COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 28, 92–108 (1984)

Some Methods of Computational Geometry Applied to Computer Graphics*

H. EDELSBRUNNER

Institutes for Information Processing, Technical University of Graz, Schiesstattgasse 4a, A-8010 Graz, Austria

AND

M. H. OVERMARS

Department of Computer Science, University of Utrecht, P.O. Box 80.002, 3508 TA Utrecht, The Netherlands

AND

R. SEIDEL

Institutes for Information Processing, Technical University of Graz, Schiesstattgasse 4a, A-8010 Graz, Austria

Received October 17, 1983; revised February 1, 1984

Windowing a two-dimensional picture means to determine those line segments of the picture that are visible through an axis-parallel window. A study of some algorithmic problems involved in windowing a picture is offered. Some methods from computational geometry are exploited to store the picture in a computer such that (1) those line segments inside or partially inside of a window can be determined efficiently, and (2) the set of those line segments can be maintained efficiently while the window is moved parallel to a coordinate axis and/or it is enlarged or reduced. © 1984 Academic Press, Inc.

1. INTRODUCTION

In recent years, the interest in processing graphical data with the computer has increased enormously. This is manifested by the existence of scientific journals whose primary interests are in computer graphics. In applications involving high quality displays, astronomic amounts of data have to be processed. This situation begs for efficiency of algorithms and data structures carrying out this computation (see, e.g., Newman and Sproull [19]). It is the belief of the authors that the study of geometric problems as is common in the area of computational geometry is one source of efficient computational methods for manipulating graphical data. For support of this thesis we refer to van Leeuwen [26] who identified a number of results and methods of computational geometry which are highly relevant for problems in computer graphics.

The area of computational geometry found its first systematic manifestation in the doctoral thesis of Shamos [25] who defines it as the study of the computational complexity of geometric problems. A recent bibliography of Edelsbrunner and

*Research reported in this paper was done while the second author visited the Technical University of Graz. He was supported by the Netherlands Organization for the Advancement of Pure Research (ZWO). The first author was supported by the Austrian Fonds zur Foerderung der wissenschaftlichen Forschung.

0734-189X/84 $3.00

van Leeuwen compiling the publications in this rapidly expanding area to mid-1982 contains some 650 entries [11].

The aim of this paper is to continue the attempt of van Leeuwen [26] in making explicit applications of results and methods of computational geometry to computer graphics. In particular, we investigate algorithmic problems involved in windowing a two-dimensional picture. Our methods apply to pictures composed of mutually nonintersecting open line segments and their endpoints. When closed line segments or intersecting line segments occur then the picture can easily be refined so that it falls into this class. We describe space-efficient data structures that allow us to identify efficiently those line segments of a given picture visible in a specified axis-parallel rectangular window. Furthermore, strategies are developed to maintain the set of visible line segments when the window is moved, enlarged, or reduced.

The tools of computational geometry that will be used in our approach are the rather general paradigms called the locus approach and the plane-sweep technique. Furthermore, solutions for locating a point in a planar subdivision, walking in planar subdivisions, and determining points in an axis-parallel rectangle are applied. Those tools are now briefly explained and references to the relevant literature are given.

The *locus approach* is applicable to search problems and is the very general idea of interpreting the query objects as points in some space and partitioning this space into nonoverlapping domains such that the answer is invariant for all points falling into one and the same domain (see Overmars [22]). Thus, the domain a query point falls in determines the answer. The locus approach leads naturally to the *point location search problem* that aims at data structures and algorithms for determining the domain in which a given query point falls. In particular the planar version of this problem has attracted considerable attention. Dobkin and Lipton [6] were the first to give a data structure such that a planar point location query can be answered in $O(\log n)$ time, n denoting the number of edges of the subdivision. The $O(n^2)$ space requirements of their solution were reduced to $O(n)$ by an ingenious method in Lipton and Tarjan [18] which retains the optimal time for answering a query. Solutions which are more attractive for practical applications were given later by Lee and Preparata [17], Preparata [24], Edelsbrunner and Maurer [10], Kirkpatrick [15], and Edelsbrunner, Guibas, and Stolfi [8]. While the first three solutions are slightly suboptimal in their requirements, the fourth has the disadvantage of not generalizing to subdivisions with nonstraight edges. In fact, the last solution is the only one optimal for nonstraight subdivisions.

Beside locating a point in a subdivision, it is of interest to find efficient strategies for walking in subdivisions. In particular, walking strategies that take constant time to go to the "next region of interest" are desired. In a recent advance, Chazelle [4] developed a method that modifies subdivisions to this end without increasing the asymptotic space bound.

The *plane-sweep technique* is again a rather general idea used in Hadwiger [14] to answer certain geometrical questions and in Nievergelt and Preparata [20] to construct a planar subdivision determined by a self-intersecting polygon. The essence of the idea is to sweep the plane with a unidirected line in some fixed direction. In algorithmic applications, all computation is performed in the, in some sense, local neighbourhood of the line. The computation is most often supported by certain data structures storing the elements currently intersecting the sweeping line.

In addition to those tools, solutions for the planar *range search problem* are used. The problem involves points in the plane as objects and axis-parallel rectangles as query objects. The points are to be stored such that those inside of a query rectangle can be determined efficiently. A survey of solutions for the range search problem can be found in Bentley and Friedman [2]. The most efficient solutions for the planar version are described in Willard [27] and in Edelsbrunner [7]. Both solutions require $O(n \log n)$ space to store n points and $O(\log n + t)$ time to find the t points inside of a query rectangle.

The paper is organized as follows: In Section 2, solutions for two search problems are presented. It is worthwhile mentioning that the results of this section imply a new solution for the planar range search problem. Section 3 discusses the application of these solutions to windowing a picture and moving a fixed-size window. The more general problem of windowing and moving with arbitrarily sized windows is considered in Section 4. In this environment it is also possible to apply a size-changing operation to the window, usually called zooming in computer graphics. For the sake of efficiency, the shape of the window, that is, the ratio of height over width, is required to be fixed. Finally, Section 5 reviews the main contributions of this paper and discusses extensions of the presented material.

2. TWO SEARCH PROBLEMS

Before proceeding to the primary concern of this paper, windowing a two-dimensional picture, we consider two particular search problems. Windowing is then performed using the solutions for these search problems as primitives. To be clear in presentation, we start with the introduction of some concepts.

A line segment s is called *closed* if it contains its endpoints and *open* if it contains neither endpoint. Two line segments are said to *intersect* if they have a point in common. If this point is in the relative interior of both then they *intersect properly*, and they *touch*, otherwise.

Let S be a set of m closed line segments in the plane. S may also contain points which are considered to be degenerate closed line segments. We define a set PICT that contains points and open line segments as follows: All endpoints of line segments in S are in PICT. Also if two line segments intersect properly in a single point then this point is in PICT. The open line segments in PICT are the maximal subsets of the union of S minus the points in PICT. We call PICT the *picture of* S or simply a *picture*. Observe that the transformation of S into its picture eliminates intersections of any kind: (1) if two line segments overlap then this overlap is cut off the two line segments and introduced as a line segment of its own; (2) if two line segments intersect but do not overlap then they are cut into nonintersecting pieces; (3) the endpoints are separated from the line segments to get rid of nonproper intersections. For the sake of a simplified exposition, we will not distinguish between the picture of a set of closed line segments and the set itself. Fig. 2.1 shows the picture of a set of not properly intersecting closed line segments.

Computationally, PICT can be obtained from S in $O((m + t)\log m)$ time, t denoting the number of intersecting pairs in S (see Bentley and Ottmann [3]). If the line segments in S do not properly intersect then $t \leq 2m$ and $5m$ is an upper bound on the number of elements in PICT.

Let now q denote an arbitrary point in the plane. The *next-element of* q *in* PICT is the unique element e in PICT whose intersection with the open horizontal ray

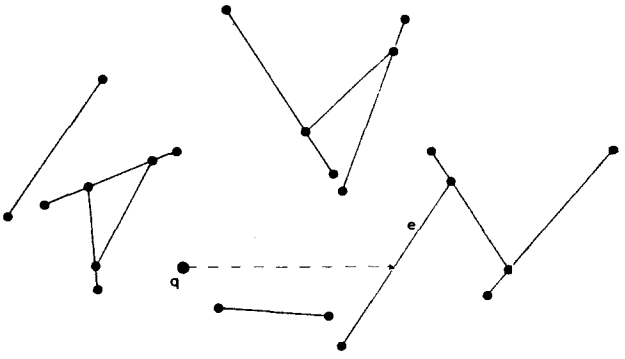

FIG. 2.1. Picture and query point q.

emanating from q towards the right is leftmost (see Fig. 2.1). If q is moved horizontally towards the right then the next-element is the first one encountered. The *next-element search problem* requires storing PICT such that the next-element of a query *point* q can be determined efficiently. Observe that the restriction of moving horizontally to the right is no loss of generality as no direction is distinguished in PICT.

The problem is solved with the aid of the locus approach: For each element e of PICT the domain D_e of the plane is constructed such that e is the next-element of a point q if and only if q is in D_e. The domain may be a two-dimensional or a one-dimensional object. In degenerate cases, it may even be empty. Given a point q, the next-element of q is detected by determining the domain that contains q.

The stated requirements uniquely define the desired subdivision of the plane: It is obtained by drawing a horizontal ray for each point p in PICT that leaves p to the left until it hits another element of PICT. Ignoring the dashed lines, Fig. 2.2 displays the described subdivision of the picture shown in Fig. 2.1.

Unless a point is the right endpoint of a horizontal line segment, its domain is a horizontal edge that is closed or unbounded to the left and open to the right. The domain of a nonhorizontal line segment s in PICT is a polygonal region R. R is open to the right where it is bounded by s. Furthermore, R intersects a horizontal line H if and only if s intersects H. In this case, R intersects H in a single interval closed or unbounded to the left. The domain of a horizontal line segment s is the smallest edge closed to the left and open to the right that covers s. The *next-element subdivision* of PICT contains as *faces* the interiors of the various polygonal regions and, in addition, the interior of the complement as the *outer face*. An *edge* is a maximal straight and open subset of the intersection of the closures of two domains. A *vertex* is an endpoint of an edge. Note that each face, edge, and vertex belongs to a unique domain D in the sense that it is contained in D. It is readily seen that there are at most $2n$ edges, $2n$ vertices, and n faces in the subdivision, if n is the cardinality of PICT. Thus, the *size* of the next-element subdivision of PICT is in $O(n)$.

For the construction of the next-element subdivision of PICT, we employ the so called plane-sweep technique (see, e.g., Nievergelt and Preparata [20]). Conceptually, a horizontal line H is swept bottom-up and some activities are performed each time H meets a point in PICT. Actually, the sweep is carried out by scanning the

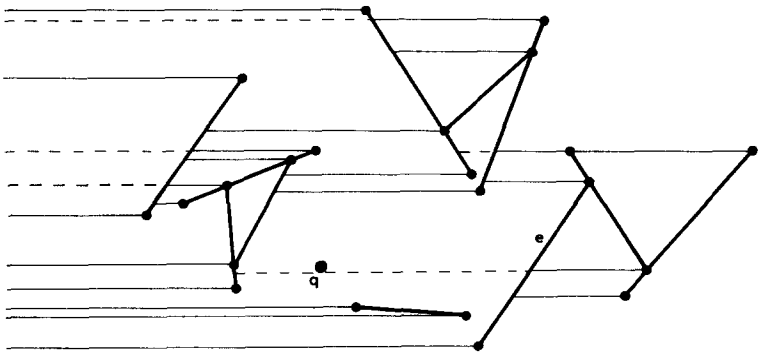

FIG. 2.2. Next-element subdivision.

vertically sorted list of the points. Consequently, the first action the algorithm takes is to sort the points with respect to y coordinates. See Knuth [16] or Aho, Hopcroft, and Ullman [1] for efficient sorting algorithms. When ties occur then the order of the points is immaterial. While H sweeps bottom-up, an initially empty dictionary X is maintained that stores in horizontal order the elements of PICT currently intersecting H. See Knuth [16] or Aho, Hopcroft, and Ullman [1] for efficient implementations of a dictionary. During the algorithm vertices, edges, and faces of the subdivision are created and various incidences between them are established. We do not describe these activities in detail and refer to [17], [15], and [8] for information on which incidence relations might be computationally useful. Now, a more detailed description of the plane-sweep algorithm is provided:

Algorithm NEXT-ELEMENT SUBDIVISION:

When H encounters a new set P of points (all having the same y-coordinate) then the following three steps are performed:

(1) Each point p of P is inserted into X and the nonhorizontal line segments having those points as upper endpoints are deleted from X. The horizontal line segments whose endpoints are in P are also inserted into X. These activities give rise to the completion of several faces of the subdivision.

(2) For each point p in P, the element e in X immediately to the left of p is determined (if it exists). The horizontal edge connecting e and p (resp. the horizontal edge unbounded to the left if e does not exist) is created in the subdivision. If e is a nonhorizontal line segment then the creation of the horizontal edge connecting p and e gives rise to the creation of a vertex splitting e into two edges. The new edges give rise to the opening, completion, and continuation of faces.

(3) Each point p of P is deleted from X, the horizontal line segments are deleted from X, and the nonhorizontal line segments which have points in P as lower endpoints are inserted into X. This gives rise to the opening of new faces to be completed later.

THEOREM 2.1. *Let* PICT *be a picture with a total of n points and open line segments. There exists a data structure for* PICT *that requires* $O(n)$ *space and*

$O(n \log n)$ time for its construction such that the next-element of a query point can be determined in $O(\log n)$ time.

Proof. The data structure realizing the asserted bounds is the next-element subdivision of PICT with some superimposed structure. The time required to construct the next-element subdivision is $O(n \log n)$ for the initial sorting, $O(\log n)$ per point and line segment for the manipulation of X, and time proportional to the size of the subdivision to set up all incidences. As the size of the subdivision is proportional to n, we conclude that $O(n \log n)$ time and $O(n)$ space suffices to create the next-element subdivision.

To achieve the $O(\log n)$ time bound to answer a query, we make use of optimal solutions for the planar point location search problem. Any of the data structures in Kirkpatrick [15] and Edelsbrunner, Guibas, and Stolfi [8] is appropriate. Their results imply: For a subdivision of the plane with m straight edges there exists a data structure that requires $O(m)$ space and $O(m \log m)$ time for construction such that the face (or edge or vertex) a given query point falls in (or on) can be determined in $O(\log m)$ time. This proves the assertion and completes the argument.

The solution for the next-element search problem can be used to determine the "next few" elements encountered by a query point q that moves horizontally to the right. More specifically, we are interested in those elements that intersect a horizontal line segment h (the way traced by q). To this end, the following strategy can be applied:

> Initially, q is the left endpoint of h.
> While q is not to the right of the right endpoint of h, the following actions are taken: The next-element e of q is determined. Unless e is a horizontal line segment let q be the intersection of e and the horizontal line supporting h. If e is a horizontal line segment, then q is the right endpoint of e.

If t elements intersects h then $O((t + 1)\log n)$ time is required to detect them. However, we can do better as there is an inherent inefficiency involved in the algorithm above: At each step, a next-element query is invoked although some extra information is available. This extra information is the element detected at one step earlier.

In fact, Chazelle [5] independently developed next-element subdivisions for finding intersections of query line segments and also found a method that exploits the extra information mentioned above. For the sake of completeness, we explain his method which relies on a modification of the subdivision used. Observe that the algorithm above can be viewed as letting q walk in the next-element subdivision. This walk starts at the left endpoint of h and ends at its right endpoint. The essential difficulty in performing this walk within the subdivision is to cross original line segments that bound faces to the right. In general, such a line segment s is cut into a number of edges and vertices in the subdivision. In order to determine the face (or edge) immediately to the right of s that intersects h, we may use binary search to find the edge (or vertex) on s that intersects h. However, this strategy requires asymptotically the same amount of time as repeated point location search.

To remedy this shortcoming, new horizontal edges are introduced which refine the next-element subdivision. Let s be a nonhorizontal line segment that is cut into a number of edges and vertices in the subdivision. Let $v_0, \ldots, v_{k+1}$ be the bottom-up

sequence of vertices on s. Clearly, v_0 and v_{k+1} are the endpoints of s, and $v_1, \ldots, v_k$ are created by horizontal edges to the right of s (see Fig. 2.2). A new horizontal edge is introduced for each v_i on s, with $1 \leq i \leq k$ and i even. This edge leaves v_i towards the left until it hits another edge of the subdivision. If no edge is hit then the new edge is unbounded to the left. The dashed edges in Fig. 2.2 refine the depicted next-element subdivision. Note that a new edge gives rise to a new vertex on the edge hit (if it exists). So the computation of the new edges has to be carried out from right to left in order to achieve the following crucial property:

Property 2.2. In the refined next-element subdivision, each face has at most two right bounding edges.

Due to Property 2.2, it is now possible to walk horizontally from left to right with constant time per line segment encountered. Furthermore, the modification of the next-element subdivision does not increase its asymptotic size, that is, the number of new edges is in $O(n)$ [4]. For an algorithm that computes the refinement in $O(n \log n)$ time we refer also to [4]. This yields:

THEOREM 2.3. *Let* PICT *be a picture with a total of n points and line segments. There exists a data structure that requires $O(n)$ space and $O(n \log n)$ time for its construction such that the t elements in* PICT *that intersect a horizontal query line segment can be determined in $O(\log n + t)$ time.*

A problem very similar to the next-element search problem is what we call the next-point search problem: Let S denote a finite set of points in the plane and let q denote a vertical and closed line segment of unit length. A point p in S is called a *next-point of* q if its x coordinate is minimal under the constraints that (i) the x coordinate of p is greater than the one of q, and (ii) the horizontal line through p intersects q. In other words, p is a point that is hit first when q is moved horizontally to the right (see Fig. 2.3). In general, q has more than only one next-point. The *next-point search problem* requires storing S such that the next-points of a vertical and closed query line segment of unit length can be determined efficiently. The restriction to vertical closed line segments of unit length and to moving it horizontally to the right is no loss of generality.

To achieve a solution, the next-point search problem is transformed into a special instance of the next-element search problem: Each point p of S is transformed into

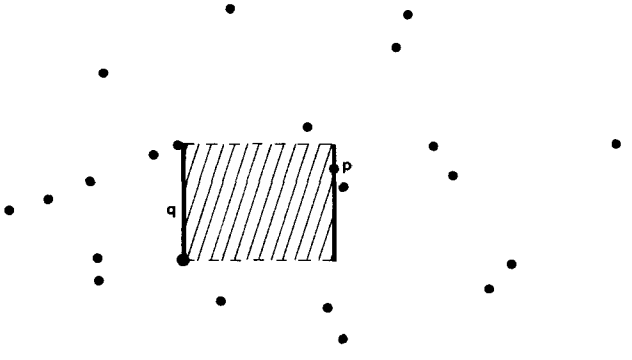

FIG. 2.3. Next-point of q.

the vertical and closed line segment $T(p)$ of unit length whose upper endpoint is p. A query line segment q is transformed into its lower endpoint $T(q)$. See Fig. 2.4 which displays the transformed scene shown in Fig. 2.3.

For the time being, let us assume that no two points of S lie on a common vertical line. Easy geometric observations show that the next-point p of a query line segment q corresponds to the next-element of $T(q)$. Also there is no next-point for q if and only if there is no next-element for $T(q)$. The special instance of the next-element search problem is solved as described above. The particular subdivision obtained is called the *next-point subdivision* of S.

We now come back to the general case, that is, there may be points in S that lie on a common vertical line. As a consequence, there may be overlapping line segments in the transformed set. Let v_1 denote a line segment that overlaps the line segments $v_2, \ldots, v_i$ such that their upper endpoints are below the one of v_1. Let p denote the topmost upper endpoint of the line segments $v_2, \ldots, v_i$. Then the open part below p is cut off v_1. This strategy guarantees that our method, as it is now, yields the bottommost next-point for a query line segment q. Constant time suffices for each additional next-point of q if the points of S are stored lexicographically ordered in a list L. If there are additional next-points then they follow p in L. The construction of L can be achieved without affecting the asymptotic bounds for the time required to set up the next-point subdivision. This yields:

THEOREM 2.4. *Let S denote a set of n points in the plane. There exists a data structure that requires $O(n)$ space and $O(n \log n)$ time for its construction such that the t next-points of a vertical and closed query line segment of unit length can be determined in $O(\log n + t)$ time.*

Modifying the next-point subdivision (it is a next-element subdivision after all) as explained above allows walking horizontally to the right with constant time per vertical line segment encountered. This yields:

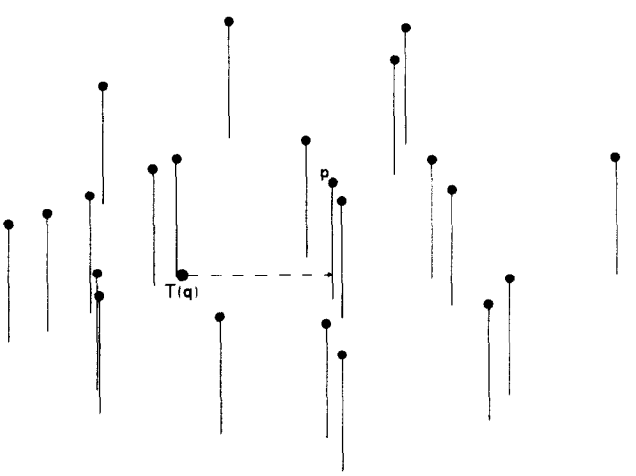

FIG. 2.4. Point to line segment transform.

THEOREM 2.5. *Let S be a set of n points in the plane. There exists a data structure that requires $O(n)$ space and $O(n \log n)$ time for its construction such that $O(\log n + t)$ time suffices to determine the t points contained in an axis-parallel rectangle with unit height.*

3. WINDOWING A SET OF LINE SEGMENTS

The two search problems introduced in Section 2 provide the tools for our approach to the central problem of this paper, namely to what is called windowing in computer graphics, and to performing certain operations on the window.

A *window* is an axis-parallel rectangle, that is, the Cartesian product of two closed intervals, one on each of the two coordinate axes. For the time being, we consider only windows of unit size, that is, the window has unit height and unit width. This restriction will be removed in Section 4. The *visibility set* of a window w is the set of elements in the picture PICT that have a point in common with w. Figure 3.1 shows a typical picture which represents a two-dimensional projection of a three-dimensional polyhedral scene. The line segments in the visibility set of the displayed window are drawn heavily. The *windowing search problem* requires storing PICT such that the visibility set of a given window can be determined efficiently.

THEOREM 3.1. *Let* PICT *denote a picture with a total of n open line segments and points. There exists a data structure that requires $O(n)$ space and $O(n \log n)$ time for its construction such that the t elements of* PICT *in the visibility set of a query window of unit size can be determined in $O(\log n + t)$ time.*

Proof. The visibility set of a query window w is computed in two parts: The line segments that intersect the boundary of w and the points on the boundary of w are determined by posing intersection queries with the four line segments that comprise the boundary. The line segments and points that are completely contained in w are computed by determining all points inside of w. This suffices since each line segment inside of w has its endpoints inside of w. Both parts can be performed within the asserted bounds (see Theorems 2.3 and 2.5). This completes the argument.

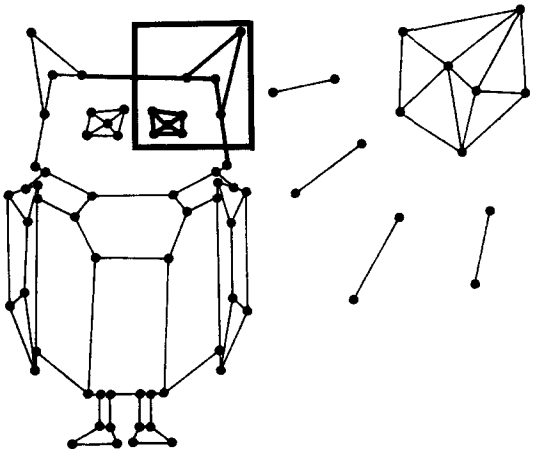

FIG. 3.1. Picture with window.

Beside determining the elements visible in a window, it is often useful to move the window in order to get the desired part of the picture onto the screen. An easy strategy to get a different part of the picture onto the screen is to redefine the window and to compute the visibility list again. This strategy is obviously inefficient if the window is moved only slightly and the visibility list changes little. In such cases the data structure introduced above for solving the windowing search problem permits a better method. Observe that the visible part of some partially visible line segments, of course, changes continuously when w is moved. We do not consider those changes but only changes in the visibility set.

Let PICT denote a picture with n open line segments and points, and let V denote the visibility set of a given window w. We need to maintain V while moving w parallel to a coordinate axis. The *moving search problem* asks for a data structure for storing PICT such that the elements of PICT that are involved in the first change of V as w is moved parallel to a coordinate axis can be determined efficiently. More specifically, the first line segments or points that have to be inserted into V or deleted from V are desired (see Fig. 3.2).

THEOREM 3.2. *Let* PICT *denote a picture with a total of n open line segments and points, let w be a window of unit size, and let V be the visibility set of w. There exists a data structure that requires* $O(n)$ *space and* $O(n \log n)$ *time for its construction such that* $O(\log n + t)$ *time suffices to determine the t elements of* PICT *involved in the first change of V when w is moved parallel to a coordinate axis.*

Proof. Without loss of generality, we restrict our attention to moving w horizontally to the right. Three subdivisions of the plane are used to detect the first elements of PICT that become visible in w when w is moved: (1) A next-element subdivision for the line segments with negative slope and their endpoints, (2) a next-element subdivision for the line segments with positive slope (including vertical ones) and their endpoints, and (3) a next-point subdivision for the points of PICT. A line segment or point that is among the first elements which become visible in w is at least one of four types:

(i) it is the next-element of the upper right corner of w and has negative slope

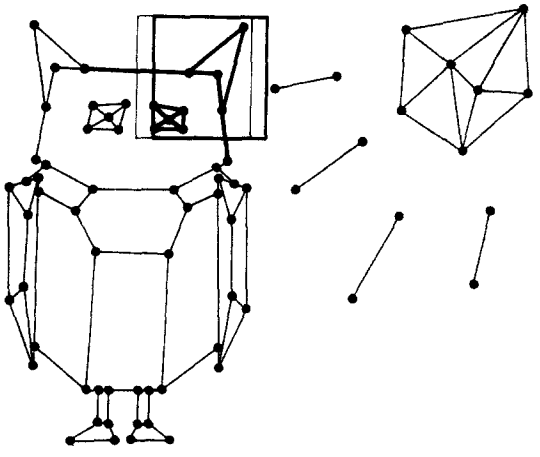

FIG. 3.2. Moving the window to the right.

if it is a line segment;

(ii) it is the next-element of the lower right corner and has positive slope or is vertical if it is a line segment;

(iii) it is a next-point of the right border of w; or

(iv) it is a line segment whose left endpoint lies on the right border of w before w is moved.

Case (iv) is trivial if the points on the right border of w that are left endpoints of some line segments are kept in a separate list. Case (i) can be checked in $O(\log n)$ time using the point location techniques described in Sections 1 and 2. Case (ii) is analogous to case (i). A difficulty arises in case (iii) if there are several next-points of the right border of w. It is expensive to determine those points if they appear not to be among those elements involved in the first change of V. This situation can be controlled by determining only one of those points first (which costs $O(\log n)$ time by methods analogous to those taken in cases (i) and (ii)) and determining the others only if this one is involved in the first change of V.

The first elements of S that fall out of w can be detected by analogous techniques. The actual elements are determined from the eight cases considered, which completes the argument.

4. ARBITRARY-SIZE WINDOWING AND ZOOMING

This section develops methods for windowing and moving with axis-parallel windows of arbitrary size. Furthermore, an operation that changes the size but not the shape of the window (that is, the ratio of height over width is invariant) is investigated. This operation is usually called *zooming*. Efficient data structures are developed that allow for windowing, moving the window vertically or horizontally, and zooming. The gain of additional generality is paid for by more space required by the data structures.

Our approach to this larger collection of problems is the same as in Section 2: We use as primitives for our solutions next-element and next-point search together with a solution for the so-called ru-point search problem to be introduced below.

Note that the solution for the next-point search problem given in Section 2 is restricted to query lines of unit length. We no longer are able to stick to this simplification since the size of the window, and thus the length of its edges, may change by application of zooming. Minor modifications of the search strategies in the layered range tree of Willard [27] or the RT-tree of Edelsbrunner [7] yield:

THEOREM 4.1 [27, 7]. *Let S denote a set of n points in the plane. There exists a data structure that requires $O(n \log n)$ space and time for its construction such that the t_1 next-points of a vertical query line segment of arbitrary length can be determined in $O(\log n + t_1)$ time. In addition, $O(\log n + t_2)$ time suffices to find the t_2 points in an axis-parallel query rectangle.*

Together with Theorems 2.1 and 2.3, Theorem 4.1, immediately implies solutions for the windowing and the moving search problem, now without restriction on the size of the window.

COROLLARY 4.2. *Let* PICT *denote a picture of n open line segments and points There exists a data structure that requires $O(n \log n)$ space and $O(n \log n)$ time for its*

construction such that (1) *the t_1 elements of S in the visibility set of a query window w (of arbitrary size) can be determined in $O(\log n + t_1)$ time, and* (2) *the t_2 elements of S involved in the first change of the visibility set of w as w is moved parallel to a coordinate axis can be determined in $O(\log n + t_2)$ time.*

It is worthwhile to mention that the authors believe that there is a more clever way to layer the layered range tree in [27] such that moving the window can be done in $O(\log\log n + t_2)$ time. For shortage of space, the details of this idea involving the use of priority queues are omitted.

So far only extensions of the results of Sections 2 and 3 to windows of arbitrary size are given. The remainder of this section is devoted to the examination of the zooming operation. A more basic search problem is considered first which will serve as a primitive for zooming later.

Let S denote a set of n points in the plane. A point $p = (p_x, p_y)$ of S is called a *right-up-point* (*ru-point* for short) of a query point $q = (q_x, q_y)$ if p is a point of S with minimal x coordinate such that $p_x - q_x \geq p_y - q_y \geq 0$. In other words: imagine moving a vertical line segment from q to the right whose lower and upper endpoint lie on the lines through q of slope 0 and 1, respectively. The first points of S that are hit by this line segment are the ru-points of q (see Fig. 4.1). Similarly, left-up-points, up-right-points, right-down-points etc. can be defined.

An *ru-point query* consists of a query point q and asks for all ru-points of q. The *ru-point search problem* requires storing S such that ru-point queries can be answered efficiently.

THEOREM 4.3. *Let S be a set of n points in the plane. There is a data structure that requires $O(n)$ space and $O(n \log n)$ time for its construction such that the t ru-points of a query point can be determined in $O(\log n + t)$ time.*

Proof. Like Theorem 2.1 we prove this one by use of the locus approach. We partition the plane into $n + 1$ polygonal domains of "equal answer." One domain comprises exactly all points that do not have an ru-point and for each p in S there is a domain D_p comprising exactly all points in the plane for which p is the bottommost ru-point. These domains determine a subdivision consisting of faces (interiors of domains), edges (relative interiors of the intersection of two closed domains), and vertices (endpoints of edges).

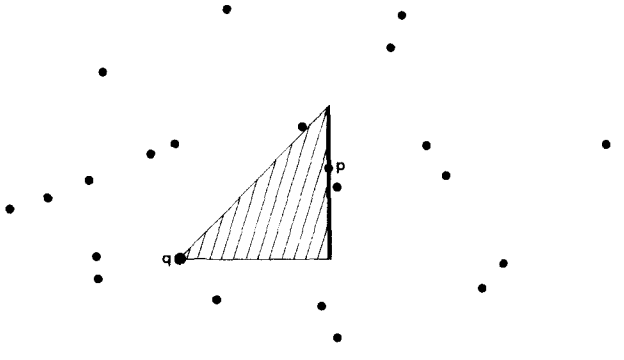

FIG. 4.1. The right-up-point of q.

With such a subdivision an ru-point query can be answered easily: Use the method in [15] or [8] to determine in $O(\log n)$ time in (or on) which face (or edge or vertex) of the subdivision a query point $q = (q_x, q_y)$ lies. This face (or edge or vertex) uniquely determines the domain that contains q. If q lies in D_p for some p in S then p is the ru-point with smallest y coordinate. If there are other ru-points for q, then they must have the same x coordinate as p and their y coordinates are greater that p's but not greater than $q_y + (p_x - q_x)$. To determine such points keep the points of S lexicographically ordered in a list L and report all points in L immediately following p whose x coordinates agree with p's and whose y coordinates are not too large. This clearly takes constant effort per point reported and thus the t ru-points of a query point can be determined in $O(\log n + t)$ time.

It remains to show how the subdivision described above can be constructed in $O(n \log n)$ time and $O(n)$ space. For a point z in the plane let W_z denote the set of points different from z lying between or on the rays of slope 0 and 1 emanating from z to the left. It is easy to see that for a point p in S its associated region D_p is W_p without those parts that belong to some W_z, for z in S and z lexicographically less than p.

Thus, we can construct the required subdivision in the following way: First sort the points of S into lexicographically increasing order (can be done in $O(n \log n)$ time) and then, incrementally for $k = 1$ to n, add the region of the kth point p to the already constructed regions of the first $k - 1$ points. Intuitively, this is done by following the rays bounding W_p until they intersect the boundary of the union of the first $k - 1$ regions (see Fig. 4.2). Algorithmically this is done as follows: Maintain a balanced tree T that stores sorted by y coordinates the points of S that lie on the boundary of the subdivision constructed so far. (This clearly can be done in $O(n \log n)$ time overall.) From T, the boundary edge that is intersected by the horizontal ray leaving p can be determined in $O(\log n)$ time. From this intersection point follow the boundary downwards (and delete from T all points of S encountered) until the edge is found that is intersected by the other ray leaving p. The time to do this (exclusive of the manipulation of T) is proportional to the number of edges traversed. However, each edge traversed ceases to be a boundary edge and cannot be traversed again. Therefore, the accumulative effort over all points of S is proportional to the number of edges in the entire subdivision.

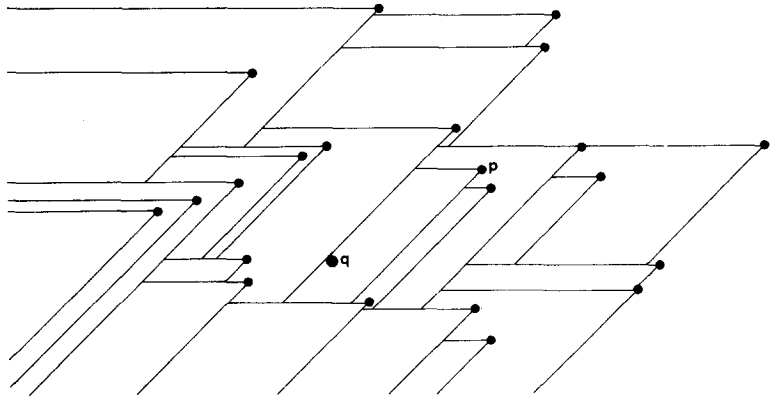

FIG. 4.2. ru-point subdivision.

But note that this is $O(n)$ as for each domain D_p added the number of vertices and edges in the subdivision increases at most by three and four, respectively. Hence, the entire subdivision cannot have more than $3n$ vertices and $4n$ edges. This completes the argument.

It is not clear whether the ru-point subdivision can be modified such that it allows for walking in constant time per ru-point encountered. Such a modification would be a first step to improve the logarithmic time bound for zooming (see Theorem 4.4).

Now we have the tools available for solving the central issue of this section, viz, performing the zooming operation on the window. In computer graphics this operation is used to display some part of the picture with varying size. Larger size is achieved by enlarging the needed part on the screen which goes along with reducing the size of the window.

Let PICT denote a picture of n line segments and points, let w denote a window, and let V denote the visibility set of w. We restrict our attention to windows w of unit shape, that is, the height of w equals the width of w. The restriction to unit shape is no loss of generality since other rectangles can be transformed into squares by linear coordinate transformations. The *zooming search problem* asks that PICT be stored such that the elements of PICT that are involved in the first change of V as w is enlarged or reduced can be determined efficiently. The enlargement or reduction of w leaves the center of w invariant and changes the height and the width of w by the same scaling factor (see Fig. 4.3).

THEOREM 4.4. *Let* PICT *denote a picture with a total of n open line segments and points, let w be a window of unit shape, and let V be the visibility set of w. There exists a data structure that requires* $O(n \log n)$ *space and time for construction such that* $O(\log n + t)$ *time suffices to determine the t elements of* PICT *involved in the first change of V as w is enlarged or reduced.*

Proof. Let us consider the reduction of w first. The first elements of PICT that fall out of w are determined by moving the four edges of w inwards. If four

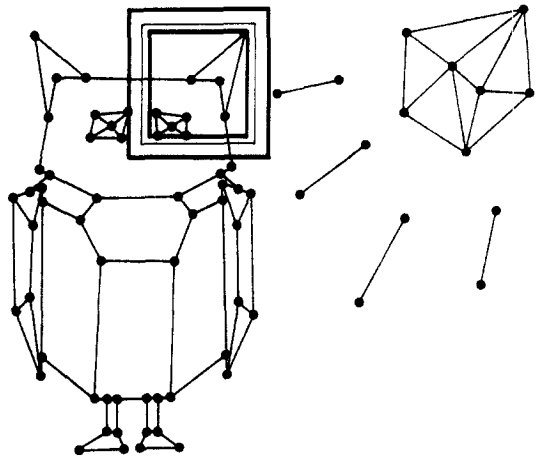

FIG. 4.3. Enlarging and reducing the window.

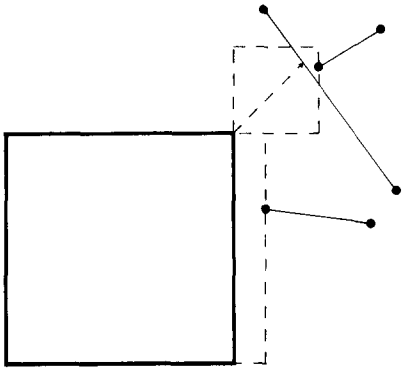

FIG. 4.4. Enlarging the window.

next-point subdivisions are available then the t first elements falling out of w can be determined in $O(\log n + t)$ time, see Theorem 4.1.

Next we examine the somewhat more complicated enlargement of w. One can distinguish three cases how an element e of PICT can be involved in the first change of V. Representative for the four borders and corners of w, the right border, and the top right corner of w are considered. The cases are:

(i) e is hit when the right border of w is moved to the right (see Fig. 4.4);

(ii) e is the next-element of the upper right corner of w which is moved upwards and to the right along the diagonal of w (see Fig. 4.4); or

(iii) e is an ru-point of the upper right corner of w (see Fig. 4.4).

The former two cases are treated as in Theorems 4.1 and 2.1. The third case is handled as described in Theorem 4.3. This completes the argument.

We close this section by noting that additional operations such as changing the size of the window by moving only one edge inwards or outwards can also be performed with the methods presented. This operation, however, changes the shape of the window and zooming with arbitrary shaped windows seems to be much harder than zooming with windows of fixed or unit shape.

5. DISCUSSION AND EXTENSIONS

We first give a review of the main contributions of this paper. Most importantly it describes the application of some methods and results of computational geometry to problems in computer graphics. Two-dimensional pictures made up of mutually nonintersecting open line segments and their endpoints are considered, e.g., the two-dimensional display of a three-dimensional polyhedral scene with hidden parts removed. We present methods that store a picture such that the set of line segments totally or partially inside an axis-parallel rectangular window can be determined efficiently, and this set can be maintained efficiently while the window is moved parallel to a coordinate axis or while it is enlarged or reduced.

Although emphasis has been laid on the application of tools to problems in computer graphics, it is worthwhile noting that our methods imply a new solution for a special case of the classical planar range search problem (see Theorem 2.5).

We close this section by mentioning extensions of the presented material in three directions: general instead of axis-parallel rectangular windows, general curves as objects instead of line segments only, and the possibility of changing the picture with little cost.

Search problems with polygonal query objects were investigated in a rather general setting in Edelsbrunner, Kirkpatrick, and Maurer [9] and for sets of points in Willard [28] and in Edelsbrunner and Welzl [12]. Their results indicate that this extension costs a great deal of the efficiency achieved for axis-parallel windows.

Fewer difficulties are to be expected if the line segments are replaced by arbitrary but computationally simple curves. What "computational simplicity" means depends on the primitive operations needed for the problem at hand. We note here that the only optimal solution for locating a point in a planar subdivision that contains nonstraight edges is the one in [8].

For performing changes in the picture at little cost, we refer to two general methods. One proceeds by splitting a data structure into a number of smaller instances that are independent of each other (see Overmars and van Leeuwen [23] which is one of the latest in a series of publications on this topic). The other method is based on constructing data structures by means of the divide-and-conquer paradigm, see Overmars [21] as well as Gowda and Kirkpatrick [13]. The applicability of their method follows from the possibility of constructing the data structures of this paper by means of this paradigm instead of the plane-sweep technique.

REFERENCES

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, Mass., 1974.
2. J. L. Bentley, and J. H. Friedman, Data structures for range searching, *ACM Comput. Surveys* **11** (1979), 397–409.
3. J. L. Bentley, and Th. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Trans. Comput.* **C-28** (1979), 643–647.
4. B. M. Chazelle, Filtering search: A New Approach to Query-answering. Proc. 24th Ann. IEEE Sympos. Found. Comput. Sci. (1983), pp. 122–132.
5. B. M. Chazelle, *Fast Computation of Segment Intersections*, Report CS-83-11, Department of Computer Science, Brown University, Providence, R. I., 1983.
6. D. P. Dobkin, and R. J. Lipton, Multidimensional searching problems, *SIAM J. Comput.* **5** (1976), 181–186.
7. H. Edelsbrunner, A note on dynamic range searching, *Bull. of the EATCS* **15** (1981), 34–40.
8. H. Edelsbrunner, L. J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, manuscript.
9. H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer, Polygonal intersection searching, *Inform. Process. Lett.* **14** (1982), 74–79.
10. H. Edelsbrunner, and H. A. Maurer, A space-optimal solution of general region location, *Theoret. Comput. Sci.* **16** (1981), 329–336.
11. H. Edelsbrunner, and J. van Leeuwen, *Multidimensional Data Structures and Algorithms: A Bibliography*, Report F105, Institutes for Information Processing, Technical University of Graz, 1982.
12. H. Edelsbrunner, and E. Welzl, *Halfplanar Range Search in Linear Space and $O(n^{0.695})$ Query Time*, Report F111, Institutes for Information Processing, Technical University of Graz, 1983.
13. I. G. Gowda, and D. G. Kirkpatrick, Exploiting Linear Merging and Extra Storage in the Maintenance of Fully Dynamic Geometric Data Structures, Proc. 18th Ann. Allerton Conf. on Commun. Control, & Comput. (1980), pp. 1–10.
14. H. Hadwiger, Eulers Charakteristik und kombinatorische Geometrie, J. Reine Angew. Math. **194** (1955), 101–110.
15. D. G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* **12** (1983), 28–35.

16. D. E. Knuth, *Sorting and Searching—The Art of Computer Programming* III, Addison-Wesley, Reading, Mass., 1973.
17. D. T. Lee, and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM J. Comput.* **6** (1977), 594–606.
18. R. J. Lipton, and R. E. Tarjan, Applications of a Planar Separator Theorem, Proc. 18th Ann. IEEE Sympos. on Found. Comput. Sci. (1977), pp. 162–170.
19. W. M. Newman, and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1979.
20. J. Nievergelt, and F. P. Preparata, Plane-sweep algorithms for intersecting geometric figures, *Comm. ACM* **25** (1982), 739–747.
21. M. H. Overmars, Dynamization of order decomposable set problems, *J. Algorithms* **2** (1981), 245–260.
22. M. H. Overmars, *The Locus Approach*, Report RUU-CS-83-12, Dept. Comput. Sci. University of Utrecht, 1983.
23. M. H. Overmars, and J. van Leeuwen, Worst-case optimal insertion and deletion methods for decomposable searching problems, *Inform. Process. Lett.* **12** (1981), 168–173.
24. F. P. Preparata, A new approach to planar point location, *SIAM J. Comput.* **10** (1981), 473–482.
25. M. I. Shamos, *Computational Geometry*, Ph.D. thesis, Dept. of Comput. Sci., Yale University, New Haven, Conn., 1978.
26. J. van Leeuwen, Graphics and Computational Geometry, Les Mathematiques de l'Informatique, Colloq. AFCET, Paris (1982), pp. 159–165.
27. D. E. Willard, New data structures for orthogonal queries, *SIAM J. Comput.*, in press.
28. D. E. Willard, Polygonal retrieval, *SIAM J. Comput.* **11** (1982), 149–165.