How to Divide a Polygon Fairly

Bernard Chazelle

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

April 1982

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-82-120

12

How to Divide a Polygon Fairly

Bernard Chazelle

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

April 1982

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539.

Abstract

Let P be a simple polygon with N vertices, each assigned a weight c_i $(0 \leq i \leq N)$ with $c_i \in \{0,1\}$. We define the weight C of P as the added weight of its vertices, i.e., $C = c_1 + \ldots + c_N$. Making the assumption that the vertices of P have been sorted along some axis --which can be done in $O(N \log N)$ time--, we prove that it is possible, in $O(N)$ time, to find two vertices a,b in P, such that the segment ab lies entirely inside the polygon P and partitions it into two polygons, each with a weight not exceeding $2C/3$. We also give a list of problems which can be solved efficiently with a divide-and-conquer strategy based on that result.

1. Introduction

It is a fact that fast algorithms often owe their efficiency to a handful of mathematical truths that bring to light unsuspected specificity -- or singularity -- about the problem under consideration. But besides recognizing their mere aptitude at serving computational purposes, one may wish to add a bit of classification into the nature of those subservient truths.

To ease the matter, let us take two examples: proving that a minimum convex decomposition of a polygon could always be found free of interior parts was a major step in deriving a polynomial algorithm for that problem [CH80]. Similarly, establishing the unimodality of the vertex-coordinate function of a convex polygon was instrumental in the setting of logarithmic intersection-algorithms for convex objects [CD80]. There is, however, a notable difference in methodology between these two cases. Whereas the first example presents us with a mathematical fact which seems inherently helpful for deriving *any* algorithm, the latter brings out a compelling flavor of binary search-like technique which strongly suggests the proper algorithmic treatment.

Another case which witnesses both phenomena is the well-known *near-neighbor problem*, in which all nearest neighbor pairs in a given set of points are to be computed. One of the most efficient algorithms for this problem relies on an elegant geometric construction known as the *Voronoi diagram* [SH77]. Although the existence of this diagram was undoubtedly the keystone in the elaboration of the algorithm, it fell short of even suggesting an effective method, since there was no immediate evidence that constructing the Voronoi diagram was to be any easier than computing the neighbor pairs directly. Another mathematical fact, i.e., the decomposability of the Voronoi construction, was indeed needed to fire the final blow and crack the problem. There we should observe that although both facts are geometric in nature, the latter is combinatorial in spirit, as it draws its motivation from the algorithmic, all-purpose, *divide-and-conquer* technique.

Lipton and Tarjan's planar separator theorem [LT77,LI77] is a notable example of a systematic technique for introducing a computational tool, i.e., *divide-and-conquer*, into a whole class of related problems, i.e., *planar graph problems*. Drawing its inspiration from this philosophy, this paper presents a theoretical result on polygon decomposition which can be applied to derive a number of efficient algorithms for geometric problems, in particular, problems of convex decompositions, triangulation, visibility, and internal distance.

2. The Polygon-cutting Theorem

Let P be a simple[1] polygon with vertices $v_1,...,v_N$ in clockwise order. Let OXY be an orthogonal system of

[1] A polygon is said to be simple iff only adjacent edges intersect.

reference. Wlog, we can always assume that no two vertices have the same X-coordinate. We define a partial, so-called *vertical*, order as follows:

> We say that "edge $e >$ edge f" iff their projections onto the X-axis overlap and, restricting ourselves to the overlapping area, e lies entirely above f (fig.1).

[FIGURE 1]

Figure 1: *The partial order among the edges of P.*

Throughout this paper, we will assume that along with a description of the boundary of P, provided by a doubly-linked list LP, we have available both a doubly-linked list LV of all the edges in topological (*vertical*) order, and a doubly-linked list LH of all the vertices sorted by X-coordinates [KN73]. The preprocessing involved in setting up these lists requires $O(N\log N)$ time and $O(N)$ space. The decomposition algorithm which we will describe later on runs in linear time, with this preprocessing in hand. Note that it is legitimate to separate both tasks, since in the applications which we will mention, the decomposition algorithm will be called recursively several times, while the preprocessing will be needed initially, once and for all.

The goal of this paper is to prove the following theorem:

> **Theorem 1:** *The Polygon-cutting Theorem.* Let P be a simple polygon with N vertices $v_1,...,v_N$, each assigned a weight c_i ($c_i=0,1$). Let $C(P)$ denote the total weight of P, defined as the sum $c_1+...+c_N$, and assume that $C(P)>2$. With the lists LP, LH, LV in hand, it is possible to find, in $O(N)$ time, a pair of vertices v_i, v_j such that the segment v_iv_j lies entirely inside the polygon P and partitions it into two simple polygons P_1, P_2 satisfying:
>
> $$C(P_1) \leq C(P_2) \leq 2C(P)/3$$

The weights of the vertices in P_1 and P_2 are the same as in P, except for v_i and v_j, for which we will assume that in both P_1 and P_2, these weights become 0. This assumption is made only for the sake of simplicity, and other conventions (e.g., keeping the same weights c_i,c_j in both P_1 and P_2) are indeed acceptable, if we are ready to add a term $+2$ to $2C(P)/3$ in the inequality of Theorem 1. To facilitate our task, we will first prove the theorem with slightly relaxed requirements.

2.1. An existence theorem

To begin with, we will prove the existence, not of two *vertices*, but of two *points* on the boundary of P, satisfying the inequalities of Theorem 1.

> **Theorem 2:** Same assumptions as Theorem 1. There exists a pair of points A,B on the boundary of P, such that the segment AB is parallel to the Y-axis, lies entirely in the polygon P, and partitions it into two simple polygons P_1, P_2 satisfying:
>
> $$C(P_1) \leq C(P_2) \leq 2C(P)/3$$

If A and B are not vertices of P, for consistency, we assign them a 0-weight. The underlying notion of "equal size" decomposition expressed in the polygon-cutting theorem motivates the introduction of a distance function $d(A,B)$, defined between two points A,B on the boundary of P as the minimum path weight between A and B. More precisely, let $v_i v_{i+1}$ (resp. $v_j v_{j+1}$) be the edge of P containing A (resp. B). If A (resp. B) is a vertex of P, it is assumed to be v_i (resp. v_j). We introduce the function h, defined as follows[2]:

$$h(A,B) = c_{j+1} + c_{j+2} + \dots + c_i$$

from which we can define $d(A,B)$:

$$d(A,B) = \min [\, h(A,B)\, ,\, C(P) - h(A,B)\,]$$

Starting at the edge $v_1 v_2$, we label each edge of P recursively, as follows:

$$\lambda(v_1 v_2) = c_1$$

$$\lambda(v_i v_{i+1}) = \lambda(v_{i-1} v_i) + c_i$$

Note that this labeling gives us an alternate way of defining the distance between two boundary points A,B:

$$d(A,B) = \min [\, |\lambda(v_i v_{i+1}) - \lambda(v_j v_{j+1})|\, ,\, C(P) - |\lambda(v_i v_{i+1}) - \lambda(v_j v_{j+1})|\,]$$

We are now in a position first to prove the existence of the segment AB, as defined in Theorem 2, then to describe an efficient method for finding it. As we will see, the first step is not superfluous; it is an essential ingredient in ensuring the correctness of the algorithm.

Choose the leftmost point of P as the starting point of the left-to-right sweep of a vertical segment $S=AB$ (A below B). S will always stretch vertically so as to keep its endpoints A,B in permanent contact with the boundary of P. It will thus be able to move continuously to the right, until it must either expand (fig.2.1) or split (fig.2.2). At any time during the course of the motion, S will be assigned a value $\Delta = h(A,B)$ to indicate how close it is to being the desired segment. We observe that initially, $\Delta = C(P)$, and that as long as S moves continuously, Δ decreases monotonously by unit steps. When either situation depicted in Figure 2 arises, we can always write

$$\Delta = h(A,B) = h(A,C) + h(C,B) \tag{1}$$

from which we can derive a decision procedure for redefining S.

> Starting from the leftmost vertex of P, move S from left to right, stretching or shrinking this segment so that it entirely lies in P, and its endpoints always lie on the boundary of P. As long as the motion of S is continuous, check whether $\Delta > 2C(P)/3$, in which case continue, else stop. When falling in either case of fig.2, reset S to AC if $h(A,C) > h(C,B)$, or to CB otherwise. Note that if S is reset to CB in the case of fig.2.1, the motion must reverse its direction.

[2] All arithmetic on indices is done [mod N].

Relation (1) shows that every discontinuity causes Δ to decrease by at most half, while otherwise Δ decreases at most by unit steps, since we have assumed that no two vertices may lie on the same vertical line. Since Δ eventually vanishes and $C(P)>2$, it must take on some value in the interval $[C(P)/3, 2C(P)/3]$, at which point the procedure will stop and return the desired segment AB. This completes the proof of Theorem 2. □

[FIGURE 2]

Figure 2: *Proving the existence of a separator S.*

2.2. A relaxed version of the polygon-cutting theorem

Unfortunately, Theorem 2 falls short of providing an efficient algorithm for computing AB. We can, however, graft to it a binary search-like structure to improve the performance of a naive implementation. The purpose of this section is thus to prove the following result:

Theorem 3: Same assumptions as Theorem 1. It is possible, in $O(N)$ time, to find a pair of points A,B on the boundary of P, such that the segment AB lies entirely in the polygon P, and partitions it into two simple polygons P_1, P_2 satisfying:

$$C(P_1) \leq C(P_2) \leq 2C(P)/3$$

The algorithm which we will describe in order to prove Theorem 3 is recursive; it requires $O(N)$ time to cut down the size of the problem by half, therefore its overall performance is linear.

1. Recall that initially, we have available both the vertical topological order (LV) of the edges of P and the horizontal order (LH) of its vertices. The latter list permits us to determine a median vertical line L in $O(N)$ time, i.e., a line parallel to the Y-axis that separates the vertices of P into two sets of size $\lceil C(P)/2 \rceil$ and $\lfloor C(P)/2 \rfloor$ respectively. Once again, this is always possible since no two vertices may have the same X-coordinate (fig.3).

2. Compute all the intersection points of L with the boundary of P, and sort them by Y-coordinates, doing all of this in $O(N)$ time with the list LV. Next, form the intersection segments $A_1B_1, \ldots, A_kB_k$ in ascending order (fig.3).

3. If M is a point lying on an edge v_iv_{i+1} of P, distinct from its endpoints, by extension, we define $\lambda(M)$ as $\lambda(v_iv_{i+1})$. Check whether any of the segments A_iB_i satisfies the relation

$$C(P)/3 \leq |\lambda(A_i) - \lambda(B_i)| \leq 2C(P)/3, \text{ for } i=1,\ldots,k.$$

   If yes, the corresponding segment A_iB_i can be chosen as AB, and the procedure can stop. Otherwise, go to 4.

4. Transform the list LP as follows. For every A_iB_i ($i=1,\ldots,k$) in turn, if $|\lambda(A_i) - \lambda(B_i)| < C(P)/3$, start at A_i or B_i (say, A_i) whichever has the smaller label $\lambda(A_i)$ or $\lambda(B_i)$, and proceed to traverse LP in clockwise order, until B_i is reached, deleting all the vertices and links visited. Finally, close the chain by inserting A_i and B_i into LP, setting a double link between the two points. If $|\lambda(A_i) - \lambda(B_i)| > 2C(P)/3$, perform the same sequence of operations, now starting at the point A_i

or B_i with the larger label $\lambda(A_i)$ or $\lambda(B_i)$.

5. Using the new description LP of the pruned polygon, update the lists LV and LH by deleting the edges and vertices no longer existing in LP.

6. Apply the algorithm recursively until termination.

[FIGURE 3]

Figure 3: *The recursive algorithm for computing the separator S.*

A few words of explanation may be needed to justify the algorithm. To be begin with, observe that once the intersection points are available in vertical order, setting up the list of segments A_iB_i is straightforward, since the list of consecutive segments formed by the intersection points alternates inside and outside P. Also, it is easy to verify that the deletion of edges is accompanied by the setting of links which, in particular, ensures that no edge is visited more than twice. It may be the case that handling a segment A_iB_i may also handle several others at the same time. For example, see the effect in fig.4 of handling A_3B_3 before A_4B_4, or vice-versa. In this figure, the dashed area is to be removed from P. Finally, we note that deletions are indeed permissible since any part of P cut off by A_iB_i weighs at most $d(A_i,B_i) < C(P)/3$. Thus any vertical segment AB in that part has a distance $d(A,B) < C(P)/3$, and may therefore be dismissed. Since, when iterating down the recursion level, all the segments A_iB_i are being cut off and their attached part removed, there remains only one connected part which lies entirely on one side of the median line, and to which, therefore, P can be reset before calling the algorithm recursively. Of course, the labeling λ remains unchanged. The correctness of the algorithm follows from Lemma 2 and from the fact that we remove only non-candidates. The reason for choosing a median line to operate the cut-offs is now apparent. Since all the vertices on one side of the line are bound to disappear, the recursion will be invoked on a problem of size reduced by at least half. This completes the proof of Theorem 3. □

[FIGURE 4]

Figure 4: *Pruning the polygon P.*

2.3. Completing the proof of the polygon-cutting theorem

We may now turn our attention back to Theorem 1. Let v_iv_{i+1} and v_jv_{j+1} be the edges of P that contain the points A and B of Theorem 3, respectively. To prove the desired result, one may be tempted to slide A and B towards the endpoints of v_iv_{i+1} and v_jv_{j+1} respectively, until one of the configurations v_iv_j, v_iv_{j+1}, $v_{i+1}v_j$, or $v_{i+1}v_{j+1}$ has been reached. Unfortunately, obstacles may prevent this from ever happening (fig.5), so our next step will be to take a closer look at these possible obstacles.

Since the quadrilateral $v_iv_{i+1}v_jv_{j+1}v_i$ contains the segment AB, it is a simple polygon, and AB partitions it into two polygons $Q_1 = Av_{i+1}v_jBA$ and $Q_2 = ABv_{j+1}v_iA$. As a corollary of the Jordan Curve Theorem [HS55][3], it is easy to show that the only obstacles encountered in Q_1 (resp. Q_2) are vertices in the set $\{v_{i+1}, v_{i+2}, ..., v_j\}$ (resp. $\{v_{j+1}, v_{j+2}, ..., v_i\}$). Moreover, the segment $S = AB$ can encounter only vertices on the convex hull H_1 (resp. H_2) of the vertices of P lying inside Q_1 (resp. Q_2) (fig.5), as is shown in following result.

Lemma 4: The segment AB intersects any edge of P (outside of A or B) if and only if it intersects the boundary of either H_1 or H_2.

Proof: Since H_1 (resp. H_2) lies entirely inside Q_1 (resp. Q_2), AB intersects any edge of P lying in Q iff the infinite line passing through AB does, hence iff AB lies outside of H_1 and H_2. □

The next task is to compute the convex hulls H_1 and H_2. We only give the details of the algorithm for H_1, the other case being strictly similar. Since we cannot afford to use a standard $O(N\log N)$ algorithm to simply compute the convex hull of the vertices of P in Q_1, we must exploit the fact that these vertices lie on a polygonal line in order to achieve linear time.

[FIGURE 5]

Figure 5: *Defining the domain of safety for AB.*

To begin with, let us give an informal description of the algorithm. The goal is, in a first stage, to produce a polygon K_1 which lies entirely in Q_1, and whose convex hull is exactly H_1. K_1 consists essentially of polygonal chains[4] L_i made of consecutive edges from the set:

$$L = \{ v_{i+1}v_{i+2}, v_{i+2}v_{i+3}, ..., v_{j-1}v_j \}$$

Each chain has the property that it lies in Q_1 and intersects $v_{i+1}v_j$ in two points, U_k and V_k. Moreover, no two segments U_kV_k overlap (fig.6.1). To compute these chains, we must distinguish between two types of edges in L. An edge v_kv_{k+1} is said to be *entering* (resp. *exiting*) if it intersects $v_{i+1}v_j$, and v_{k+1} lies inside (resp. outside) Q_1. The algorithm proceeds as follows:

Wlog, assume that there is at least one vertex v_{i+2} from v_{i+1} to v_j in clockwise order. Traverse L from $v_{i+1}v_{i+2}$ to $v_{j-1}v_j$, stopping at entering and exiting edges and taking the following actions. If the current edge is *entering*, it may be the endpoint U_k of a new chain L_k. To decide of this, look at the next *exiting* edge in L; if it intersects $v_{i+1}v_j$ in a point V_k on the segment U_kv_j, we have indeed a new chain L_k from U_k to V_k. Otherwise, not only don't we have a new chain L_k, but the chain just visited may enclose previously computed chains, which must then be deleted. For that purpose, we use a stack to hold the pairs (U_kV_k, L_k), so that

[3]The *Jordan Curve Theorem* states that a closed curve in the plane partitions the plane into two connected regions: the *inside* and the *outside*.

[4]We define a polygonal chain as any connected sequence of segments such that only consecutive segments may intersect.

deletions may be done efficiently. The algorithm is straightforward, so we may omit the details.

Initially, the stack is empty, and the current edge e is the first *entering* edge of L.

```
begin
Let $f$ be the next exiting edge in $L$ following $e$, and let $U, V$ be
respectively the intersections of $e$ and $f$ with $v_{i+1}v_j$.

if $V$ lies on $Uv_j$
then
    Let $L_u$ be the chain between $e$ and $f$ in $L$.
    Push $(UV, L_u)$ onto stack.
    Go to next entering edge $e'$ in $L$, whose intersection
    with $v_{i+1}v_j$ lies on $Vv_{j+1}$, then iterate.

else
    Go to next entering edge $e'$ whose intersection $U'$
    with $v_{i+1}v_j$ lies on $v_{i+1}V$.
    Pop all pairs $(U_k V_k)$ off the stack as long as $V_k$ lies on $U'v_j$.
    Iterate.
end
```

[FIGURE 6]

Figure 6: *Computing the polygon K_1*

To prove the correctness of the algorithm, we begin by observing that the intersection of L with Q_1 consists of chains whose endpoints lie on $v_{i+1}v_j$, and that K_1 consists of exactly all the *maximal* chains. A chain is said to be *maximal* if it does not lie in the enclosure of any other chain with $v_{i+1}v_j$. A maximal chain is also characterized by the fact that the segment formed by its endpoints does not lie inside any other such segments.

From the Jordan Curve Theorem [HS55], we derive the fact that a maximal chain from U to V, in clockwise order, has its endpoint V lying above U (i.e., on the segment Uv_j). In consequence, only the chains which *move* towards v_j are candidates for being part of K_1, which justifies the selection criterion of the algorithm. On the other hand, we can also show that a non-maximal chain which moves towards v_j is necessarily enclosed by another chain moving away from v_j. This explains the deletion rule. Finally, the last observation to make is that a chain from U to V which moves away from v_j (i.e., U lies on Vv_j) must be enclosed by a subsequent maximal chain, therefore since maximal chains are computed "towards" v_j, we may skip directly to the next *entering* edge in L that intersects $v_{i+1}v_j$ at a point U' below V (i.e., U' lies on $v_{i+1}V$) - see illustration of the various cases in fig.6.1. This completes the proof of correctness, and shows that all the chains L_k may be computed in sorted order along the segment $v_{i+1}v_j$, all these computations requiring $O(N)$ time.

The final step in computing K_1 is to connect all the chains L_k together in the order in which they appear in the stack. To do so, we borrow segments from $v_{i+1}v_j$, as shown in fig.6.2. We may now apply any standard linear convex hull algorithm for simple polygons [LE80,SH77] in order to obtain H_1 in $O(N)$ time.

Assuming that both H_1 and H_2 are available, we are now in a position to give an algorithm for finding the two vertices of Theorem 1. The idea is to connect the vertices of H_1 with those of H_2, so as to triangulate the polygon H^* defined as the area between H_1 and H_2 containing AB. We claim that at least one of the edges of the triangulation will provide the desired pair of vertices, with the property of Theorem 1. The algorithm proceeds as follows:

Let $h_1,...,h_p$ and $k_1,...,k_q$ be the vertices of H_1 and H_2, respectively, as we traverse them from $v_{i+1}v_i$ to v_jv_{j+1}, i.e., $h_1 = v_{i+1}$, $h_p = v_j$, $k_1 = v_i$, $k_q = v_{j+1}$. We maintain two pointers, h on H_1 and k on H_2, moving them from h_1 to h_p and k_1 to k_q, respectively, and computing the triangulation on the fly. Note that, at all times, the segment hk intersects H_1 and H_2 only at its endpoints (fig.7.1).

[FIGURE 7]

Figure 7: *Triangulating the domain of safety.*

The simplest way of describing the algorithm is recursively. Initially, hk is $v_{i+1}v_i$; the algorithm terminates with $hk = v_jv_{j+1}$.

Let $hk = h_tk_u$, and consider the quadrilateral[5] $hh_{t+1}k_{u+1}kh$. We will show in Lemma 5 that at least one of its diagonals, hk_{u+1} or kh_{t+1}, connects H_1 and H_2 without intersecting these polygons outside of its endpoints, i.e., lies entirely in H^*. Moreover, this diagonal can be found in constant time. We may then determine that diagonal, add it to the triangulation, set hk to it, and iterate.

The algorithm clearly runs in linear time. Also, the assurance that it effectively produces a triangulation of H^* comes from the fact that it keeps only edges which lie entirely in H^*, and that the pointers h and k pass a vertex only after a diagonal has been assigned to it. Thus there only remains to prove the following lemma:

Lemma 5: If the segment h_tk_u connects H_1 and H_2 and lies entirely inside H^*, so does one of the diagonals h_tk_{u+1} or k_uh_{t+1}. Moreover, this diagonal can be found in constant time.

Proof: Consider the line passing through hk, oriented from h to k. If a point lies to the right (resp. left) of this line, we will say that it lies *below* (resp. *above*) hk. Since H_1 and H_2 are convex, at

[5]For consistency, we define h_{p+1} and k_{q+1} as v_{j+1} and v_j, respectively.

least one of the vertices v_j or v_{j+1} lies above hk, therefore it is impossible that both h_{t+1} and k_{u+1} lie below hk. Indeed this would involve the existence of at least three intersection points between a line and a convex boundary, leading to a contradiction. If only one segment, say h_{t+1} lies above hk, it can be easily determined in constant time, and since in that case, all of H_2 lies below hk, the diagonal $h_{t+1}k$ does not intersect H_2 (nor H_1 either) outside of its endpoints, and may thus be chosen as the next segment of the triangulation (fig.7.2). If, on the other hand, both h_{t+1} and k_{u+1} lie above hk, the quadrilateral $hh_{t+1}k_{u+1}k$ is a simple polygon, therefore it contains at least one of its diagonals entirely (fig.7.3), and this diagonal can be found in constant time. Note that, because of its convexity, H_1 (resp. H_2) lies totally on one side of the line passing through hh_{t+1} (resp. kk_{u+1}), therefore the whole quadrilateral, hence the chosen diagonal, lies inside the polygon H^*, which completes the proof. □

The purpose of triangulating the polygon H^* will become apparent with the following result.

Lemma 6: There exists an edge uv in the triangulation of H^* which satisfies the relation: $C(P)/3 \leq h(u,v) \leq 2C(P)/3$.

Proof: From Theorem 3, we know that AB partitions P into two polygons with weights between $C(P)/3$ and $2C(P)/3$, which gives the relations

$$C(P)/3 \leq \min(\, h(A,B), h(B,A)\,) \leq \max(\, h(A,B), h(B,A)\,) \leq 2C(P)/3$$

As a result, any pair of vertices a,b on H_1 (resp. H_2), with b following (resp. preceding) a in the list $\{h_1,\dots,h_p\}$ (resp. $\{k_1,\dots,k_q\}$) satisfies the relation:

$$h(b,a) \leq 2C(P)/3 \qquad (1)$$

On the other hand, each triangle abc of the triangulation has one side ab on the boundary of either H_1 or H_2, with the two others ac, bc constructed by the triangulation algorithm. Wlog, let ac be the segment of the triangle constructed first (i.e., ac lies below bc). We always have

$$h(c,a) = h(b,a) + h(c,b) \qquad (2)$$

Now we can show that a simple upward scan through the faces of the triangulation, i.e., starting at the triangle adjacent to v_iv_{i+1} and ending at the triangle adjacent to v_jv_{j+1}, will inevitably lead to the desired segment of Theorem 1. To see that, we may obviously assume that none of the edges ab of H_1 or H_2 satisfies the relations:

$$C(P)/3 \leq h(b,a) \leq 2C(P)/3\,,$$

otherwise, we have achieved our goal.

In that case, Relation (1) shows that for any triangle abc visited, the edge on the boundary of H^*, say ab, satisfies the stronger inequality

$$h(b,a) < C(P)/3\,,$$

which, combined with Relation (2), leads to

$$h(c,b) > h(c,a) - C(P)/3.$$

Since $h(v_i,v_{i+1}) = C(P) - c_{i+1}$ and $h(v_{j+1},v_j) = c_{j+1}$, it follows that if $a_1b_1, a_2b_2, \dots$ is the sequence of interior edges visited in the traversal of the triangulation, with the points a_m (resp. b_m) on H_1 (resp. H_2), the sequence $h(a_1,b_1), h(a_2,b_2), \dots$ is monotonously decreasing from $C(P) - c_{i+1}$ to c_{j+1} by jumps of at most $C(P)/3$. In consequence, it must take on at least one value in the interval

$[C(P)/3, 2C(P)/3]$, which can be chosen as the pair u,v. □

The proof of Theorem 1 is now complete. Computing H_1 and H_2 definitely constitutes the most difficult part of the algorithm to implement. We may observe, however, that this overhead will often be unnecessary since, in practice, it may be seldom the case that the segment AB of Theorem 3 is prevented from sliding towards the endpoints of its supporting edges.

3. Applications to polygon decomposition problems

It is intuitive that the polygon-cutting theorem should lead to efficient methods for partitioning a polygon into convex pieces. We will examine two instances of this problem: in one, what is desired is a partition of the polygon into a *small* number of convex pieces, while in the other, only a triangulation of the polygon is sought, without consideration of optimality[6].

3.1. Convex decompositions

Given a simple, non-convex polygon P, find a minimum number of convex, pairwise disjoint polygons, whose union is P.

This problem has been well-studied [CH80,CD79,FS81,GJ78,SC78,SV80,TO80], and several algorithms have been discovered for producing minimal or near-minimal decompositions. Here we consider only decompositions which do not introduce new points, i.e., all the vertices of the polygons are vertices of P.

In connection with the previous section, we will assign to each vertex v_i of P a weight $c_i=1$ if its adjacent edges form a reflex angle (in which case, v_i is called a *notch*), and a weight $c_i=0$ otherwise. Thus we can apply the polygon-cutting theorem (Theorem 1) iteratively to decompose P into smaller polygons. Note that since, with our convention, the endpoints of the splitting segment lose their weights, the number of notches C_1,C_2 of the two parts is each bounded by $2C/3+2$, where C is the number of notches in the original polygon. As a result, we must stop the iteration when the algorithm ceases to reduce the number of notches, i.e., when all the parts have a number of notches satisfying: $C \leq 2C/3+2$, i.e., $C \leq 6$. Finally, to resolve the remaining reflex angles, we consider each of them in turn, proceeding as follows:

Let Q be the polygon (with at most 6 reflex angles), and v be the notch exhibiting the reflex angle to be resolved. Let L (resp. R) denote the *ray* (i.e., semi-infinite line) starting at v in the direction of the edge ending (resp. starting) at v (fig.8). Compute the intersection(s) of L (resp. R)

[6]Applying a quality criterion to the triangulation of a polygon is common practice in numerical analysis, where an area distribution or a shape function is often to be optimized. There are many good reasons, however, for making the availability of *any* triangulation desirable. See [CH82], for example.

with the boundary of Q, and keep only the intersection point A (resp. B) closest to v. If A and B lie on different edges, there exists at least one vertex on the part of the boundary of P between A and B which can be joined to v, so as to resolve the reflex angle at v (fig.8.1). For example, we can choose the vertex w between A and B that minimizes the angle (vB,vw), while keeping it positive. If, on the other hand, A and B lie on the same edge v_Av_B (fig.8.2), we compute the vertex a of the list $(v,...,v_A)$, given in clockwise order, which lies in the triangle vAv_A and minimizes the angle (vA,va). Similarly, we compute the vertex b which lies in vBv_B and minimizes the angle (vb,vB). Both of these operations can be executed in linear time. Note that minimizing the angles ensures that both va and vb lie entirely in Q. It is also easy to show that the combination of these two segments resolves the reflex angle at v by splitting Q into 3 polygons (note that in most cases, a and b will be v_A and v_B, respectively).

[FIGURE 8]

Figure 8: *Completing the convex decomposition of P.*

The decomposition algorithm thus consists of a recursive "cutting" phase which relies on the algorithm given for the polygon-cutting theorem. The recursion stops when the polygon currently examined has fewer than 7 notches, at which point the procedure just described is called upon to finish off the decomposition. We observe that if either the vertex v_i or v_j, say v_i, in Theorem 1 is a notch, i.e., $c_i = 1$, v_i appears in both of the resulting polygons P_1 and P_2, but is a notch for at most one of them. Therefore if, by extension, we let $C(N)$ denote the weight of P and $C(N_1)$ (resp. $C(N_2)$ be the weight of P_1 (resp. P_2), we can write:

$$C(N) \leq C(N_1) + C(N_2) \qquad (1)$$

$$C(N_1) \leq C(N_2) \leq 2C(N)/3 + 2 \qquad (2)$$

Note that $C(N_1)$ and $C(N_2)$ are actual weights, i.e., they count exactly the number of notches in P_1 and P_2, as opposed to the weights of P_1 and P_2 as defined in Theorem 1, which did not account for the endpoints of the splitting segment. It is easy to see that, in the worst case, we will end up with $C(N)/6$ polygons with, each, 6 notches, and the final phase will use 2 cuts for the resolution of each reflex angle. This will result in $13C(N)/6$ convex pieces, which is to be compared with the minimum number of convex pieces, shown to be always greater than or equal to $C(N)/2+1$ in [CH80].

Next we turn to the complexity of our decomposition algorithm. While it clearly needs $O(N)$ space, evaluating its run-time $T(N)$ calls for further investigation. If we neglect the preprocessing phase for the time being, we have the relations

$$T(N) = T(N_1) + T(N_2) + O(N), \text{ if } C(N) > 6 \qquad (3)$$

$$N_1 + N_2 = N + 2 \qquad (4)$$

$$T(N) = O(N), \text{ if } C(N) \leq 6 \qquad (5)$$

Consider the recursion tree, and label each node with the number p of vertices in the corresponding polygon. At the leaves, we have $C(P) \leq 6$, while for their ancestors, $C(P) > 6$. Relations (3) and (5) show that up to within a constant factor, $T(N)$ is equal to the sum of the labels in the tree, while from Relation (4), it follows that if $L(i)$ counts the sum of all labels at level i, we have $L(0) = N$ and

$$L(i) \leq L(i\text{-}1) + 2^i,$$

which gives $L(i) \leq N + 2^{i+1}$. If k is the height of the tree, we easily find that

$$L(0) + ... + L(k\text{-}1) \leq Nk + 2^{k+1},$$

and since, from (2), we have $k = O(\log C(N)) = O(\log N)$, including the $O(N\log N)$ preprocessing in the running time, we can conclude:

Theorem 7: In $O(N\log N)$ time and with $O(N)$ space, it is possible to decompose a simple N-gon P into fewer than 4.333...×OPT convex pieces, without introducing new vertices, where OPT is the minimum number of convex pieces necessary to partition P.

3.2. Triangulation

When all the pieces of a convex decomposition are triangles and no new vertices are introduced, the decomposition is called a *triangulation* of the polygon. An $O(N\log N)$ algorithm for computing a triangulation of a simple polygon has been given in [GJ78]. The method requires the somewhat cumbersome use of AVL-trees or similar logarithmic search trees. This can be avoided by using a strategy based on the polygon-cutting theorem.

We may choose to assign a weight $=1$ to each vertex of P and apply the polygon-cutting theorem recursively, until the polygon under consideration has fewer than 7 vertices, at which point it is straightforward to complete the triangulation. We omit the details. An alternative consists of computing a convex decomposition of P as described in the previous section, then triangulate each convex polygon. To do so, pick any vertex of the polygon and join it to every other.

In both cases, a triangulation of P can be explicitly computed in $O(N\log N)$ time, which matches the performance of [GJ78]. We recall that it is yet unknown whether $kN\log N$ is optimal for this problem.

Theorem 8: Using the polygon-cutting theorem, it is possible to triangulate a simple N-gon in $O(N\log N)$ time and $O(N)$ space.

It is shown in [CH82] how the additional information provided by an arbitrary triangulation of a simple polygon is sufficient to derive optimal algorithms for a number of geometric problems. Using the previous result to compute a triangulation of P, it is then possible to determine the area visible from any point inside P in linear time. Also, *internal path* problems, i.e., problems involving the computation of the shortest path between two points inside P can then be solved optimally [CH82].

4. Conclusions and future research

The decomposition principle in geometry expresses the feasibility of local treatments for the solution of general problems on arbitrary figures. The polygon-cutting theorem presented in this paper asserts the applicability of this principle in the case of simple polygons, and by doing so, leads to efficient, simple divide-and-conquer methods for solving a variety of geometric problems.

The merit of this approach lies primarily in the versatility of its applications as well as in the increased efficiency which it affords. The most immediate open question is whether sorting the vertices in preprocessing is indeed required. If not, the algorithm would automatically become linear. In this paper, we have deliberately chosen simplicity and practicality over generality by restricting the weights attached to the vertices to take on the values 0,1. This was motivated by the fact that this restriction still allowed us to apply divide-and-conquer to an arbitrarily chosen subset of vertices, while adding simplicity to the exposition. The reader will observe, however, that it is straightforward to extend the theorem to a more general weight function.

The applications mentioned in this work are only a few examples among a number of other problems which can benefit from the polygon-cutting theorem. Enlarging the list of applications given here is certainly a worthwhile endeavor.

REFERENCES

[CH80] Chazelle, B.M.

Computational geometry and convexity, PhD thesis, Yale University, 1980. Also available as CMU Tech. Rept. CMU-CS-80-150, July 1980.

[CH82] Chazelle, B.M.

The power of triangulation: applications to problems of visibility and internal distance, CMU- Tech. Report, Carnegie-Mellon University, March 1982.

[CD79] Chazelle, B.M., Dobkin, D.P.

Decomposing a polygon into its convex parts, Proc. 11th SIGACT Symp., Atlanta, 1979, pp. 38-48.

[CD80] Chazelle, B.M., Dobkin, D.P.

Detection is easier than computation, Proc. 12th SIGACT Symp., Los Angeles, 1980.

[FS81] Ferrari, L., Sankar, P.V., and Sklansky, J.

Minimal rectangular partitions of digitized blobs, Proc. 5th International Conference on Pattern Recognition, Miami Beach, Dec. 1981, pp. 1040-1043.

[GJ78] Garey, M.R., Johnson, D.S., Preparata, F.P., and Tarjan, R.E.

Triangulating a simple polygon, Info. Proc. Lett., Vol. 7(4), June 1978, pp. 175-179.

[HS55] Hall, D.W., Spencer, G.

Elementary topology, Wiley, New York, 1955.

[KN73] Knuth, D.E.

The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, Mass., 1973.

[LE80] Lee, D.T.

On finding the convex hull of a simple polygon, Tech. Report # 80-03-FC-01, Dept. of EE&CS, Northwestern University, 1980.

[LP77] Lee, D.T., Preparata, F.P.

Location of a point in a planar subdivision and its applications, SIAM J. Comput. 6(1977), pp. 594-606.

[LT77] Lipton, R.J., Tarjan, R.E.

A separator theorem for planar graphs, Waterloo Conference on Theoretical Computer Science, August 1977, pp. 1-10.

[LI77] Lipton, R.J., Tarjan, R.E.

Applications of a planar separator theorem, Proc. 18th IEEE FOCS Symp., Providence, RI, 1977, pp. 162-170.

[MP78] Muller, D.E., Preparata, F.P.

Finding the intersection of two convex polyhedra, Theoret. Comput. Sci., 7 (1978), pp. 217-236.

[SC78] Schachter, B.

Decomposition of polygons into convex sets, IEEE Trans. on Computers, Vol. C-27, 1978, pp. 1078-1082.

[SV80] Schoone, A.A., van Leeuwen, J.

Triangulating a star-shaped polygon, Tech. Rept. RUV-CS-80-3, University of Utrecht, April, 1980.

[SH77] Shamos, M.I.

Computational geometry, PhD thesis, Yale University, 1977.

[TO80] Toussaint, G.T.

Decomposing a simple polygon with the relative neighborhood graph, Proceedings of the Allerton Conference, Urbana, Illinois, October, 1980.

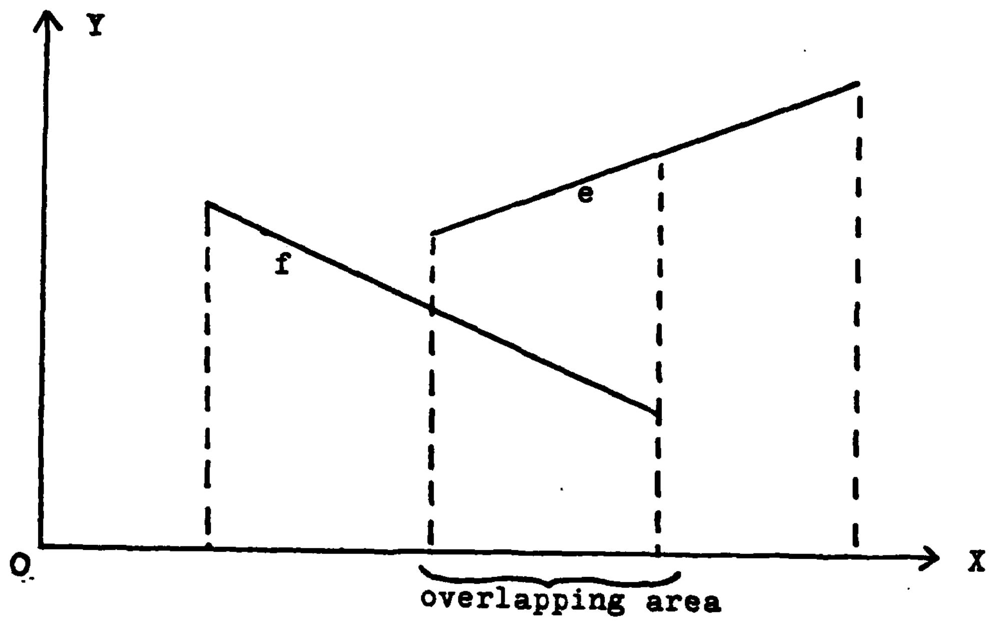
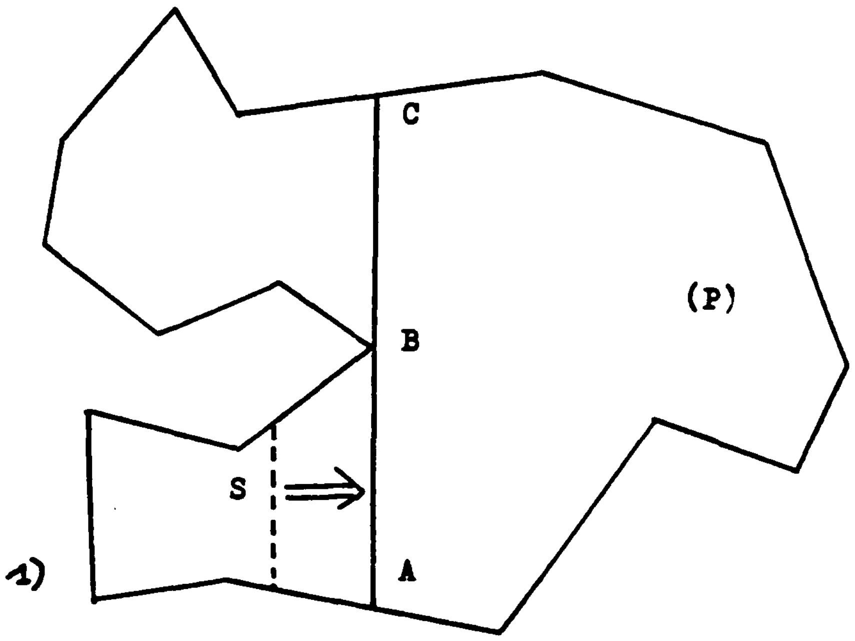


Figure 1



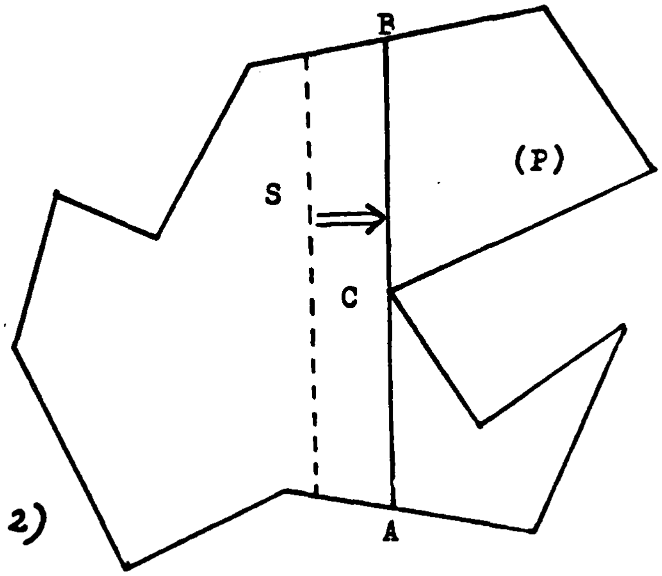
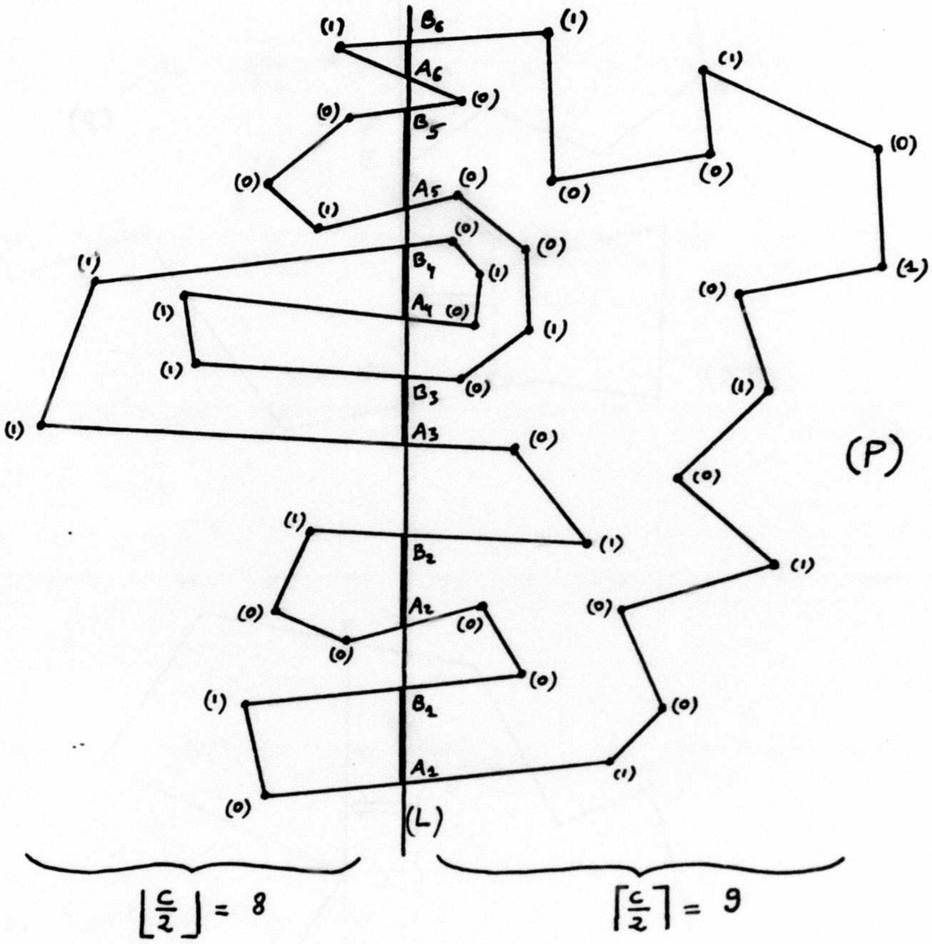


Figure 2



B_6
A_6
B_5
A_5
B_4
A_4
B_3
A_3
B_2
A_2
B_1
A_1
(L)
(P)
$\left\lfloor \frac{c}{2} \right\rfloor = 8$
$\left\lceil \frac{c}{2} \right\rceil = 9$

<u>Figure 3</u>

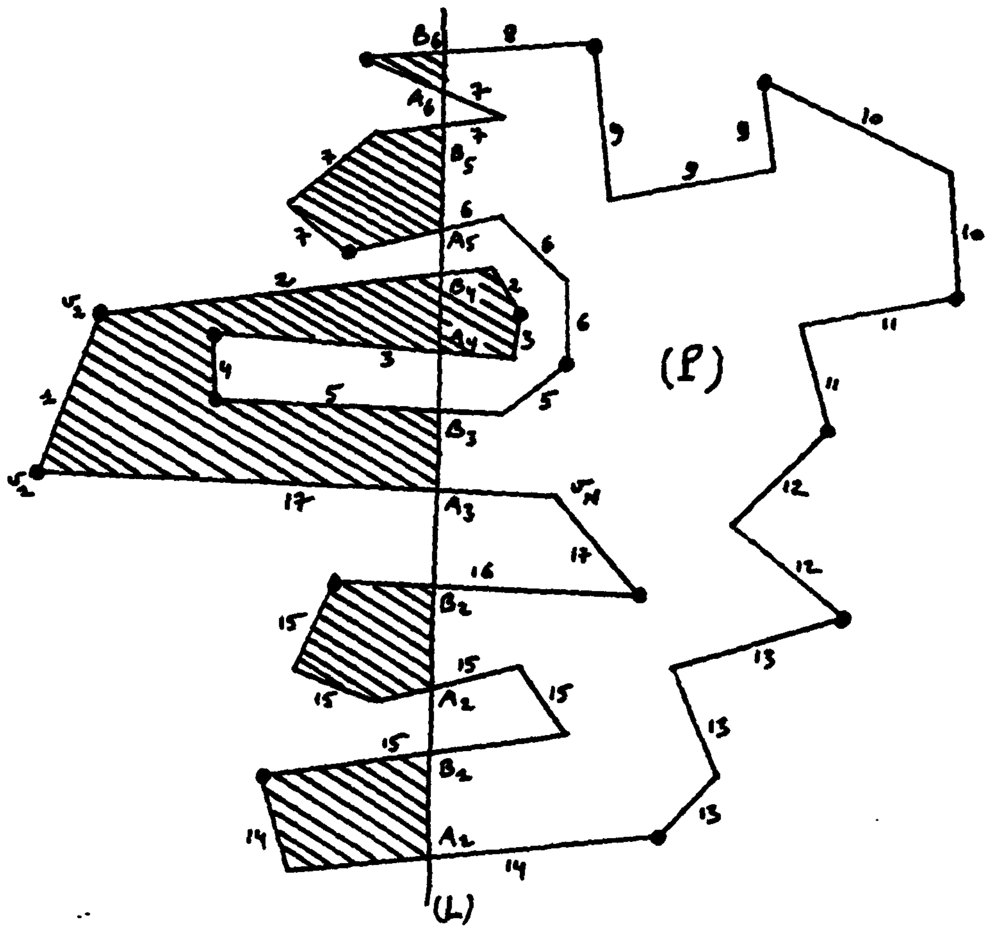


Figure 4

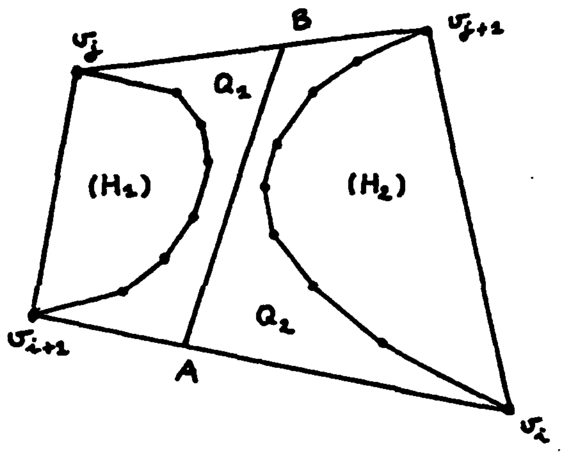


Figure 5

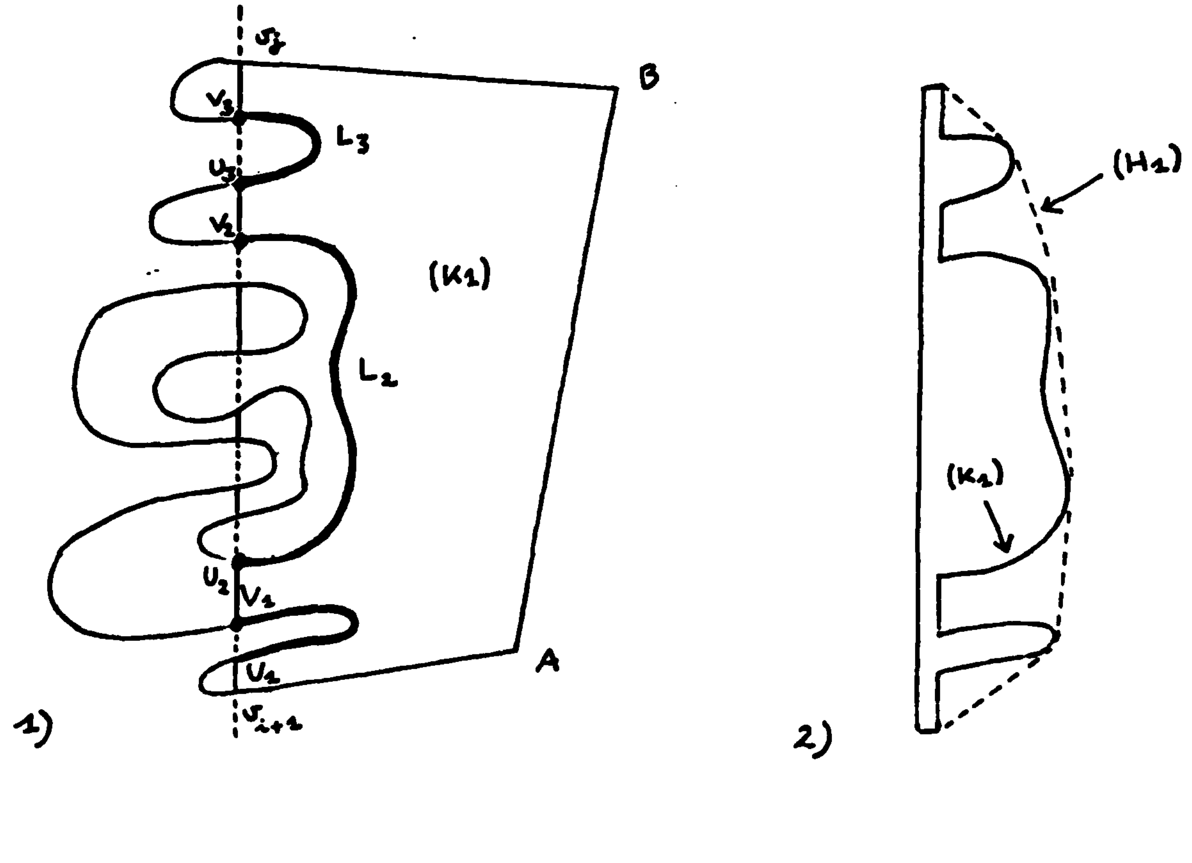


Figure 6

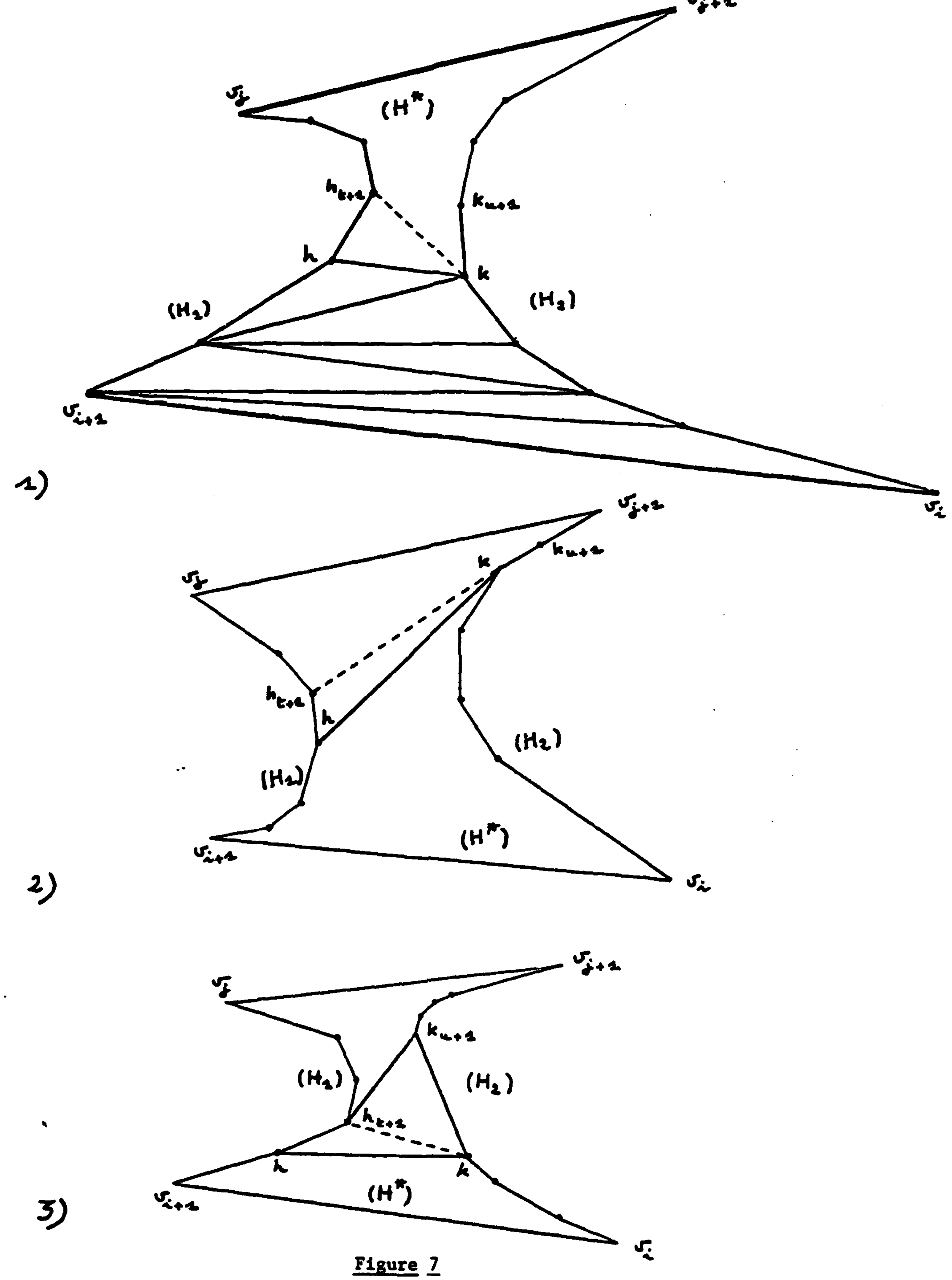
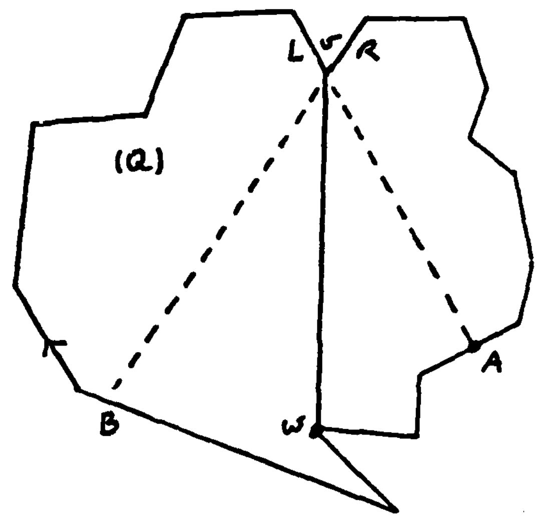
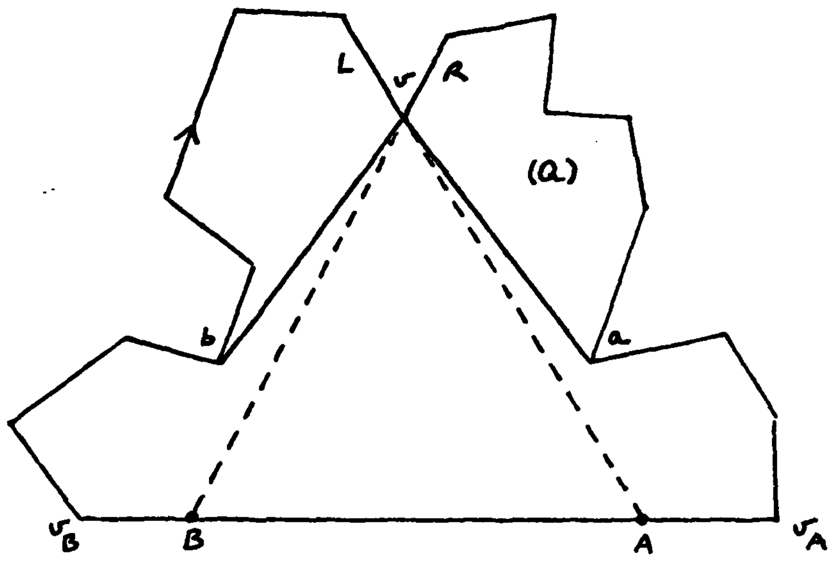

σ_{j+1}
σ_j
(H^*)
h_{t+1}
k_{u+1}
h
k
(H_1)
(H_2)
σ_{i+1}
σ_i
1)
σ_{j+1}
k
k_{u+1}
σ_j
h_{t+1}
h
(H_2)
(H_1)
(H^*)
σ_{i+1}
2)
σ_i
σ_{j+1}
σ_j
k_{u+1}
(H_1)
(H_2)
h_{t+1}
h
k
(H^*)
σ_{i+1}
3)
σ_i

Figure 7



1)



2)

Figure 8