

ZOOMING BY REPEATED RANGE DETECTION

Herbert Edelsbrunner and Mark H. Overmars

RUU-CS-84-10  
october 1984



Rijksuniversiteit Utrecht

**Vakgroep informatica**

Budapestlaan 6 3584 CD Utrecht  
Corr. adres: Postbus 80.012 3508 TA Utrecht  
Telefoon 030-53  
The Netherlands

ZOOMING BY REPEATED RANGE DETECTION

Herbert Edelsbrunner and Mark H. Overmars

Technical Report RUU-CS-84-10  
october 1984

Department of Computer Science  
University of Utrecht  
P.O. Box 80.012  
3508 TA Utrecht  
the Netherlands

## ZOOMING BY REPEATED RANGE DETECTION

Herbert Edelsbrunner and Mark H. Overmars

**Abstract.** In a number of recent papers techniques from computational geometry (the field of algorithm design that deals with objects in multi-dimensional space) have been applied to some problems in the area of computer graphics. In this way, efficient solutions were obtained for the windowing problem that asks for those line segments in a planar set that lie in a given window (range) and the moving problem that asks for the first line segment that comes into the window when moving the window in some direction. In this paper we show that also the zooming problem, that asks for the first line segment that comes into the window when we enlarge it, can be solved efficiently. This is done by repeatedly performing range queries with ranges of varying sizes. The obtained structure is dynamic and yields a query time of  $O(\log^2 n)$  and an insertion and deletion time of  $O(\log^2 n)$ , where  $n$  is the number of line segments in the set. The amount of storage required is  $O(n \log n)$ . It is also shown that the technique of repeated range search can be used to solve several other problems efficiently.

**Keywords and phrases.** Windowing, moving, zooming, line segments, range searching, nearest neighbor searching.

### 1. Introduction.

Several recent papers [2,5,6,7] consider problems in computational geometry related to problems in computer graphics. These problems include the ones given in the following list. i) The *windowing problem* asks for those elements in a set of non-intersecting line segments in the plane which lie in a given rectangle with edges parallel to the axes (called a window or a range). It is a variant of the well-known range searching problem but in the windowing problem the objects are non-intersecting line segments rather than points. ii) The *moving problem* asks for the first line segment in a set that becomes (partially) visible in the window when the window moves parallel to one of the coordinate axes. iii) The *zooming problem* asks for the first line segment that becomes visible when the window is enlarged in a certain way (see Section 3 for a more precise definition of the problem). For both the windowing problem and the moving problem, efficient solutions are given in [2]. In the static case (i.e., the set of line segments is fixed) both problems were solved within a query time of  $O(k + \log n)$ , a preprocessing time of  $O(n \log n)$  using  $O(n \log n)$  storage, where  $n$  is the number of line segments in the set and  $k$  the number of segments that are visible in the window or become visible in the window at the same moment, respectively. In the dynamic case, both problems can be solved within a query time of  $O(k + \log^2 n)$ , an update time of  $O(\log^2 n)$  and a preprocessing time of  $O(n \log n)$  using  $O(n \log n)$  storage (see [5]). Unfortunately, the zooming problem was only partially solved.

---

#### Authors addresses:

Prof. H. Edelsbrunner, Department of Computer Science, University of Illinois, 1304 West Springfield Avenue, Urbana, Illinois 61801, U.S.A.

Dr. M.H. Overmars, Department of Computer Science, University of Utrecht, P.O.Box 80.012, 3508 TA Utrecht, the Netherlands.

In this paper we present an efficient dynamic solution to the zooming problem. This solution is based on a technique which finds the smallest window that contains a new visible line segment by repeatedly performing windowing queries with windows of varying sizes. We show how to choose the window sizes such that the process halts after a small number of iterations. This results in a data structure with a query time of  $O(\log^2 n)$ , an insertion and deletion time of  $O(\log^2 n)$  using  $O(n \log n)$  storage. The technique of solving one searching problem by repeatedly solving another searching problem might be useful for a number of other problems as well. For example, in Section 4 we show how the technique can be used to solve some variants of the nearest neighbor searching problem efficiently in a dynamic fashion.

## 2. Preliminary results.

In this section we recall some known results that will be used later on.

**Theorem 2.1.** *Given a set of  $n$  points in the plane there exists a structure which allows us to determine in  $O(\log n)$  time whether a given range is empty, contains one point or contains more than one point. The structure uses  $O(n \log n)$  storage and has  $O(\log^2 n)$  insertion and deletion time bounds.*

**Proof.** We use the structure described by Edelsbrunner[1], based on the priority search tree of McCreight[3]. Using  $O(n \log n)$  storage, the structure allows us to report the  $k$  points in a given query range in  $O(k + \log n)$  time. The  $O(\log n)$  bound follows if we stop the process after reporting the second point. □

A second result we need is about the so-called *shooting problem* : given a set of  $n$  non-intersecting line segments, we ask for the line segment that is hit first when we shoot from a given point  $p$  horizontally to the right.

**Theorem 2.2.** *Given a set of  $n$  non-intersecting line segments in the plane, the shooting problem can be answered in a query time of  $O(\log^2 n)$ . The structure uses  $O(n \log n)$  storage and has  $O(\log^2 n)$  update time bounds.*

**Proof.** Shooting is equivalent to moving an infinitely small window to the right. Hence, it can be solved using the structure for moving described in [5] that has the stated complexity. □

Clearly, the same result holds for shooting in another, prechosen direction.

## 3. The Zooming problem.

This section gives a dynamic data structure which solves the zooming problem, using the results described in the previous section. Let us first give a more formal definition of the zooming problem.

**Definition 3.1.** *The zooming search problem asks to store a set of non-intersecting but possibly touching line segments in the plane in such a way that, given a rectangle  $R$  with its sides parallel to the coordinate axes, we can efficiently determine the first structural change in the set of line segments that are (partially) visible in  $R$ , when the rectangle is enlarged or reduced. A structural change is one of the following cases: (i) A line segment that was not visible becomes partially visible, (ii) a partially visible line segment becomes totally visible, (iii) a line segment that intersects one boundary of  $R$  starts intersecting another boundary, (iv) a totally visible line segment becomes partially visible, (v) a partially visible line segment becomes invisible. Enlarging a rectangle means: leaving the center at the same place and increasing the width and the height by the same factor  $k > 1$ . Reducing a rectangle means: leaving the center at the same place and reducing the width and the height by the same factor  $k > 1$ .*

We will only consider the problem of enlarging the rectangle. The problem of reducing the rectangle can easily be solved by ordering the line segments visible inside the rectangle in the appropriate way (see [2]) or by moving the boundaries of the window to the inside in which case it can be solved

using the methods for moving windows. When we enlarge the window, only structural changes (i) (ii) and (iii) can occur. See Figure 3.1. for examples. For sake of clarity we assume that no two endpoints of line segments lie on a common horizontal or vertical line. We will also assume that no two changes occur at the same moment. In practice this is not true, but the method can easily be adapted to report all changes that occur at the same moment.

To solve the zooming problem, we divide it into 8 subproblems. We separately determine the first change when we move the upper border upwards, the left border leftwards, the right border rightwards and the bottom border downwards. Next we determine the first change in each of the corners. The answer to the zooming query is that change of the 8 changes found that occurs first. (See Figure 3.2. for the 8 subproblems.)

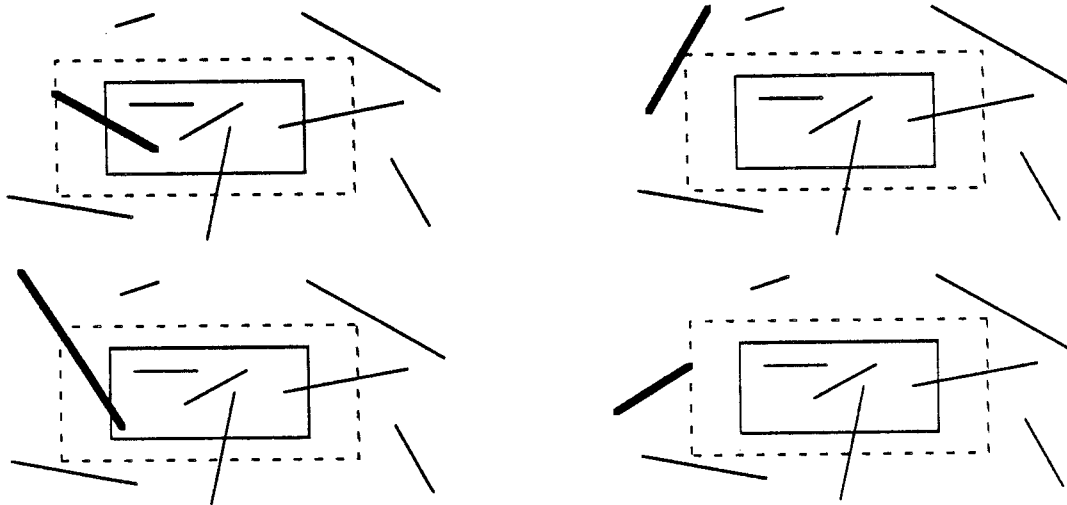


Figure 3.1.

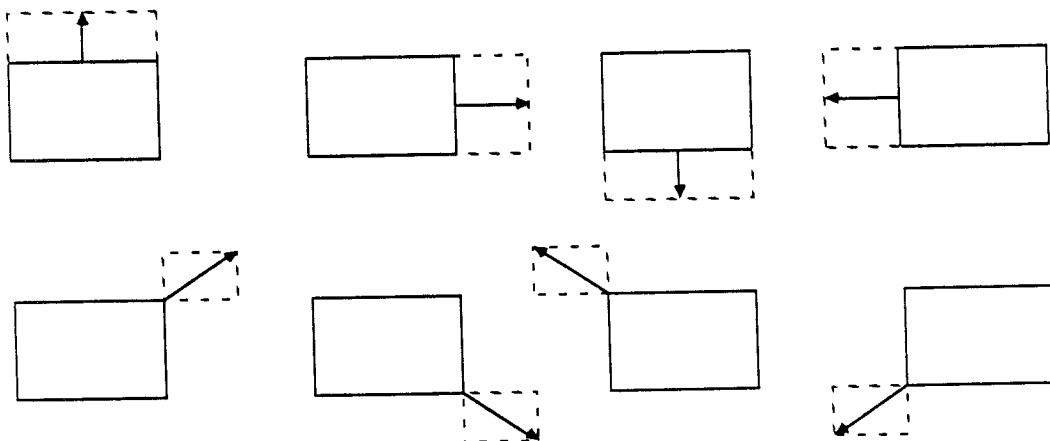


Figure 3.2.

The first four subproblems can be solved by the structures for moving the window upwards, leftwards, rightwards or downwards described in [5]. Hence, we can solve these subproblems in a query time of  $O(\log^2 n)$ , an update time of  $O(\log^2 n)$  using  $O(n \log n)$  storage.

We will now show how to find the first change in a corner. We will only consider the top right corner. The other corners can be treated in a symmetrical way. We can reformulate the problem as follows: given a point  $C$  (the corner) and a direction  $d$ , determine the first point  $C'$  in the direction  $d$  from  $C$  such that the rectangle  $R$  with  $C$  and  $C'$  as corners (partially) contains a line segment (see Figure 3.3.). We will call this problem the *growing range problem*.

We first restrict the set to contain only points rather than line segments. So, we search for the first point  $C'$  such that rectangle with  $C$  and  $C'$  as corners contains a point  $p$  of the set. Clearly, this rectangle has the point  $p$  on one of its edges. Hence, either the  $x$ -coordinate of  $C'$  or the  $y$ -coordinate of  $C'$  is equal to the  $x$ - or  $y$ -coordinate of a point in the set. We split the problem in two subproblems. First we determine the first  $C_x'$  such that the range contains at least one point and  $C_x'$  has its  $x$ -coordinate equal to that of a point in the set. Next we determine the first  $C_y'$  such that the range contains at least one point and  $C_y'$  has its  $y$ -coordinate equal to that of a point. Clearly, either  $C_x'$  or  $C_y'$  is the answer to the query. We will only show how to determine  $C_x'$  efficiently.  $C_y'$  can be computed analogously.

We store the points in the set ordered by their  $x$ -coordinates in the internal nodes of a balanced binary search tree  $T$ . We also store them in a structure  $S$  for solving the range detection problem (see Theorem 2.1.). Let  $\alpha$  be the root of tree  $T$  containing the point  $p_\alpha = (x_\alpha, y_\alpha)$ . The method is now best described by the following recursive procedure:

---

Let  $C' = C$ .

- (1) If  $\alpha$  is a leaf, and  $C' \neq C$ , then  $C'$  is the answer. If  $C' = C$  there is no answer.
  - (2) If  $\alpha$  is not a leaf, then compare  $x_\alpha$  with  $x_C$ , the  $x$ -coordinate of  $C$ . Case 1: If  $x_\alpha < x_C$  continue the search in the right subtree. Case 2: If  $x_\alpha > x_C$  determine the point  $C_\alpha$  in direction  $d$  from  $C$  with  $x$ -coordinate equal to  $x_\alpha$ . Perform a query with the range between  $C$  and  $C_\alpha$  on  $S$ . If the range contains no points continue the search in the right subtree. Otherwise, set  $C' = C_\alpha$  and continue the search in the left subtree.
- 

In this way we perform at most  $O(\log n)$  range detection queries to determine  $C_x'$ . Similar for  $C_y'$ .

**Lemma 3.1.** *Given a set of  $n$  point in the plane, the growing range problem can be solved in a query time of  $O(\log^2 n)$ , an update time of  $O(\log^2 n)$ , using  $O(n \log n)$  storage.*

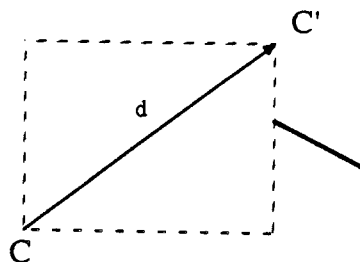


Figure 3.3.

**Proof.**

To perform a query we perform  $O(\log n)$  range detection queries that take  $O(\log n)$  each. Updating the structure  $S$  takes  $O(\log^2 n)$  according to Theorem 2.1. Clearly, updating and searching in  $T$  takes only  $O(\log n)$ . Also the amount of storage required follows from Theorem 2.1. □

We will now solve the general problem, i.e., we have a set of non-intersecting line segments, rather than points. To this end, we first perform the query described above on the set of endpoint of the line segments. This results in an answer  $p$ , where  $p$  is the endpoint of some segment  $s$ . Let  $R$  be the range that contained  $p$  and no other endpoint. If  $s$  is not the right answer then there must be another segment  $s'$  with no endpoint in  $R$  that intersects  $R$ . All other possible answers  $s'$  must intersect the boundary of  $R$  twice and, because they do not intersect, appear ordered along this boundary. (See Figure 3.4.) Hence, the right answer is either  $s$  or the first line segment we encounter when we move from  $C$  along the boundary of  $R$ . Finding the first segment that intersect the boundary can be done by using the structure for shooting (Theorem 2.2.). We shoot from  $C$  horizontally along the boundary of  $R$ . When we hit a line segment that is not  $s$  we stop. (When we hit  $s$  we continue shooting. See e.g. Figure 3.5.) When we arrive at  $C_2$  (see Figure 3.4.) we shoot vertically until we either hit a segment that is not  $s$ , or we reach  $C_3$ . Similar, we shoot from  $C$  upwards to  $C_4$  and next horizontally to  $C_3$ . Determining whether  $s$  or the found segment  $s'$  is the right answer can be done in time  $O(1)$ .

**Lemma 3.2.** *The growing range problem in a set of  $n$  non-intersecting line segments can be solved in a query time of  $O(\log^2 n)$ , an update time of  $O(\log^2 n)$ , using  $O(n \log n)$  storage.*

**Proof.**

This immediately follows from Lemma 3.1. and Theorem 2.2. □

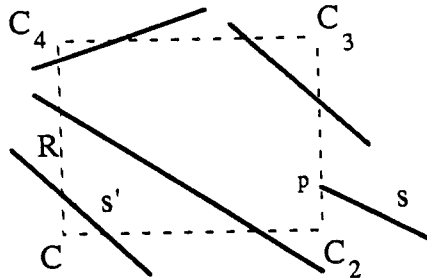


Figure 3.4.

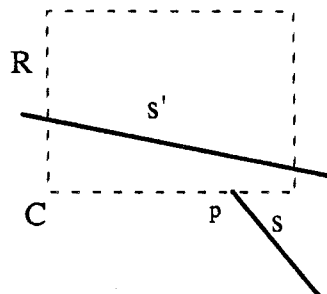


Figure 3.5.

**Theorem 3.3.** *There exists a structure for the zooming problem in a set of  $n$  non-intersecting line segments with a query time of  $O(\log^2 n)$ , an update time of  $O(\log^2 n)$  using  $O(n \log n)$  storage.*

**Proof.**

The zooming problem is solved by four instances of the moving problem and four instances of the growing range problem. The theorem follows from Lemma 3.2. and the known bounds for the moving problem. □

**4. Extensions.**

The results in this paper can be extended in many ways. First of all, there is no need for the window to grow in all directions with the same speed. Hence, the method can also be used for windows that grow in one or two directions. Second, it is possible to reduce the deletion time to  $O(\log n)$  by using some global rebuilding techniques from Overmars[4]. Detail are left to the reader. Third, we do not have to restrict the set to contain only line segments. The methods in [5] and, hence, also our methods apply equally well to other type of objects, like e.g. circles, arcs, etc., as long as they do not intersect.

The technique of repeated range detection has some other interesting applications as well. As an example, consider the nearest neighbor problem in a  $L_1$  or  $L_\infty$  metric. In this case, finding the nearest neighbor to a given point  $p$  can be solved by growing a square around  $p$  until it contains a point of the set. It is clear that our method for growing a range can be used to solve this problem. Hence, we obtain.

**Theorem 4.1.** *Given a set of  $n$  points in the plane, there exists a structure such that  $L_1$  and  $L_\infty$  nearest neighbor queries can be performed in time  $O(\log^2 n)$ . The structure uses  $O(n \log n)$  storage and has  $O(\log^2 n)$  update time bounds.*

We believe that many other problems can be solved as well by repeated range detection or repeatedly performing some other type of query.

**5. References.**

- [1] Edelsbrunner, H., A note on dynamic range searching, Bull. of the EATCS 15 (1981), pp. 34-20.
- [2] Edelsbrunner, H. , M.H. Overmars and R. Seidel, Some methods of computational geometry applied to computer graphics, Computer Vision, Graphics, and Image Processing 28 (1984), pp. 92-108.
- [3] McCreight, E.M., Priority search trees, Techn. Rep. CSL-81-5, XEROX Palo Alto research center, 1981.
- [4] Overmars, M.H., The design of dynamic data structures, Lect. Notes in Comp. Science Vol. 156, Springer-Verlag, 1983.
- [5] Overmars, M.H., Range searching in a set of line segments, Proc. Symp. on Computational Geometry, 1985, pp. 177-185.
- [6] Overmars, M.H., Geometric data structures for computer graphics, in: R.A. Earnshaw (Ed.), Fundamental Algorithms for Computer Graphics, NATO ASI Series F Vol. 17, Springer-Verlag, 1985, pp. 919-931.
- [7] van Leeuwen, J., Graphics and computational geometry, Les Mathematiques de l'Informatique, Colloq. AFCET, Paris (1982), pp. 159-165.