is merely a merge of these two lists , and since the number of edges and arcs in each list is bounded by n, this step of the algorithm is linear. Step 5 is also a mere traversal of two sorted lists and is carried out in O(n) time.

CHAPTER 5

5.1 - Problem statement

Given two convex polygons $P = \{p_1, p_2, \ldots, p_m\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$ and a line segment $S = [a, b]$ of length r, we would like to compute the reachability regions in P and Q for the endpoints a and b respectively, with the constraint that a remains within the boundaries of P and b within those of Q. Stating the problem more precisely : Given 2 disjoint convex polygons P and Q find the union (set) of reachable regions PR in P and QR in Q such that :
For every point $x \in PR$, there exists a point y in QR such that $d(x, y) = r$ and
For every point $s \in QR$, there exists a point t in PR such that $d(s, t) = r$.

Calculating the reachable region is the same procedure for each polygon, in other words the reachability region of one does not influence the reachability region of the other. The description of the reachability region(s) in a polygon is that :

(1) the boundary may contain *arcs*

(2) it is *not* necessarily a *convex* region

(3) it may be *disconnected*, as we will see later it can have O(n) parts, each of which is a reachable region.

5.2 - Preliminary results

We will now show a few results before describing the algorithm in detail.

definition :

PR = { p in P | for every p in PR , there exists q in Q | d(p, q) = r }

QR = { q in Q | for every q in QR , there exists p in P | d(q, p) = r }

Theorem 1 :

The reachability region of one polygon does not influence the reachability region in the other polygon . Let PR and QR denote these regions, for every point x in PR, there exists point y in QR, such that $d(x, y) = r$ and for every point s in QR, there exists point t in PR, such that $d(s, t) = r$.

Proof :

we know that : $\forall$ point $x \in PR$, there exists y in Q st $d(x, y) = r$ by definition, now if y is in Q - QR then there will be no point in P, and a fortiori no point in PR, such that their distance = r, QED.

Let us now show the condition of existence of a reachability region in P and in Q. Such a region exists *if and only if* the length of the line segment is greater than the minimum distance and smaller than the maximum distance between the two polygons, and consider figure 5.1 .

Theorem 2 :

A line segment $S = [a, b]$ of length r can be placed such that $a \in P$ and $b \in Q$ *if and only if* $dmin \leq l \leq dmax$

Proof :

only if : $a \in P$, $b \in Q$, trivial because any pair (p, q) of P and Q is such that $dmin \leq d(p, q) \leq dmax$ and in particular (a, b).

if : Here we use the convexity of the two polygons. Here is a constructive proof:
if $dmin < l < d\,(n1, m2)$ then place a on n1 and b in (n2, m2)
if $d\,(n1, m2) < l < dmax$ then place b on m2 and a on (m1, n1)

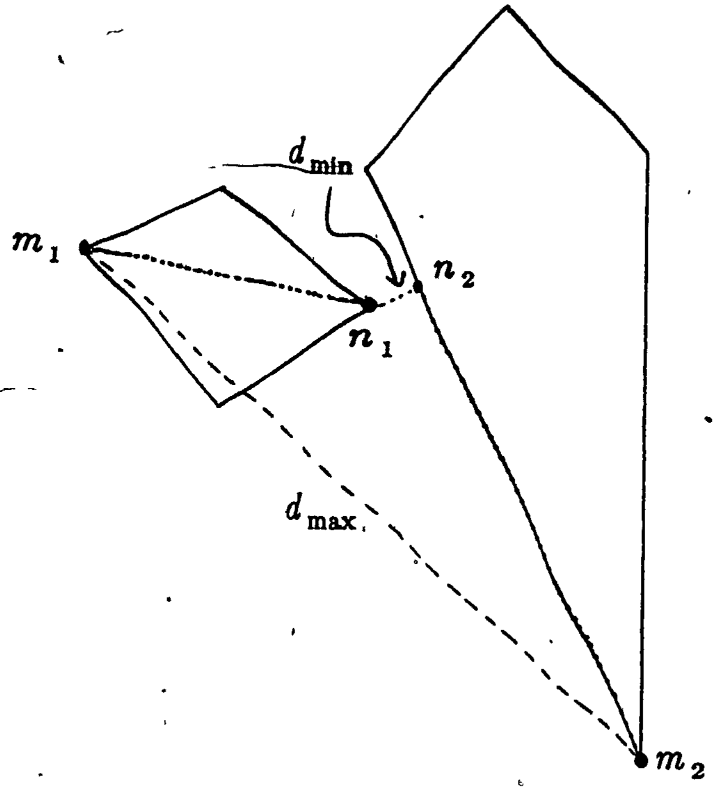


FIGURE 5.1

In other words the pairs of points (x, y) st x belongs to [m1, n1] and y belongs to [n2, m2] define (or cover) all possible distances between dmin and dmax continuously, QED.

Therefore if we already know the values of dmin and dmax, we could in constant time answer the query : can a given line segment l = [a, b] be placed such that a is in P and b in Q.

To compute dmin we use Edelsbrunner's algorithm, [Ede], which runs in sublinear time and is O(log m + log n) and to compute dmax we use Battacharya and Toussaint's, [BT1], which is linear in the number of vertices.

5.3 - The algorithm

We start with a few definitions and notations.

1- CP and CQ denote respectively the *convex hull* of circles of a given radius about each of the vertices of P and Q.

2- UP and UQ denote respectively the *unreachable* regions in P and Q.

3- RP and RQ denote the *reachable* regions in P and in Q.

We now state the algorithm which is a call to different procedures defined in previous chapters.

Input :

- two convex polygons P= $\{p_1, p_2, \dots, p_m\}$, Q = $\{q_1, q_2, \dots, q_n\}$.
- a line segment of length r , S = [a , b].

Output :

Reachability regions, RP and RQ, in P and Q for point a in P and point b in Q.

Algorithm :

begin

step 1 : calculate dmin (P,Q) and dmax (P,Q)

step 2 : if dmin $\leq$ l $\leq$ dmax then continue to step 3

else stop.

step 3 : if l $=$ dmin then the reachability region is the pair of points

(p, q) of P and Q that realize dmin.

Idem if l$=$ dmax, (we may have $>$1 pair of points)

step 4 : Compute the reachability region in Q :

1. calculate CP

2. intersect CP and Q, obtain Q′

(we know that this intersection is not empty because

dmin $\leq$ l $<$ dmax)

3. calculate UP

4. if UP $=$ empty set then RQ $=$ Q′

else

1. intersect UP and Q′, obtain Q″

2. if Q″ $=$ empty set then RQ $=$ Q′

else RQ $=$ Q′ - Q″

step 5 : compute the reachability region in P, as in *step 4*.

end.

5.4 - The analysis

Correctness :

The correctness follows from theorems 1 and 2 and the results of the previous chapters.

Complexity :

In step 1 , the minimum and the maximum distances can be computed in O(log m + log n) and O(n) time respectively, using Edelsbrunner and Toussaint and Battacharya's algorithms. The test in steps 2 and 3 takes constant time to perform. Step 4 has an overall complexity of O(n) as proved in chapters 3 and 4. And step 5 is the repetition of the same procedures for the second polygon. The total running time of the algorithm is thus linear.

CHAPTER 6

We have solved the problem of finding all the reachability regions in two convex polygons, for the tips of a ladder, constrained to remain within the boundaries of the polygons, in time linear in the number of vertices. This involved the calculation of the intersection of circles of equal radii about (convex) polygon vertices in linear time. This in turn suggested the extension of the algorithm to one for the general case of circles of different radii, also running in linear time.

We presented an algorithm to compute in linear time the intersection region between two convex regions, whose boundaries may be composed of arcs, in the particular context of our problem. The problem, described in the introductory chapter, of separating two star-shaped polygons, can be generalized to an arbitrary number of star-shaped polygons. One interesting open question is how to compute, efficiently, the reachability region in n convex polygons for the vertices of a n-gon, free to move but with the restriction that each of its n vertices remain in a polygon. Another interesting problem that arises is the computation of these regions for a ladder in two simple polygons. The extension of these problems in the three and higher dimensional spaces remains open , such as moving a ladder in two polyhedra, a triangle in three polyhedra , and a polyhedron in n polyhedra.

REFERENCES

[Bes]
Besicovitch A. S., "*The Kakeya Problem*", Amer. Math. Monthly, 70 (1963), 697-706.

[BO]
Bentley J.L. and Ottmann T. A., "*Algorithms for reporting and counting geometric intersections*", IEEE Trans. Comput., vol. C-28, (Sept 1987), 643-647.

[Bro]
Brown Kevin Q., "*Geometric Transforms for Fast Geometric Algorithms*", PhD thesis, Carnegie-Mellon University, (Dec. 1979), 54-57.

[BT1]
Bhattacharya B. and Toussaint G. T., "*A linear algorithm for determining the translation separability of two simple polygons*", Tech. Rep. No. SOCS 86.1, McGill University (Jan. 1986).

[BT2]
Bhattacharya B. and Toussaint G. T., "*Efficient algorithms for computing the maximum distance between two finite planar sets*", Journal of Algorithms, 4, (1983), 121-136.

[CD]
Chazelle B. and Dobkin D., "*Detection is easier than computation*", Proc. Twelfth Annual ACM Symposium on Theory of Computing, (April 1980) 146-153.

[Cha1]
Chazelle B. and al., "*The complexity and decidability of separation*", Tech. Rep. CS-83-84, University of Waterloo, (Nov. 1983).

[Cha2]
Chazelle B., "*Computational geometry and convexity*", Ph.D. thesis, Carnegie-Mellon University, (July 1980).

[Cun]
Cunningham F., JR., "*The Kakeya problem for simply connected and for star-shaped sets*", Amer. Math. Monthly, 78 (1971) 114-129.

[CZZ]
Chien R.T., Zhang L. and Zhang B., "*Planning Collision-free paths for robotic arm among obstacles*", IEEE Trans. on Pattern Anal. and Machine Intelligence PAMI-6 (1984) 91-96.

[Daw1]
Dawson R. J., "*On the mobility of bodies in R^n*", manuscript, Cambridge University, England.

[Daw2]

Dawson R. J., "*On removing a ball without disturbing the others*", Mathematics Magazine, vol. 57, no. 1, (Jan. 1984), 27-30.

[Ede]
Edelsbrunner H., "*Computing the extreme distances between two convex polygons*", Journal of Algorithms 6,(1985) 213-224.

[Gol]
Goldberg, M., "*A solution of problem 6-11: Moving furniture through a hallway*", SIAM Review, 11 (1968) 75-78.

[GY]
Guibas L.J. and Yao F.F., "*On translating a set of rectangles*", Proc. Twelfth Annual ACM Symposium on Theory of Computing, 1980; 154-160.

[HJW1]
Hopcroft J., Joseph D. and Whitesides S., "*On the movement of robot arms in two-dimensional bounded regions*", Cornell Univ., Dept. Comp. ci., TR82-486 (1982a).

[HJW2]
Hopcroft J., Joseph D. and Whitesides S., "*Determining points of a circular region reachable by joints of a robot arm*", Cornell Univ., Dept. Comp. Sci., TR 82-516 (Oct1982b).

[HJW3]
Hopcroft J., Joseph D. and Whiteside S., "*Movement problems for 2-dimensional linkages*", SIAM J. Comput. 13 (1984) 610-689.

[How]
Howden W.E., "*The sofa problem*", Computer Journal, vol 11 (1968) 299-301.

[Jar]
Jarvis R. A., "*On the identification of the convex hull of a finite set of points in the plane*", Info. Proc. Letters **2**, (1973), 18-21.

[KZ]
Kant K. and Zucker S. W., "*Trajectory planning problems, I: Determining velocity along a fixed path*", Seventh Int. Conf. on Pattern Recognition vol.1 (1984), 196-198.

[LP]
Lee D. T. and Preparata F. P., "*An optimal algorithm for finding the kernel of a polygon*", Journal of the ACM **26**, (1979), 415-421.

[LS]
Algorithm for a ladder moving in two-dimensional Space Amidst Polygonal Barriers", Proc. of the Symp. on Computational Geometry, Baltimore, (June 1985), 221-227.

[Man]
Mansouri M., "*On translating convex polyhedra in 3D space*", notes, (Dec 1984).

[McT]
McKenna Michael and Toussaint Godfried T., "*Finding the minimum vertex distance between two disjoint convex polygons in linear time*", Comp. and Maths. with Applications, vol. 11, No. 12, (1985), 1227-1242.

[Mel]
Melville R., "*An implementation study of two algorithms for the minimum spanning circle problem*", in Computational Geometry, G. T. Toussaint, Ed., North Holland, 1985.

[Mos]
Moser L., "*Problem 6-11 : Moving furniture through a hallway*", SIAM Review,8 (1966) 381.

[MT]
Mansouri M. and Toussaint G. T. , "*Translation queries for convex polygons in the plane*", Proc. IASTED Conf. on Robotics and Automation, Lugano, Switzerland, (June 1985).

[O'R]
O'Rourke J., et al., "*A new linear algorithm for intersecting convex polygons*", Computer Graphics and Image Processing, Vol. 19,(1982), 384-391.

[OSY1]
O' Dunlaing C., Sharir M. and Yap C.K., "*Retraction: A new approach to motion planning*", 15th ACM STOC (1983), 208-220.

[OSY2]
O' Dunlaing C., Sharir M. and Yap C.K., "*Generalized Voronoi Diagrams for moving a ladder:I . Topological analysis* " , Courant Institute Tech. Rep. 139 (Nov.1984).

[OSY3]
O' Dunlaing C., Sharir M. and Yap C.K., "*Generalized Voronoi Diagrams for moving a ladder :II . Efficient construction of the Diagram*", Courant Institute Tech. Rep. 140 (Nov.1984).

[OW]
Ottman T. and Widwayer p., "*On translating a set of line segments*", Computer Vision , Graphics, and Image Processing, vol. 24,(1983) 382-389.

[OY]
O' Dunlaing C. and Yap C.K., "*The Voronoi method for motion planning: I. The case of a disc*", Courant Institute Tech. Rep. 53 (Mar. 1983).

[PM]
Preparata F. P. and Muller D. E., "*Finding the intersection of n half-spaces in time O(nlogn)*", Theoretical Computer Science 8, (1979), 45-55.

[Sch]
Schoenberg I.J., Mathematical time exposures, 168-184.

[Seb]

Sebastian J., "*A solution to problem 6-11: moving furniture through a hallway*", SIAM Review 12 (1970), 582-586.

[SH]
Shamos M.I. and Hoey D. "*Geometric intersection problems*", Proc. 17th IEEE Annu. Symp. Found. Comput. Sci. (Oct. 1976), 208-215.

[Sha]
Shamos M.I., "*Computational Geometry*", Ph.D. thesis, Yale University, (1978).

[SiS]
Sifrony S. and Sharir M., "*A new efficient motion planning algorithm for a rod in polygonal space*", Proc. of the Second Annual Symposium on Computational Geometry, New York (June 1986), 178-186.

[SS1]
Schwartz J.T. and Sharir M., "*On the piano movers' problem: I. The special case of rigid polygonal body moving amidst polygonal barriers*", Comm. Pure Appl. Math. vol 36(1983a) 345-398.

[SS2]
Schwartz J.T. and Sharir M., "*On the piano movers' problem: II. "General techniques for computing topological properties of real algebraic manifolds*", Adv. Appl. Math. 4 (1983b), 298-351.

[SS3]
Schwartz J.T. and Sharir M., "*On the piano movers' problem: III. Circular bodies moving amidst polygonal barriers*", Int. J. Robotics Research 2 (1983c), 46-75.

[SS5]
Schwartz J. T. and Sharir M., "*On the piano movers' problem : V. The case of a rod moving in three-dimensional space amidst polyhedral obsatcles*", New York University, Courant Institute, Robotics Research Report 13, (1983d).

[ST]
Sack J.-R. and Toussaint G.T., "*Movability of objects*", IEEE International Symposium on Information Theory, St. Jovite, Canada,(Sept. 1983).

[Str]
Strang G., "*The width of a chair*", Amer. Math. Monthly 89 (1982) 529-534.

[TE]
Toussaint G. T. and ElGindy H. A.,"*Separation of two monotone polygons in linear time*", Robotica, vol. 2, (1984), 215-220.

[Tou1]
Toussaint G.T.,"*Movable Separability of sets*", in Computational Geometry, Ed., G. T. Toussaint, North Holland, (1985).

[Tou2]
Toussaint G.T., "*The complexity of movement*", IEEE International

Symposium on Information Theory, St. Jovite, Canada,(Sept. 1983).

[Tou3]
Toussaint G.T., "*Solving geometric problems with the rotating calipers*", Proc. MELECON, Athens, Greece, (May 1983).

[Tou4]
Toussaint G.T., "*Computational geometric problems in pattern recognition*", Pattern Recognition Theory and Applications, Ed. J. Kittler, Oxford, (April 1981), 73-91.

[TS]
Toussaint G. T. and Sack J.-R., "*Some new results on moving polygons in the plane*", Proc. Robotic Intelligence and Productivity Conf., Detroit, Michigan, Nov. 18-19 (1983), 158-163.

[TV]
Tarjan R.E. and Van Wyk C.J., personal communication.

[Whi]
Whitesides S. H.,"*Computational geometry and motion planning*", in Computational Geometry, Ed., G. T. Toussaint, North Holland, (1985).

[Yap1]
Yap C. K., "*Algorithmic motion planning*", New York University, Courant Institute of Mathematical Sciences (Sept. 1985).

[Yap2]
Yap C. K., "*Coordinating the motion of several discs*", TR-105, New York University, Courant Institute of Mathematical Sciences (Feb. 1984).