A Fast Planar Partition Algorithm, II

Ketan Mulmuley
The University of Chicago

1 Introduction

In [Mul1] we gave a randomized, optimal, and efficient algorithm to find the partition of a plane induced by a set of linear segements. Optimal algorithms for the same problem were also independently given in [Ch],[Cl]. In this paper we extend the optimal planar partition algorithm of [Mul1] to many other related problems.

First we examine what happens if the segments in question are algebraic instead, but of a bounded degree. We shall approach this problem in two ways. The first approach is purely algebraic. Following this approach, we shall give an optimal, randomized $O(m + n \log n)$ algorithm to find the planar partition induced by a set of algebraic segments of bounded degree, where n is the number of segments and m is the number of intersections. The second approach to the problem is based on linear approximations. Here we approximate every algebraic segment by a chain of linear segments. The problem now is to find the planar partition induced by a given set of linear chains of a bounded degree, where the degree of a chain is defined to be the maximum number of intersections between the chain and a straight line. Note that we are not making any assumptions about the size of a chain, i.e. the number of linear segments in a chain. The reason is that even if the degree of an algebraic segment is bounded, it is not necessary that the size of its linear approximation will be bounded. For example, a larger circle has to be approximated by a larger set of linear segments than a smaller a circle. In this setting, we consider the problem of finding the planar partition induced by n linear chains of bounded degree of total size N. We give an $O(N + n \log n + m)$ algorithm to find such a partition, where m is the number of intersections of the chains. Note that if we simply applied the planar partition algorithm of [Mul1] to the derived set of N linear segments, we will get an $O(N \log N + m)$ algorithm.

The other problem treated in this paper is clipping. This is a very effective form of divide and conquer which is extensively used in practice, especially in computer graphics [Suther]. It is done as follows. One first divides a given window into many subwindows, and then "clips" the input against these subwindows. This gives us a set of smaller problems, one for each subwindow, and one can recur if necessary. In practice, the overhead of this conventional clipping per window W is $O(\phi_w + n_w)$, where n_w is the number of endpoints of the input segments within W and ϕ_w is the number of input segments intersecting W. The cost $O(n_w)$ is unavoidable. The linear dependence of the overhead on the "flux" ϕ_w is, however, undesirable, and constitutes a major bottleneck for the conventional clipping. The reason is that the number of intersections between the input and all subwindow borders becomes large quite soon, as one increases the number of subwindows. In this paper, we present a new clipping technique called *virtual clipping*, for which the overhead per window W depends only logarithmically on the flux ϕ_w. And yet, one gets all advantages of the conventional clipping, in the sense that, the work done within any given subwindow, *in the amortized sense*, obeys exactly the same bound as if the input were actually clipped against that subwindow. Note that the cost of virtual clipping is logarithmic in the window flux *regardless of the position of that clipping window with respect to the input.* Besides making the clipping more efficient, this also makes it more robust than the conventional clipping with respect to the decisions regarding the positions and the number of clipping subwindows. Unlike the space requirement of the conventional clipper, the space requirement of the virtual clipper is guaranteed to be linear, regardless of the input or the locations of the clipping subwindows. This makes virtual clipping a favorable alternative to the conventional clipping. Our technique is intimately based on the ideas used in the planar partition algorithm of [Mul1]. The name virtual clipping comes from the fact that our algorithm does not clip the input against the specified subwindows actually, but only *virtually.* This technique can be used not just in connection with the planar partition problem but many others, which include the problems in computer graphics.

As an application of virtual clipping, we give a very efficient planar point location algorithm. Optimal algorithms taking $O(n \log n)$ preprocessing time, $O(n)$ space and $O(\log n)$ query time are known [Lipton,Kirpat,Edel,Sarnak]. (See also [Prep].) Yet a planar point location algorithm based on conventional clipping works equally well and sometimes even better in practice [Edah]. Because of the advantage of virtual clipping over conventional clipping, our algorithm is expected to be a favorable candidate in practice. In the worst case, our algorithm takes $O(n \log n)$ time, but in practice, this should be linear. Its space requirement is guaranteed to be $O(n)$. Contrast this with the $O(n^{3/2})$ worst

case time and space requirement of a conventional clipping algorithm [Edah]. The querry time of our algorithm in practice should be $O(1)$, but no $O(\log n)$ theoretical guarantee can be given. Virtual clipping makes it feasible to choose quite small subwindows, making it unnecessary to set up any elaborate search structure within any subwindow. This makes the search structure of our algorithm simplest among all known search structures for the point location problem. This also means that in practice the query time of our algorithm should compare favorably with that of the other algorithms.

The main theoretical tool which is used in the analysis of the algorithms in this paper is a probabilistic game involving certain stoppers and triggers. This is an extensive generalization of the games considered in [Mul1]. Our success in analyzing this general game directly enables us to handle, in a unifying framework, the situations which we could not handle before. It also allows us to give a more direct analysis of the algorithm in [Mul1], which is a bit simpler. The probabilistic games of a similar nature have extensive applications in quite different situations too [Mul2]. For applications to the hidden surface removal problem see [Mul3].

We shall assume that the reader is familiar with the algorithm in [Mul1], and the terminology used in its description. On the other hand, the analysis in this paper is almost self contained.

2 Stoppers and Triggers

A main theoretical tool used in the analysis of our algorithms is the probabilistic analysis of a certain game. As this game is theoretically interesting in its own right, we shall begin with this game and its analysis. The game itself is a generalization of the similar games in [Mul1]. Its analysis given here is, however, more direct.

Assume that we have three sets M, H and K. We assume that M is linearly ordered. H and K can be unordered. We also assume that H and K are disjoint. They can, however, intersect M. Imagine M placed on the positive real axis, according to its order; the ordering of M increases in the positive direction. This is just for the sake of visualization, otherwise, M is completely abstract. Imagine an observer located at the origin.

Now we shall conduct a novel experiment. The experiment consists in repeatedly selecting, in a random fashion, an element from $M \cup H \cup K$, *without replacement*, until every element from the union has been selected. The observer will be active during a part of this experiment which is determined as follows: The observer becomes active, if at all, immediately *after* all elements of H have been chosen, provided no element from K has been chosen before this instant. If the observer becomes active at all, he will go into the inactive state immediately after some element from K has been chosen; if K is empty he will remain active thereafter. Thus H can be regarded as a set of triggers and K can be regarded as a set of stoppers.

Let us say that an element $a \in M$ was observed by the observer *when it was chosen* if 1) the observer was in the active state at this instant 2) no element $b < a$ in M had been chosen before this instant. The idea is that the chosen elements of M are supposed to act as barriers to the sight of the observer. Hence if an element $b < a$ in M had been chosen before a, the observer could not see a when it was chosen. Let O be the number of elements of M that were observed by the observer in the active state. We want to estimate $E(O)$, the expected value of the random variable O. Let $|M| = m$, $|H| = h$ and $|K| = k$. The functional form of $E(O)$ depends very crucially on what h and k are, as the following theorem shows.

Theorem 1 *Given the sets M, H and K as above, the expected value $E(O)$ is bounded as below:*

1) $h = 0$, $k = 0$: $E(O) \leq \ln(m+1) + \gamma$, *where γ is the Euler's constant.*
2) $h = 0$, $k > 0$:
   a) $K \cap M = \emptyset$: $E(O) \leq \ln\left(\frac{m}{k} + 1\right)$.
   b) $K \cap M \neq \emptyset$: $E(O) \leq 1 + \ln\left(\frac{m}{k} + 1\right)$.
3) $k = 0$, $h > 0$: $E(O) < \frac{2}{h+1}$.
4) $h > 0$, $k > 0$:
   a) $K \cap M = \emptyset$: $E(O) \leq \frac{1}{\binom{h+k}{k}} = \frac{1}{\binom{h+k}{h}}$.
   b) $K \cap M \neq \emptyset$: $E(O) \leq \frac{2}{\binom{h+k}{k}} = \frac{2}{\binom{h+k}{h}}$.

Proof. Let O_i be the random variable which is one if the ith element of M was observed by the observer in his active phase, and zero otherwise. Then $O = \sum_i O_i$ and $E(O) = \sum_i E(O_i)$. Hence, it suffices to estimate $E(O_i)$, for every i.

For a fixed i, let c_i be the ith element in M and let M_i denote the subset of M consisting of the elements less than or equal to c_i. We make the following crucial observation: $E(O_i)$ can depend only on the sets M_i, H, and K. This is the restriction argument that was used in [Mul1], and is easy to prove. Informally this can be easily seen as follows. Imagine a new observer who can only see the elements in $M_i \cup H \cup K$. Then as far as he can see, the elements in $M_i \cup H \cup K$ are still chosen with a uniform randomness, and by the very definition of O_i, it is strictly a function of what happens within the universe $M_i \cup H \cup K$.

First consider only the case $K \cap M = \emptyset$. Let $U_i = M_i \cup H \cup K$.

To estimate $E(O_i)$, we now restrict our attention to $M_i \cup H \cup K$.

First notice that $E(O_i) = 0$, if $H \cap M_i \neq \emptyset$. By our assumption, K and M are disjoint and H and K were already disjoint. Thus we need to estimate $E(O_i)$ only when M_i, H and K are all disjoint; thus $u_i = |U_i| = i + h + k$.

Now notice that c_i can be observed if and only if

1) the first h elements to be chosen from U_i, belong to H. The probability of this happening is $\frac{h!}{u_i(u_i-1)\cdots(u_i-h+1)} = \frac{h!}{(i+h+k)\cdots(i+k+1)}$.
2) The $(h+1)$st element chosen from U_i is c_i. The probability of this happening is $\frac{1}{(U_i-h)} = \frac{1}{i+k}$.

Hence, $E(O_i) = \frac{h!}{(i+h+k)\cdots(i+k)}$.

If $H \cap M_i \neq \emptyset$, $E(O_i) = 0$. Thus we conclude:

$$E(O_i) \leq \frac{h!}{(i+h+k)\cdots(i+k)}.$$

Now we proceed by cases.

1) $h = 0$, $k = 0$: in this case $E(O_i) \leq \frac{1}{i}$, hence

$$E(O) = \sum_{i=1}^{m} \frac{1}{i} \approx \ln m + \gamma,$$

where γ is the Euler's constant.

2) $h = 0$, $k > 0$: in this case $E(O_i) \leq \frac{1}{i+k}$ and hence

$$E(O) = \sum_{i=1}^{m} \frac{1}{i+k} \leq \int_0^m \frac{1}{x+k}\,dx = \ln\left(1+\frac{m}{k}\right).$$

3) $k = 0$, $h > 0$: in this case $E(O_i) \leq \frac{h!}{(i+h)\cdots i} = \frac{1}{(h+1)} \cdot \frac{1}{\binom{i+h}{h+1}}$. Hence

$$\begin{aligned} E(O) &\leq \frac{1}{h+1}\sum_{i=1}^{m} \frac{1}{\binom{i+h}{h+1}}, \\ &< \frac{1}{h+1}\sum_{j=0}^{\infty} \frac{1}{\binom{j+r}{r}}, \quad \text{where } r = h+1, \\ &\leq \frac{1}{h+1}\sum_{j=0}^{\infty} \frac{1}{\binom{j+2}{2}}, \quad \text{as } r \geq 2, \\ &= \frac{2}{h+1}\sum_{j=0}^{\infty}\left(\frac{1}{j+1} - \frac{1}{j+2}\right) \\ &= \frac{2}{h+1}. \end{aligned}$$

4) $h > 0$, $k > 0$: in this case $E(O_i) \leq \frac{h!}{(i+h+k)\cdots(i+k)} = \frac{1}{h+1} \cdot \frac{1}{\binom{i+h+k}{h+1}}$. Hence

$$\begin{aligned} E(O) &\leq \frac{1}{h+1}\sum_{i=1}^{m} \frac{1}{\binom{i+h+k}{h+1}} \\ &< \frac{1}{r}\sum_{j=k}^{\infty} \frac{1}{\binom{j+r}{r}}, \quad \text{where } r = h+1 \text{ (thus } r \geq 2), \\ &= \frac{1}{r}\sum_{j=k}^{\infty} \frac{1}{\frac{(j+r)(j+r-1)}{r(r-1)}\binom{j+r'}{r'}}, \quad \text{where } r' = r-2, \\ &\leq (r-1)\sum_{j=k}^{\infty} \frac{1}{(j+r)(j+r-1)\binom{k+r'}{r'}} \\ &= \frac{(r-1)}{\binom{k+r'}{r'}}\sum_{j=k}^{\infty}\left(\frac{1}{j+r-1} - \frac{1}{j+r}\right) \\ &= \frac{r-1}{\binom{k+r'}{r'}} \cdot \frac{1}{(k+r-1)} \\ &= \frac{1}{\binom{k+r-1}{r-1}} = \frac{1}{\binom{k+h}{h}} = \frac{1}{\binom{h+k}{k}}. \end{aligned}$$

Now consider the case when $K \cap M \neq \emptyset$.

Let $M' = M - K$ be the relative complement of M with respect to K and let $m' = |M'|$. Notice that M', K, H are mutually disjoint. Let O' be the number of elements from M' which were observed by the observer during the active state. Then $E(O')$ can be bounded as in the preceding part of the proof, because M', K, and H are mutually disjoint.

Let O'' be the number of elements from K which were observed by the observer during his active state. It remains to bound $E(O'')$. Note that $O'' \leq 1$, as the observer becomes inactive immediately after an element from K is chosen. In case $h = 0$ and $k > 0$, this immediately implies that $E(O) \leq 1 + \ln\left(1 + \frac{m'}{k}\right) \leq 1 + \ln\left(1 + \frac{m}{k}\right)$. The only case remaining is the case when $h > 0$ and $k > 0$. Fix an element c in k. Now note that an element c from K can be observed only if (but not necessarily if):

1) The first h element chosen from $H \cup K$ belong to H and
2) the $(h+1)$st element chosen from $H \cup K$ is c itself.

If O_c denotes the random variable which is one if c is observed and zero otherwise, it follows that

$$E(O_c) \leq \frac{h!}{(k+h)\cdots(k+1)} \cdot \frac{1}{k}.$$

Hence

$$E(O'') = \sum_{c \in K} E(O_c) \leq \frac{h!}{(k+h)\cdots(k+1)} = \frac{1}{\binom{k+h}{h}} = \frac{1}{\binom{h+k}{k}}.$$

The rest follows because $E(O) = E(O') + E(O'')$. □

3 Algebraic segments

In [Mul1] we gave a randomized optimal, and efficient algorithm to find the partition of a plane induced by a set of linear segements. Now we examine what happens if the segments are algebraic instead, but of a bounded degree. There are two approaches to this problem, one is purely algebraic and the other is based on linear approximations to the algebraic segments.

3.1 An algebraic approach

In this section, we shall take the algebraic approach. For the sake of simplicity, we shall assume that the segments are bounded and are surrounded by a window. If there are unbounded segments, we can always take the window at "infinity".

We have to first decide how an algebraic segment is going to specified in an unambiguous way.

Let us first consider a simpler case when the algebraic segment in question is monotonic and bounded; we say that a segment is monotonic if it intersects any vertical line at most once. In this case, we specify the algebraic equation, $f(x, y) = 0$, that the segment satisfies. We also specify, in addition, the two endpoints of the segment. As the segment is monotonic, we can also orient it so that the x coordinate increases in the direction of the orientation. Unfortunately,

all this information is not always enough. For example, let p and q be the two extreme points of the circle where the tangents are vertical. Also assume that p has the smaller x coordinate. Now there are two segments from p to q which satisfy the same algebraic equation. Thus we also need to specify the tangential orientation of the segment at p. This can be done, for example, just by specifying if the segment is oriented upwards or downwards at p.

We assumed above that the segment is monotonic. However, monotonocity is not a severe restriction. Indeed, any algebraic curve of a bounded degree can always be broken down into a bounded number of monotonic segments. Let us see how. Assume that the curve satisfies an equation $f(x,y) = 0$, which is of a bounded degree. We find all "critical' points on the curve where the tangents become vertical. These points satisfy the equations $f(x,y) = 0$ and $\frac{\partial f}{\partial y} = 0$. By Bezout's theorem, there are only a bounded number of solutions to this system of equations. Moreover, all these equations can be easily found out by forming a resultant R, which is a polynomial of a bounded degree in one variable. As R has a bounded degree, it is reasonable assume that its roots can be found in a bounded time. In practice, the roots have to be found by a numerical method, such as Newton's method. Hence, to be precise we should also take into consideration the bit complexity of the root extraction. This indeed can be done, because in an algebraic problem such as this, we only need to approximate a root by calculating its first "few" bits [Canny]. For simplicity, we shall ignore the bit complexity issue here, and just charge ourselves $O(1)$ time for the extraction of a root. So assume that we know all real solutions of the system $f(x,y) = 0$ and $\frac{\partial f}{\partial y} = 0$. This gives us all critical points on the curve. But we also need to know how these critical points are topologically connected to each other. Towards this end, define the rank of a point a on the curve $C : f(x,y) = 0$ to be the number of points of intersection between the curve C and the semi-infinite vertical line going upwards from a. The rank at $a = (x_0, y_0)$ can be found by simply evaluating the Sturm sequence of the function $g(y) = f(x_0, y)$ at $y = y_0$ [Waerden]. Such Sturm sequences were also used by Canny in his road map algorithm [Canny]. Once we know the rank of every critical point as well as its type (whether it is a left extreme or a right extreme) it is easy to figure out the topological structure by just scanning the critical points from left to right. Hence, we can assume hereafter that the segments are monotonic.

We also need to specify how to find the points of intersection of two monotonic segments R and S which satisfy the equations $f(x,y) = 0$, and $g(x,y) = 0$ respectively. By Berout's theorem, the number of solutions to these equations is bounded. We find, as before, all real solutions to the system $f(x,y) = 0$ and $g(x,y) = 0$. This gives us a set of points V of plausible solutions. Next we need to know which points in V lie on R and S. Let us see how we can find the subset of V which lies on R. We can then do a similar thing for S. Let r_0 and r_1 be the endpoints of R and assume that r_0 has the smaller x coordinate. Define the rank of R at a given x-coordinate x_0 to be the number of intersections between the curve $C : f(x,y) = 0$ and the vertical line $x = x_0$, which lie strictly above R. Using the Sturm sequences as above, it is easy find the rank of R at any x coordinate by scanning the critical points of C from left to right. Now a point $a = (x_0, y_0)$ in V belongs to R iff the rank of a coincides with the rank of R at x_0. The scheme given here is purely theoretical. As we shall see later, in practice, we can do much better.

Now assume that we are given n bounded monotonic algebraic segments. All segments will be oriented in the direction of the increasing x-coordinate. We wish to find the induced partition of the plane, which is formed by passing a vertical attachment through each endpoint extending in either direction to a window border or another input segment.

The algorithm is an extension of the algorithm in [Mul1], so we shall only elaborate upon the differences. There were actually two algorithms given in [Mul1]. The algorithm in this section is an extension of the one which maintains a trapezoidal decomposition of the window at every stage.

We first form an initial partition G_0 of the window by passing vertical attachments extending to the window borders through all endpoints of the input segments. All these vertical attachment are contractible and will contracted throughout the algorithm repeatedly. Starting with G_0, we successively add the segments in a random order to get a succession of partitions $G_0, G_1, \ldots, G_n$, where G_n will be the partition sought.

Shown in fig 1 is a partition G_4 obtained after adding four randomly selected segments to G_0. Note how the vertical attachments through various endpoints have been contracted. Also note that each face of partition has a face-length less than or equal to four. This is done by passing a contractible vertical attachment through every point of intersection encountered. Because these vertical attachments are contractible, they will be contracted in the course of the algorithm, just like any other contractible attachment. Finally, note that the partition G_4 also contains vertical attachments through the endpoints of the segments not yet added. In the fig. 1, these are shown as dotted.

Remark: It is actually not necessary to maintain the points of attachment of the vertical segments through the endpoints of the segments not yet added. For example, consider the trapezoid $pqrs$ in fig. 1 which contains endpoints a_0 and a_1. The segments through the points a_0 and a_1 have not been added so far. In the figure the trapezoid $pqrs$ has been explicitly decomposed further by the vertical attachments through the points a_0 and a_1. In practice, this explicit decomposition is not necessary. Indeed, we only need to maintain the representation of $pqrs$ together with a list of the interior endpoints of the unadded segments ordered by their x-coordinates. This representation is more efficient. However, for the sake of conceptual simplicity, we shall assume that all decompositions are explicit. The same remark is also applicable to all other algorithms to be discussed in this paper.

The inductive step of adding $S = S_{k+1}$ to G_k is achieved by 1) locating the first region R_0 in G_k that S begins to traverse, 2) travelling along S in G_k, by repeatedly doing face traversals and face transitions, 3) updating G_k as we travel.

First consider the face traversal. Shown in fig 2a, is a face R of G_k which the segment S has entered through a point a_0. We wish to find the "next" point a_1 on the border of R that S intersects. When all segments are linear, R is convex, and hence, apart from a_0, there a unique point of intersection between S and the border of R. When the segments are algebraic, this need not be the case in general. Hence, using the procedure given in the begining of this section, we find all points of intersection between S and every border of R; remember that R has at the most four borders. Among these points of intersection, we select the one which is next to a_0, in its x-coordinate. This point is a_1. (In practice, we do not need to find all points of intersection between S and a border of R. If one using Newton's method, we can use as an initial guess a point close to a_0.) Note that we can return to the same face R of the old partition G_k several times; see fig 2a again. If we update G_k as we travel through it, this should cause no problem. It is important for this reason, however, that the update proceed concurrently with the travel. Monotonicity ensures that at no time in our travel we need to contract a vertical attachment added during the preceding part of our travel along S.

Next consider a face transition. Fig 2b shows S, the segment being added, about to leave the face R_0 of G_k at point v. Suppose that the point of exit v lies on an input segment T. (If v lies on a vertical attachment, the case is easier.) We travel left on T until we meet the vertex a which is visible on the other side of T; note that travelling "left" makes sense because T is monotonic. At a we turn around, and travel right until we meet S again on the other side of T. When we do so, we are in the face R_1 that S enters next. We are now ready for the next face traversal.

We travel in G_k from the initial point of S to its endpoint, using repeated face traversals and face transitions. As we travel, we update G_k. This consists in splitting the faces of G_k that are traversed by S and appropriately contracting the vertical attachments that are intersected by S.

The partition G_n obtained at the end of the above algorithm, is precisely partition that we sought.

The analysis of this algorithm is very similar to the analysis of the algorithm for linear chains, to be considered next. Hence, we shall merely state the final result.

Theorem 2 *The expected running time of the above algorithm is $O(d^2(m + n\log n))$, where m is the number of intersections, n is the number of segments, and d is a bound on the degrees of the segments.*

3.2 Linear chains

In this section we shall approach the problem concerning algebraic segments in a different way. This approach is based on linear approximations. We shall approximate every algebraic segment by a chain of linear segments. The problem now is to find the planar partition induced by a given set of linear chains. As every chain is meant to represent an algebraic segment of a bounded degree, we will assume, in this approach, that its degree is bounded; the degree of a chain is defined to be the maximum number of intersections between the chain and a straight line. For the sake of simplicity, we will also assume, in addition, that every chain is monotonic, i.e. it intersects any vertical line at most once. This assumption can be readily removed. Note that we are not making any assumptions about the size of a chain, i.e. the number of linear segments in a chain. Let n be the number of linear chains, and let N be the sum of the sizes of all chains. In this section, we give an $O(N + n\log n + m)$ algorithm to find a partition of the plane induced by the chains, where m is the number of intersections of the chains.

The algorithm is an extension of the algorithm in [Mul1], so again we shall only elaborate upon the differences.

Each chain has two endpoints and possibly many intermediate points, which will be called link points. We form the initial partition G_0, by passing contractible vertical attachments extending to the window borders, through the endpoints of the chains. *Note that G_0 does not contain vertical attachments through the link points.* Now we successively refine G_0, by throwing the chains in a random order, so as to get a succession of partitions $G_0, \ldots, G_n$. G_n will be the partition sought.

Consider the $(k+1)$st refinement which consists in adding a randomly chosen chain $S = S_{k+1} = (s_0, s_1, \ldots, s_k)$ to G_k, so as to get G_{k+1}. We locate the initial face R_0 that S starts traversing, by going to the vertical attachment associated with s_0. Now we travel from s_0 to s_1 in G_k and update it precisely as in [Mul1]. The only difference comes when we arrive at s_1, which can lie in the middle of some face R; see fig 3a. In this case, we split the face R by passing a *contractible* vertical attachment through s_1 which extends in either direction to the border of R. Being contractible, the attachment will contract later just like the other contractible attachments. Having done this, we proceed to s_2, and so on. Note that we can visit the same face R of the old partition G_k several times (see fig 3b), but if we update G_k, as we travel, this should cause no problem.

For this algorithm, the following two theorems hold.

Theorem 3 *The expected number of face splits is $O(n\log n + N + m)$, where m is the number of intersections of the chains. The constant within O is small and does not depend upon the degree bound d.*

Theorem 4 *The expected number of vertices visited during the face transitions is $O(d^2(n\log n + N + m))$, where d is the degree bound. If the same vertex is visited during many face transitions, then every visit is counted.*

Theorem 5 *The average facelength of the partition remains less than or equal to 4 throughout the algorithm.*

Just as in [Mul1], Theorem 3 and Theorem 4 by themselves do not guarantee that the expected running time of the algorithm is $O(n\log n + N + m)$, because the facelength, at least theoretically, is unbounded. Theorem 5 indicates that this should not be a problem in practice. One can get around this problem, as in [Mul1], by passing a contractible vertical attachment through every point of intersection encountered in the algorithm. For this new algorithm, Theorem 3 and Theorem4 still hold. As every face formed in this

algorithm has length ≤ 4, it follows that the algorithm is $O(n \log n + N + m)$. In the following analysis we shall only analyze this new algorithm for linear chains. The proof of Theorem 5 is similar to the proof of the analogous theorem for the planar partition algorithm in [Mul1].

If t is an endpoint, a link point, or a point of intersection, we denote by F_t the number of times the vertical attachment through t is contracted after it came into existence.

The total number of face splits $F = m + N + \sum F_t$. Thus we only need to estimate $E(F_t)$. For a vertex t, define n_t to be the vertical spanlength at t, i.e. the number of chains that intersect the imaginary vertical line through t. The following lemma proves Theorem 3.

Lemma 1 *1. If t is an endpoint, then $E(F_t)$ is $O(\log n_t)$.*

2. If t is a point of intersection, then $E(F_t)$ is $O(1)$.

3. If t is a link point, then $E(F_t)$ is $O(1)$.

Proof.

Case 1, t is an endpoint: Let F_t^u (F_t^l) be the number of times the upper (lower) part of the vertical attachment through t is contracted in the course of the algorithm. We will only estimate $E(F_t^u)$; $E(F_t^l)$ can be estimated similarly. Let M be the set of chains which intersect the imaginary semi-infinite vertical line going upwards from t. Because of the monotonicity, each chain can intersect this line only once. Hence, the set M can be linearly ordered according to the order of the associated points of intersection, with the order increasing upwards. Let H and K be empty sets. The Lemma follows by applying Theorem 1 to the sets M, H and K.

Case 2, t is a point of intersection of chains R_1 and R_2:

We proceed precisely as above. The only difference is that now we let $H = \{R_1, R_2\}$ instead.

Case 3, t is a link point of a chain R:

Let $H = \{R\}$, and proceed as before. []

Now we turn to estimating the cost of face transitions. A face transition across a vertical attachments is achieved in a constant time. Moreover, a face transition across a vertical attachment is always accompanied by a contraction (splitting) of that attachment. As we have already estimated the expected number face splits, we need not worry about the face transitions across vertical attachments anymore. We estimate the expected cost of the rest of the transitions by amortization. More precisely, we shall distribute this cost among the endpoints and the link points of the chains, as well as the points of intersections of the chains. Then we only need to estimate the cost charged to a fixed endpoint, link point, or a point of intersection. Towards this end, we first define, as in [Mul1], what it means for a point of attachment p, lying on a segment T of some input chain, to witness a face transition when a new chain S is being added. Notice first that the number of intersections between T and S is bounded. Hence, during the addition of S, only a bounded number of face transitions can occur across T. Fix a point of attachment p on T, and define the right side and the left side of T with respect to p arbitrarily. We say that p witnesses, on its right side, a face transition along S and across T if 1) S intersects T, 2) there is no chain C, added before S, that intersects T on the right side of p and before S (more precisely, between p and the nearest point, on the right side of p, where S intersects T). Note that every point of attachment that is visited during a face transition of S across T is a witness to the transition, but not conversely. Moreover, the number of transitions of S across T is bounded by d, where d is the degree bound. This means that, if p is a witness to a transition of S across T, it can be visited at the most d times during the face transitions along S. If c ($\leq d$), is the number of intersections between S and T, all face transitions of S across T can be carried out in time $O(cw+c)$, where w is the number of points of attachment on T which were the witnesses. The cost $O(c)$ can be charged to the c new points of intersection formed. Hence, we only need to estimate the expected sum of w over all transitions. If t is an endpoint, a link point, or a point of intersection, let Q_t be the number of face transitions witnessed by an either end of the vertical attachment through t, after it came into existence. Theorem 4 now follows from the following lemma.

Lemma 2 *1. If t is an endpoint, then $E(Q_t)$ is $O(\log n_t)$.*

2. If t is a point of intersection, then $E(Q_t)$ is $O(1)$.

3. If t is a link point, then $E(Q_t)$ is $O(1)$.

Proof.

Case 1, t is an endpoint of a chain:

Let ϕ_t be the set of chains which intersect the infinite vertical line T through t. For every $S \in \phi_t$, place an observe o_s at the (unique) intersection of S with T. Fix S. Let M_s^l be the set of chains which intersect S to the *left* of o_s (looking from t). M_s^r is defined analogously to be the set of chains intersecting S to the right of o_s. Let $H_S = \{S\}$. Let K_s be the set of chains intersecting T between o_s and t. We linearly order M_s^l by the rightmost points of intersection of chains in M_s^l with S. More precisely, a chain $R \in M_s^l$ is defined to be less than $R' \in M_s^l$ iff the rightmost point of intersection between R and S is to the right of the rightmost of point of intersection between R' and S. Thus the ordering increases along S away from the observer o_s in the left direction.

Define the active state of the observer o_s, as in Theorem 1, by letting $M = M_s^l$, $H = H_s$ and $K = K_s$. Let $k_s = |K_s|$. Let O_s^l be the number of elements observed by o_s during his active state. O_s^r is defined analogously. From Theorem 1, it follows that $E(O_s^l)$ is $O\left(\frac{1}{k_s}\right)$. Similarly $E(O_s^r)$ is $O\left(\frac{1}{k_s}\right)$. Let $O_s = O_s^l + O_s^r$ be the number of elements in $M_s^l \cup M_s^r$ observed by o_s along S in either direction, during his active state. Then $E(O_s)$ is $O\left(\frac{1}{k_s}\right)$. Now it is easy to see that $Q_t = \sum_{s \in \phi_t} O_s$. Hence $E(Q_t) = O(\sum_{s \in \phi_t} \frac{1}{k_s}) = O(\log n_t)$.

Case 2, t is a point of intersection of chains R_1 and R_2:

Proceed as above, but now let $H_s = \{S, R_1, R_2\}$. From Theorem 1, it follows that $E(O_s)$ is $O\left(\frac{1}{k_s^3}\right)$. Hence $E(Q_t) = O(\sum_{s \in \phi_t} \frac{1}{k_s^3}) = O(1)$.

Case 3, t is a link point of a chain R:

Now let $H_s = \{S, R\}$. From Theorem 1, it follows that $E(O_s)$ is $O\left(\frac{1}{k_s^2}\right)$. Hence $E(Q_t) = O(\sum_{s \in \phi_t} \frac{1}{k_s^2}) = O(1)$. □

Remark: The analysis of this section, when specialized to the case of segments, instead of the chains of segments, gives a somewhat simpler and a more direct proof of the optimality of the algorithm in [Mul1].

4 Virtual clipping

The planar partition algorithm in [Mul1] takes $O(m+n \log n)$ time, where n is the number of input segments, and m is the number of intersections. We are interested in knowing if the second term can be made almost linear in n in practice. Clearly if m is very large compared to $n \log n$, this is worthless. But if m is not that large, as is often the case in computer graphics, this is clearly useful. We shall see how this can be achieved by using a form of clipping, which we shall call *virtual clipping*. In this section, we shall describe the algorithm that results when virtual clipping is incorporated in the algorithm of [Mul1].

But first let us recall the conventional clipping. This is a form of divide and conquer which is often used in practice. For the planar partition problem, the conventional clipping can be used as follows.

1) divide the window into subwindows,
2) "clip" the input segments against the subwindows,
3) solve the subproblem for each subwindow. If the size of the clipped input for a subwindow is below a certain threshold use "the basic algorithm" to solve the subproblem, otherwise recur.

Two factors determine the efficiency of the resulting algorithm: 1) the cost of clipping; 2) the cost of the basic algorithm. Indeed the threshold size in the third step has to be chosen judiciously, so that the two costs are balanced. The cost of conventional clipping per window W is at least $O(a.n_w + \phi_w)$, where n_w is the number of endpoints of the input segments within W, ϕ_w is the number of intersections of the input segments with the borders of W, and a is the search time required to locate a subwindow containing a given endpoint of a segment. In practice, one can use some kind of a bucket search to locate a subwindow, hence a is almost a constant. Hence the cost of conventional clipping per window W is $O(\phi_w + n_w)$ in practice. The cost of $O(n_w)$ is unavoidable. The cost $O(\phi_w)$, on the other hand, is undesirable, and constitutes a major bottleneck in the conventional clipping. The reason is that the number of intersections between the input and all subwindow borders becomes large quite soon, as one increases the number of subwindows.

The virtual clipping introduced in this section clips the input against any subwindow not actually but "virtually". For all practical purposes, the cost of virtual clipping per window W is $O(n_w + \log(1 + \phi_w))$. This clearly shows why it is preferable to the conventional clipping. Finally, our "basic algorithm" will be none else but the planar partition algorithm in [Mul1]. We have already seen in [Mul1] that it is optimal and very efficient. This makes the combination of virtual clipping and the basic algorithm of [Mul1] a very efficient algorithm in practice.

Incorporation of virtual clipping in the basic algorithm of [Mul1] turns out to be very natural. We shall describe now the algorithm that results after this incorporation. As to be expected, it does not clip the input segments against the subwindows right in the beginning, but only when it becomes necessary. In the begining of the algorithm, we divide the window, hierarchically, based only on the distribution of the endpoints of the input segments. This subdivision of the window can be done recursively, until the number of endpoints in every subwindow is below a certain threshold. Recall that the conventional clipper will recursively clip a subwindow further if the number of input segments intersecting the subwindow is larger than a threshhold. The number of input segments intersecting a given subwindow can be much larger than the number of endpoints contained within the subwindow. Because our clipping is virtual, we can not base our decisions on the number of input segments intersecting a subwindow. Indeed, we can not afford to know this number! But, let us see why it is justified, in our case, to make the decisions regarding the number and the locations of the subwindows solely on the distribution of the endpoints. There are three reasons for this. First, our basic algorithm, which is the planar partition algorithm in [Mul1], discovers every point of intersection t of the input segments in a constant (amortized) time regardless of the size of the window containing t. Hence, as far as the decision regarding the subwindows is concerned, one does not need to worry too much about the time spent in detecting the intersections of the input segments. Secondly, the running time of the virtual clipper will depend only logarithmically on the flux through any subwindow. Thirdly, if we disregard the time spent in detecting the intersections of the input segments, the time spent "within" any subwindow will depend, in the amortized sense, only on the number of input endpoints contained within that subwindow. All this makes it feasible to divide the window right in the beginning of the algorithm, solely on the basis of the distribution of the input endpoints.

So assume that we are already given a subdivision of the main window. Theoretically, for an arbitrary input, a subdivision of the main window can always be found recursively in $O(n \log n)$ time, so that the number of endpoints within every subwindow of this division is below a required threshold. In practice, invariably, some kind of a bucket sort can be used, which should take almost linear time. In the rest of the section, we shall make *no assumption* about the window subdivision or the input. We simply assume that a window subdivision is given to us, and that we are also told which endpoint belongs to which subwindow. Other than this, we assume nothing regarding the input. In fact, we do not even assume that the number of endpoints within a subwindow is below any threshold. Obviously better the subdivision of the window, better the running time. But the lack any extra

assumptions will clearly show us how robust our algorithm is with respect to the locations of the clipping subwindows.

The algorithm is an extension of the planar partition algorithm in [Mul1], and has the same outline: we first form an initial partition G_0, and refine it successively by adding a randomly chosen input segment to get a succession of partitions, $G_0, \ldots, G_n$. We shall elaborate upon the differences.

The first major difference comes right in the formation of initial partition G_0. In the basic algorithm of [Mul1], the initial partition is formed by passing a vertical attachment through every endpoint which extends in either direction upto the main window border. Now we shall form G_0, by passing a vertical attachment through every input endpoint t, which extends in either direction up to the borders of the subwindow containing t. Fig 4 shows a subdivision of the main window into seven subwindows, based on the distribution of the endpoints in an input. The resulting initial partition G_0 is also shown. The window borders are thickened and we shall follow the same convention in other figures.

After this the algorithm proceeds as in [Mul1]: starting with G_0 we successively refine the current partition by adding a randomly chosen segment to it. The refinement of G_k due to the addition of a segment $S = S_{k+1} = (s_0, s_1)$ consists in: 1) locating the first face R_0 of G_k that S starts traversing, by using a pointer to the vertical attachment through s_0; 2) travelling from s_0 to s_1 in the partition by doing face traversals and face transitions repeatedly; 3) updating the partition as we travel. The only additional thing that needs to be specified is what happens when one passes through a subwindow border. In fig 5a, the new segment $S = S_{k+1}$ crosses the border between windows W_1 and W_2. It is clear that the part of the border just to the right of S can be erased (contracted) without violating the convexity of the resulting partition. The resulting partition is shown in fig 5b. A similar thing can be done if the part of the border immediately to the left of S can be contracted. Shown in fig 5c is a situation where no contraction can be carried out, as that will destroy the convexity of the partition. Note that, because of our rules for contraction, the situation in fig. 5d cannot arise. Also, there is one exception to the above rule for contraction. We do not contract a window border if this involves destroying (or removing) a window corner. Thus, in fig. 5e, we contract only one part (see fig. 5f).

It also becomes necessary to modify the procedure in [Mul1] for a face transition, when the transition takes place across a subwindow border. In fig. 6 the new segment S is about to leave a face in the window W_1 and enter a face in the window W_2. The procedure in [Mul1] will find the face of W_2 that S enters, as follows (see fig 6): travel left on the border until we reach the first vertex d which is visible on the other side of the border. Here turn around, and travel right until we reach S again on the other side. When we do so, we are in the correct face. The problem here is that we end up visiting too many points of attachment, especially the ones in the windows W_3, W_4, W_5, One can get around this difficulty as follows. Recall that by our definition of visibility in [Mul1], the window corner a is invisible in the face R. Now we shall force a to be visible in R. This is done by simply including a in the representation of the face R. A similar thing is done for every window corner, which is a t-junction. Coming back to the transition of S across the border between W_1 and W_2, it is clear that we can now turn around at the corner a, thereby visiting only the points of attachments on the borders of W_1 and W_2.

The rest of the algorithm, is as in [Mul1], hence we shall not discuss it any further. However, a few remarks are in order. Even when the input segment S lies strictly within a window, we can "stray" outside the window during our travel along S. Fig 7 shows straying during one typical face transition during a journey along S. Such straying cannot be avoided. Indeed, that is an unavoidable part of the virtual clipping. Yet, our analysis shows that, in the amortized sense, the expected work done "within" any subwindow is close to the expected work that would be done even if the input were actually clipped along the window border. Thus we get the full effect of the conventional clipping at a nominal overhead.

Remark: Virtual clipping can also be used in conjuction with our algorithms for algebraic segments and linear chains. In fact, it can be used in any problem that benefits from the conventional clipping. These include the problems in computer graphics.

The analysis of virtual clipping is somewhat complicated. It can be found in [Mul4]. Here we shall merely state the final result.

For t, which is an input endpoint or a point of intersection of input segments, let n_t^u denote the number of input segments which intersect the imaginary vertical segment extending upwards from t to the border of the subwindow containing t. We define n_t^b similarly with respect to the bottom border of the window containing t. We shall call $n_t = n_t^u + n_t^b$ the vertical spanlength at t. For a window W, let ϕ_w^l denote the number of input segments intersecting the left border of W. Note that this border is going shrink in the course of the algorithm, but ϕ_w^l is defined with respect to the initial complete left border of w. We define $\phi_w^r, \phi_w^u, \phi_w^b$ for the right, upper, and bottom borders similarly. Let $\phi_w = \phi_w^l + \phi_w^r + \phi_w^u + \phi_w^b$ be the total flux through W. For a given window W, let n_w be the number of endpoints within W, and let m_w be the number of intersections of the input segments within W.

It turns out that the total cost (i.e.the running time) of the algorithm can be amortized in such a way that if C_w is the total cost (of face splits as well as transitions) charged to the endpoints, and the intersections of the input segments within W, as well as to the corners of W, then

Theorem 6 $E(C_w) =$

$$O\left(\log(1+\phi_w) + \sum_{t \in W}\left(\log\left(\frac{1+\phi_w^u}{1+n_t^u}\right) + \log\left(\frac{1+\phi_w^b}{1+n_t^b}\right)\right)\right)$$

$+O(n_w + m_w + \log(1+n_t))$, *where t ranges over the endpoints within W. The expected running time of the algorithm is obtained by summing $E(C_w)$ over all windows.*

Note that $O(m_w + n_w + \sum_{t \in W} \log(1+n_t))$ is precisely the expected running time of the basic algorithm in [Mul1], if we

were to actually clip the input against the subwindow W and then run this basic algorithm on the clipped input. (This excludes the cost of actual clipping.) The remaining terms in the expression for $E(C_w)$ give the overhead of virtual clipping. Note that $\frac{1+\phi_w^u}{1+n_t^u}$ is the ratio of a horizontal flux and a vertical flux. For a square window, the two flux quantities should be comparable. Hence, the logarithmic ratio of the horizontal and the vertical flux, when averaged over all endpoints within the window W, is very close to a constant. This means that, for all practical purposes, the overhead of virtual clipping per window W is $O(\log(1+\phi_w)+n_w)$. In contrast, the overhead of the conventional clipping per window W is $O(\phi_w+n_w)$. This clearly shows why virtual clipping is preferable to the conventional clipping.

5 A planar point location algorithm

As an application of virtual clipping, we will give an efficient planar point location algorithm. The point location problem is defined as follows. We are given a planar graph G, not necessarily connected. We have to preprocess the input and build a search structure such that given a query point p, we can quickly locate p within G.

Assume that the input graph G is surrounded by a window. Subdivide the window (recursively, if necessary) into many "buckets" or subwindows, such that each subwindow contains only a constant number of input points. Now run the algorithm of the last section. This builds a planar partition induced by G. But this planar partition also has in addition, the corners of all subwindows or buckets embedded in it. This turns out to be useful in answering a query. Given a query point p, we first locate it in the appropriate bucket (subwindow), using a bucket search; in practice this will take only a constant time. Now choose any corner u of this subwindow, which is already embedded in the partition, and travel from u to p precisely as in the partition algorithm, by repeatedly doing face traversals and face transitions. In practice, this should again take only a constant time. Using the terminology of Theorem 6, it follows that the preprocessing time is $O(n_w+\log(1+n_t))+O\left(\sum_W \log(1+\phi_w)+\sum_t\left(\log\left(\frac{1+\phi_t^u}{1+n_t^u}\right)+\log\left(\frac{1+\phi_t^b}{1+n_t^b}\right)\right)\right)$, where W ranges over all windows, and t ranges over all endpoints in G. Here n_t is the vertical span length at t defined with respect to the window W containing t and ϕ_t^u is the flux through the upper border of the window W containing t; ϕ_t^b is similarly defined. (Note that in Theorem 6, m_w, the number of intersections of the input segments within a window W, is now set to zero.) In the worst case, this preprocessing time is $O(n\log n)$, where n is the number of vertices in G. By what we have discussed in the last section, it is clear that the dominating term in the running time expression is $\sum_t \log(1+n_t)$, where t runs over all vertices of G. Remember that n_t is the number of edges in G that intersect the imaginary vertical segment through t extending only upto the borders of the subwindow containg t. By choosing the subwindows sufficiently small, one can ensure that this term too becomes almost linear in n. Thus, in practice, the algorithm should run in linear time. The space requirement of the algorithm is always $O(n)$. Though the search time should be $O(1)$ in practice, no theoretical $O(\log n)$ guarantee can be given.

We shall compare our algorithm with the one based upon conventional clipping [Edah], which is reported to be as fast, and sometimes faster, than other planar partition algorithms. For this conventional clipping algorithm no theoretical guarantee can be given even for the preprocessing time or the storage requirement—they could be as high as $O(n^{3/2})$. But even in practice, the storage and the time requirement of the virtual clipper should be better, because it detects and retains only a few points of intersection with the window borders. This also makes it possible to choose the subwindows quite small, so that the number of input points within any given subwindow is a small constant. In turn, this makes it unnecessary to build any elaborate search structure within a subwindow. Conventional clippers do not have this freedom. Unlike the algorithm of [Edah], our algorithm detects the point of intersection of the edges in the input graph G, if it is not actually planar. This makes it more robust.

6 Concluding remarks

In [Mul4], we also give one more, completely different, algorithm to find the planar partition induced by a set of linear segments. A novel feature of this algorithm is that it combines randomization with a topological sweep, as in [Guib]. The theory of probabilistic games, as used in this paper, can be extended much further. For this extension and the related applications see [Mul2] and [Mul3].

References

[Canny] Canny J., The complexity of robot motion planning, Ph.D. thesis, M.I.T., 1987.

[Ch] Chazelle B., Edelsbrunner H., An optimal algorithm for intersecting line segments in the plane, *Proceedings of the FOCS, 88.*

[Cl] Clarkson K, Applications of random sampling to computational geometry, II, *Proc. 4th Ann. Sympos. Comput. Geom., 1988.*

[Edah] Edahiro M., Kokubo I., and Asano T., A new point location algorithm and its practical efficiency—comparison with existing algorithm, *ACM Transactions on Graphics* 3(2), 1984.

[Edel] Edelsbrunner H., Guibas L., Stolfi J., Optimal point location in monotone subdivisions, *SIAM J. Computing, vol. 15, no.2, pp. 317-340,1986.*

[Guib] Guibas L. and Seidel R., Computing convolutions by reciprocal search, *Discrete Comput. Geom.* 2: 175–193 (1987).

[Kirpat] Kirpatrik D., Optimal search in planar subdivisions, *SIAM J. Comput.* 12(1), 1983.

[Lipton] Lipton R. and Tarjan R., Applications of a planar separator theorem, *Proceedings of the FOCS, 77.*

[Mul1] Mulmuley K., A fast planar partition algorithm, I, *Proceedings of the 29th FOCS, 1988, full version to appear in a special computational geomtery issue of the Journal of Symb. Logic.*

[Mul2] Mulmuley K., On levels in arrangements and Voronoi diagrams, *Technical report, TR 88-21, University of Chicago, December, 88.*

[Mul3] Mulmuley K., An efficient hidden surface removal algorithm, *manuscript.*

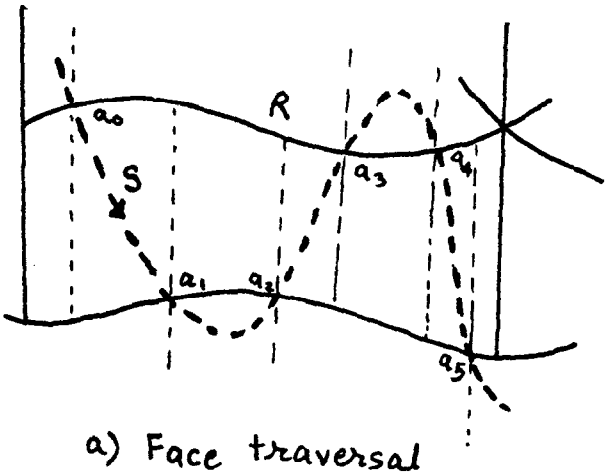
[Mul4] Mulmuley K., A fast planar partition algorithm, II, *complete manuscript, submitted to JACM.*

[Prep] Preparata, F. and M. Shamos, *Computational Geometry, An Introduction.* Springer-Verlag, 1985.

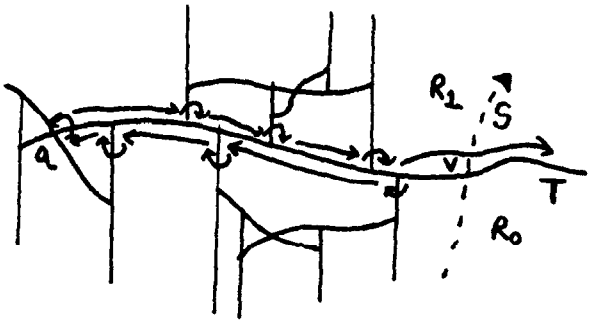
[Sarnak] Sarnak, N., Tarjan, R., Planar point location using persistent search trees, *Communications ACM, vol. 27, no. 7, pp. 669-679, 1986.*

[Suther] Sutherland, I. E., R. F. Sproull, and R. A. Schumaker, A characterization of ten hidden surface algorithms, *Computing Surveys* 6: 1–55, 1974.

[Waerden] Van der Waerden B. L., *Algebra,* v. 1. Frederic Ungar Publishing Co.



a) Face traversal



b) Face transition

Fig 2

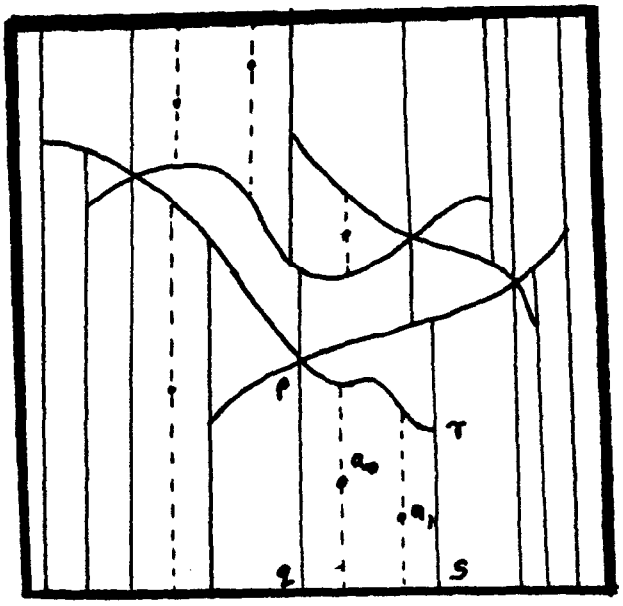


Fig 1 : G_4

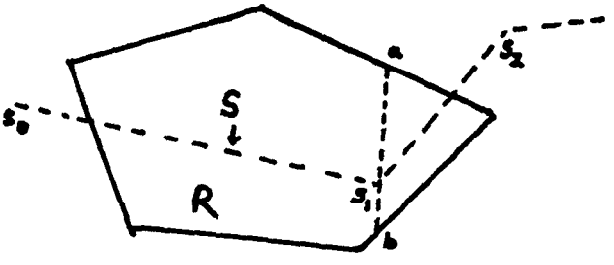


Fig 3a

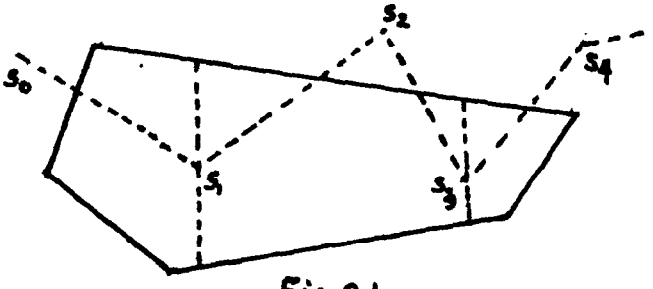
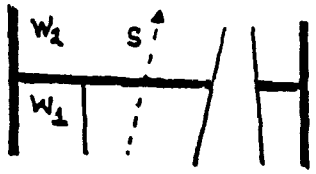
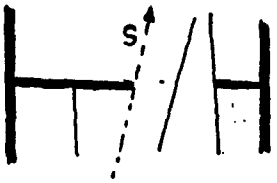


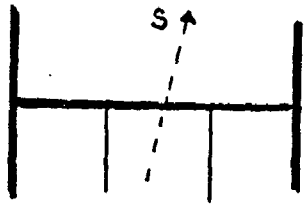
Fig 3b



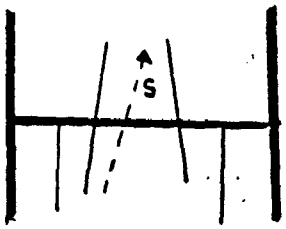
a) Before Contraction



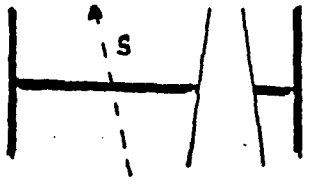
b) After contraction



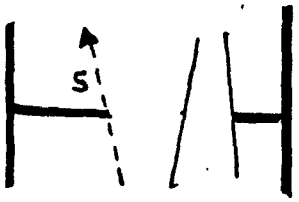
c) No contraction is carried out



d) This situation cannot arise



e) Before contraction



f) After contraction

Fig 5 (Window borders are thick)

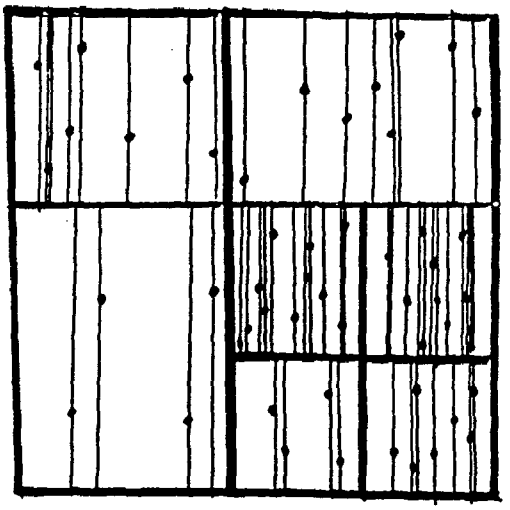
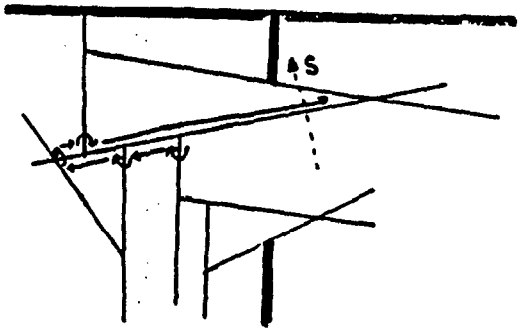


Fig 4 : Go



Straying during face transition

Fig 7

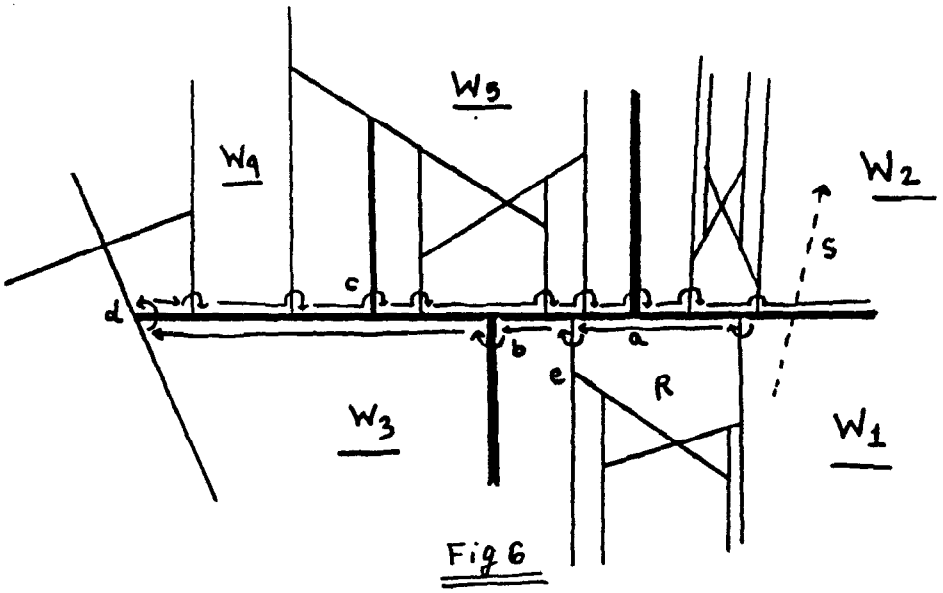


Fig 6