GEOMETRIC DATA STRUCTURES FOR COMPUTER GRAPHICS

Mark H. Overmars

RUU-CS-85-13
april 1985

Rijksuniversiteit Utrecht

Vakgroep informatica

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

GEOMETRIC DATA STRUCTURES FOR COMPUTER GRAPHICS

Mark H. Overmars

Technical Report RUU-CS-85-13
april 1985

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

GEOMETRIC DATA STRUCTURES FOR COMPUTER GRAPHICS

Mark H. Overmars

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

1. Introduction.

The area of Computational Geometry deals with the study of algorithms for problems concerning geometric objects like e.g. lines, polygons, circles etc. in the plane and in higher dimensional space. Since its introduction in 1976 by Shamos[17] the field has developed rapidly and nowadays there are even special conferences and journals devoted to the topic. A list of publications by Edelsbrunner and van Leeuwen[8] collected in 1982 already contained over 650 papers. And this number has rapidly increased since then.

Clearly, a large number of problems in Computer Graphics deal with geometric objects as well. Examples are hidden line elimination, windowing problems, intersection problems etc. Hence, Computer Graphics can benefit from the techniques developed in Computational Geometry. van Leeuwen[18] was the first to make this explicit. (Of course already a number of problems considered in Computational Geometry were based on problems arising in Computer Graphics.) He showed that solutions to problems like nearest neighbor searching and range searching could be useful in Computer Graphics.

In this paper we will give some examples of how techniques from Computational Geometry can be applied to Computer Graphics. We will concentrate on the windowing problem: given a large picture, built out of non-intersecting line segments, compute that part of the picture visible in a axis-parallel rectangle. (See figure 1.1. for an example.) We will show how to store the picture in a kind of two-dimensional data structure such that for each given window we can efficiently determine what part of the picture is visible in the window. This is particular useful when the

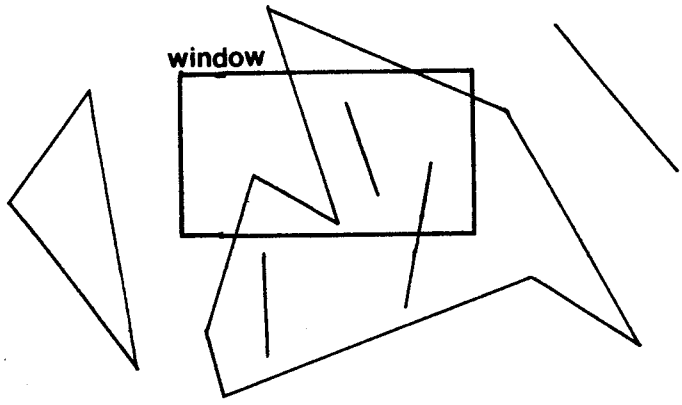

Figure 1.1.

pictures are large (compared to the size of the window) and do not change often. For example, large maps stored in a database. We will give both static solutions (i.e., the picture does not change) and dynamic solutions (i.e., the picture can be updated by means of insertions and deletions of line segments).

The paper is organized as follows. First we recall some known results from computational geometry. Next we apply these to the windowing. In the final section we give some extensions. The methods presented are not worked out in detail. Details can be found in the cited papers.

Although the results are merely theoretical they show new ways of solving this type of problems that might lead to fast practical solutions.

The results in this paper are based on work by Edelsbrunner, Overmars and Seidel[7,14].

2. Preliminary results.

In this section we will mention some results and techniques from computational geometry that we will use in the following sections.

2.1. Point location.

The point location problem is the following: Given a subdivision of the plane in polygonal areas, store this subdivision in such a way that the area a given point lies in can be determined efficiently. The polygonal subdivision is made up of n line segments. The problem has been considered in great detail. (See e.g. Lee and Preparata[10], Preparata[16], Edelsbrunner and Maurer[5], Edelsbrunner, Guibas and Stolfi[4] and Kirkpatrick[9].) The best known result is:

> Theorem 2.1.1. [4,9] Given a polygonal subdivision of the plane of n line segments, it can be stored using O(n) storage such that point location queries can be answered in O(log n) time.

For the method used see the cited papers. This is the best we could hope to achieve. The method in [4] also applies to subdivision that are composed of non-straight edges.

The point location problem is useful in solving many other searching problems in the plane.

2.2. The locus approach.

The locus approach is a general technique for solving many types of searching problems. In a searching problem we have a set of objects we want to store such that given a query object we can answer some question about this query object with respect to the set of objects. These query objects are chosen from some query space. For example, they can be all possible points in the plane or all possible windows. For a fixed set of objects it is often possible to split the query space into a number of areas of constant answer, i.e., areas such that for all query objects in one area the answer is the same. In such a case we can solve the searching problem by making such a subdivision of the query space and storing with each area the corresponding answer. Performing a query now consists of determining the area the query object lies in and reporting the corresponding answer. Hence, we can solve the problem using point location. (See Overmars[13] for a general treatment of the locus approach.)

2.3. Range searching.

The range searching problem is the following: given a set of point in the plane (or a multi-dimensional space), store them in such a way that those points lying in a given axis-parallel rectangle (range) can be determined efficiently. Hence, the range searching problem is a restricted version of the windowing problem in which the picture consists of points rather than line segments. A number of solution to the range searching problem have been proposed. (See e.g. Bentley and Friedman[1], Edelsbrunner[3], Lueker[11], Overmars and van Leeuwen[15] and Willard[19,20].) We will shortly describe one. (For details see e.g. [19].)

A range tree is a balanced binary search tree containing all points in the set sorted with respect to their first coordinates in the leaves. With each internal node β we associate a balanced binary search tree of all point in the subtree rooted at β, sorted with respect to their second coordinate. To perform a query with a range $([x_1:x_2],[y_1:y_2])$ we search with both x_1 and x_2 in the tree. Now look at all nodes γ whose father is on one of the two search paths and who are lying in between the two search paths. (See figure 2.3.1.) Their subtrees together span the whole interval between x_1 and x_2. Hence, all points in between x_1 and x_2 are stored in exactly one of the structures associated with the nodes γ. In these associated

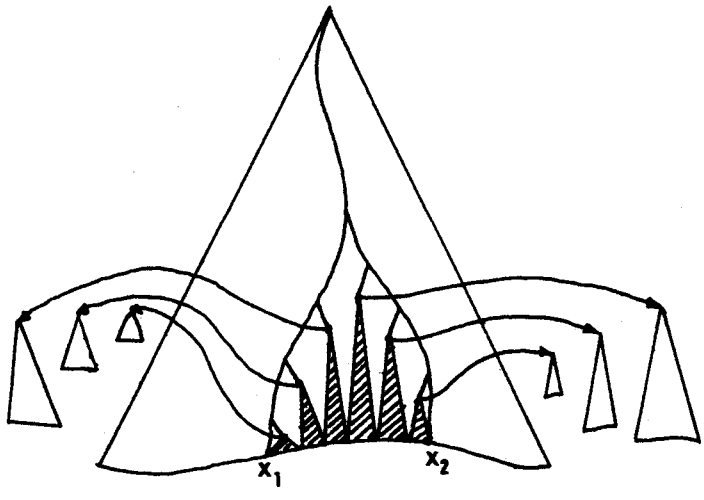

Figure 2.3.1.

structures we search for the points lying in between y_1 and y_2.

As the number of nodes γ is bounded by $O(\log n)$ (n=the number of points) and the time required to search an associated structure is bounded by $O(\log n)$ plus the number of answers, the total query time is bounded by $O(\log^2 n)$ plus the total number of answers. One easily shows that the amount of storage required is bounded by $O(n.\log n)$.

Theorem 2.3.1. One can store a set of n point using $O(n.\log n)$ storage such that range queries can be performed in $O(k+\log^2 n)$ time where k is the number of answers. The structure can be updated in $O(\log^2 n)$ time per insertion and $O(\log n)$ time per deletion of a point.

The insertion and deletion methods can be found in [19].

3. Windowing.

We will describe two different solutions. The first one is static and only works for fixed size windows. Its advantage is that it only uses linear storage. The second solution is a fully dynamic solution, allowing for arbitrary sized windows, but it uses $O(n.\log n)$ storage. We will not go into details. They can be found in [7,14].

3.1. A static solution.

In this subsection we assume that the window has some fixed size. (In fact, we only need it to have fixed height.) To determine which line segments lie in the window we solve two subproblem. Firstly, we determine which line segments have one of their endpoints in the window, and secondly, we determine the line segments that intersect the boundary of the window. In this way we clearly obtain all segments that are visible in the window.

Let us first concentrate on the problem of finding the segments with one of their endpoints in the window. Hence, we are given a set of points and we ask for those that lie in the window. This is in fact the range

searching problem. We cannot use the result from the previous section because we want to achieve linear storage. We will now solve the range searching problem for fixed size ranges in a better way. To find those points that lie in the window we move the left boundary of the window horizontally toward the right boundary, reporting each point we pass. To do this efficiently we need to have a way to determine the next point hit by the left boundary. Hence, we have to solve the following subproblem: given a set of point in the plane, store them in such a way that the first point hit when moving a vertical line segment of fixed height h to the right can be determined efficiently. See figure 3.1.1. for an example. We solve this subproblem by transforming it into another problem. We replace each point p in the set by a vertical line segment of height h with p as topmost point. Now we can as well ask the question: determine the first line segment hit when moving the bottom point of the query line segment to the right. See figure 3.1.2. This problem we can solve using the locus approach. We get a subdivision of the plane as shown in figure 3.1.3. Using point location we can find the area the query point (= the bottom point of the left boundary of the window) lies in and this gives us the first point hit when moving the left boundary to the right. Using a technique by Chazelle[2] for walking in a planar subdivision we can continue moving the left boundary to the right by walking with the bottom point

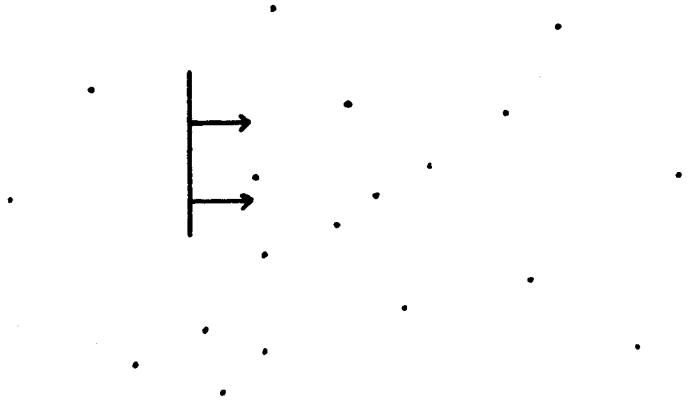

Figure 3.1.1.

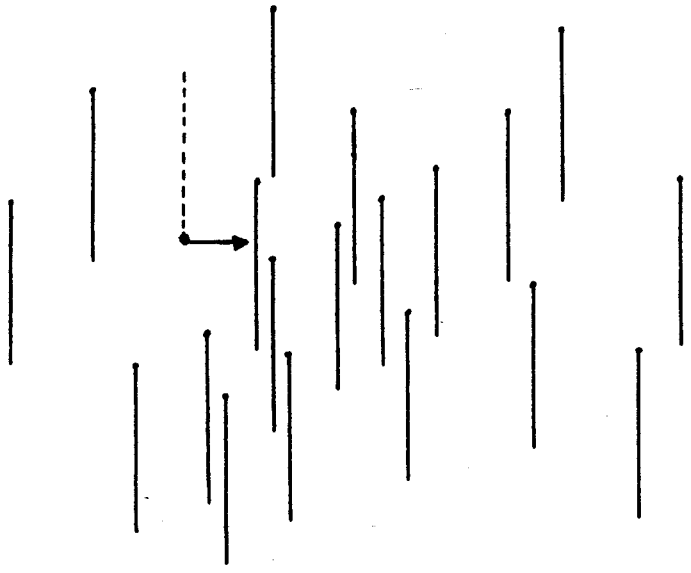

Figure 3.1.2.

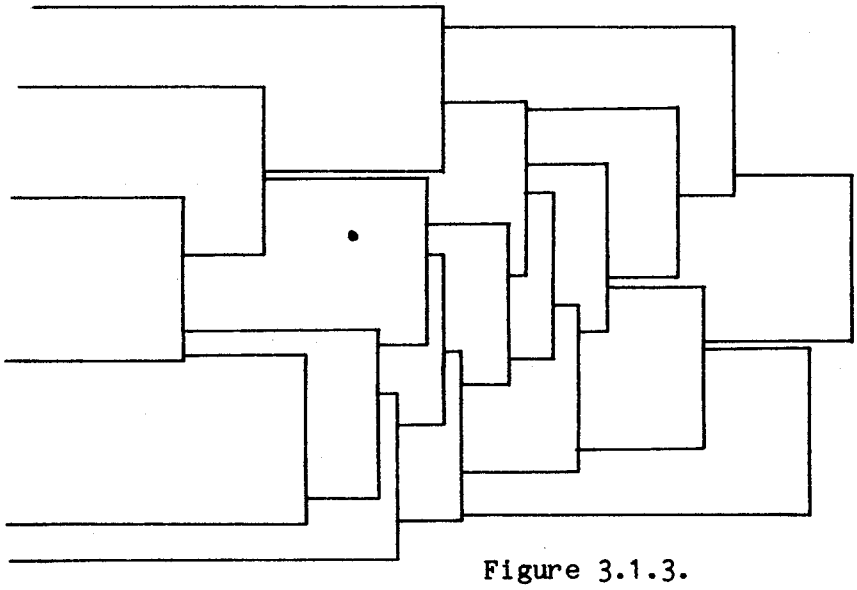

Figure 3.1.3.

through the subdivision, reporting each area we pass until we reach the right boundary. Combining the results for point location and walking in a subdivision this leads to the following result:

> Theorem 3.1.1. Given a set of n points in the plane we can store them using O(n) storage such that those k points lying in a fixed sized window can be determined in O(k+log n) time.

This solves our first subproblem. The second subproblem asks for those line segments that intersect the window. We will only consider the top

boundary of the window. The other boundaries can be treated in a similar way. To solve this problem we move a point from the left side of the top boundary to the right side, reporting all line segments we pass. To do so we again use the locus approach to be able to determine the first line segment hit when moving the point along the top boundary. See figure 3.1.4. for the subdivision we get. Again we can walk through the subdivision to find all line segments intersecting the top boundary using the technique of Chazelle[2].

Theorem 3.1.2. Given a set of n non-intersecting line segments in the plane, those k segments intersecting a given horizontal line segment (the top boundary of the window) can be determined in O(k+log n) time.

A similar method applies to the other boundaries of the window. Combining the two results we obtain:

Theorem 3.1.3. Given a set of n non-intersecting line segments in the plane, we can store them using O(n) storage such that those k line segments visible in a fixed sized window can be determined in O(k+log n) time.

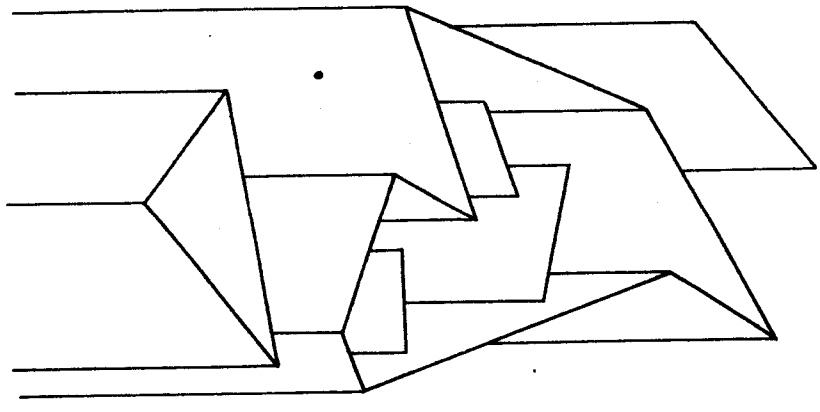

Figure 3.1.4.

Of course, some provision has to be made such that visible line segments are reported only once. See [7] for details.

3.2. A dynamic solution.

We will now describe a fully dynamic solution to the windowing problem that allows for arbitrary sized windows. To do so we again split the problem in two parts like in the previous subsection. For finding the segments with an endpoint in the window we now use the dynamic range tree. This yields a query time of $O(\log^2 n)$, an insertion time of $O(\log^2 n)$ and a deletion time of $O(\log n)$ using $O(n.\log n)$ storage. So we only have to consider the second subproblem: find those line segments that intersect the boundary of the window. We will again only look at the top boundary. The other boundaries follow in a similar way.

First we project all line segments on a vertical line. This divides the line in a number of "elementary" intervals $[-\infty,a_1]$, $[a_1,a_2]$, ... , $[a_n,\infty]$. See figure 3.2.1. We build a balanced binary search tree that has these elementary intervals as leaves. An internal node β contains the whole interval covered by the subtree rooted at β. With each node β we associate a structure (to be described below) that contains all line segments whose projection covers the interval of β but not the interval of the

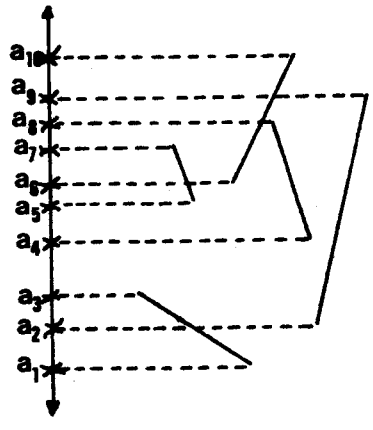

Figure 3.2.1.

father of β. In figure 3.2.2. the line segments that have to come in the associated structures are indicated.

To search with the top boundary of the window (being the line from (x_1,y_2) to (x_2,y_2)) we search with y_2 in the structure. Now it is easy to see that each line segment intersecting the top boundary is in exactly one of the structures associated with the nodes on the search path of y_2. Hence, we only have to search in these structures.

The line segments that are associated with a node β we restrict to the part that lies in the horizontal slab corresponding to the interval stored in β. See figure 3.2.3. As the line segments do not intersect they appear ordered in this slab. In this order we store them in the internal nodes of a balanced binary search tree T_β. When we have to search in a structure T_β y_2 did pass through node β and, hence, the top boundary lies in the slab. We now search with x_1 and x_2 in T_β. In this way we can find in $O(\log n)$ time (plus the number of answers) those segments in T_β that intersect the top boundary. We have to do this for all nodes β on the search path of y_2. These are $O(\log n)$ nodes. Hence, the total query time becomes $O(\log^2 n)$ plus the number of answers.

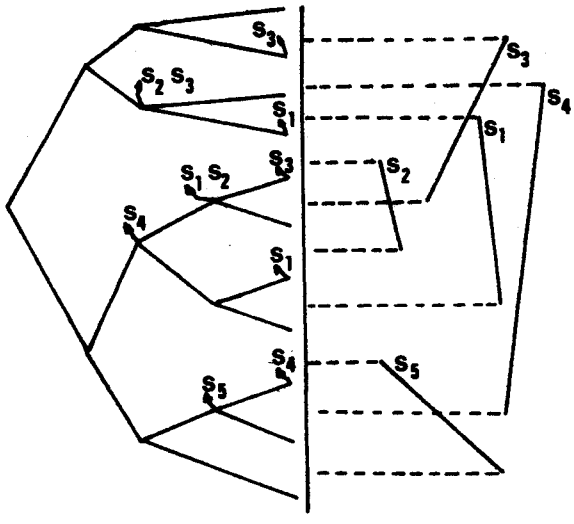

Figure 3.2.2.

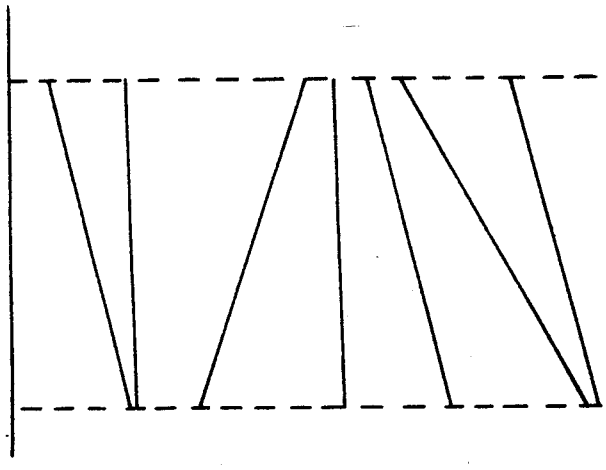

Figure 3.2.3.

It is easy to see that each line segment is stored in at most O(log n) associated structures. Hence, the amount of storage required is bounded by O(n.log n).

Insertion and deletion methods can be obtained in a way similar to Willard[19] or by using general dynamisation techniques (see Overmars[12] for an overview). This yields insertion and deletion time bounds of $O(\log^2 n)$.

Combining the known result for range searching and the structure described above we obtain:

> Theorem 3.2.1. Given a set of n non-intersecting line segments in the plane, we can store them using O(n.log n) storage such that those k segments visible in a given window can be determined in $O(k+\log^2 n)$ time. The structure is dynamic and can be updated in $O(\log^2 n)$ time.

For more details see [14].

4. Conclusions and Extensions.

We have shown how some techniques and solutions from Computational Geometry can be used to solve the windowing problem in Computer Graphics.

Although the methods are theoretical they might lead to new, fast, practical solutions. Similar techniques apply to related problems like moving a window over a picture and enlarging and reducing a window. For solutions to these problems see [6,14].

The techniques in this paper can be extended in many ways. For example, there is no need to restrict the attention to pictures build out of line segments. The solutions can easily be adapted to work for more general classes of objects like e.g. circle arcs, small polygons, etc., as well.

Many open problems do remain. A very interesting question is whether these techniques can be turned into practical solutions. Also on the theoretical level numbers of open problems do remain. An important question being whether the solutions are optimal or can be improved by reducing the query time or amount of storage required.

5. References.

[1] Bentley, J.L. and J.H. Friedman, Data Structures for Range Searching, ACM Comput. Surveys 11 (1979), pp. 397-409.

[2] Chazelle, B.M., Fast Computation of Segment Intersections, Techn. Rep. CS-83-11, Dept of Computer Science, Brown University, 1983.

[3] Edelsbrunner, H., A Note on dynamic Range Searching, Bull of the EATCS 15 (1981), pp. 34-40.

[4] Edelsbrunner, H., L.J. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, manuscript.

[5] Edelsbrunner, H. and H.A. Maurer, A Space-optimal Solution of General Region Location, Theoretical Computer Science 16 (1981), pp. 329-336.

[6] Edelsbrunner, H. and M.H. Overmars, Zooming by Repeated Range Detection, Techn. Rep. RUU-CS-84-10, Dept. of Computer Science, University of Utrecht, 1984. (to appear)

[7] 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.

[8] Edelsbrunner, H. and J. van Leeuwen, Multidimensional Data Structures

and Algorithms: A Bibliography, Techn. Rep. F105, Inst. f. Information Processing, TU Graz, 1982.

[9] Kirkpatrick, D.G., Optimal Search in Planar Subdivisions, SIAM J. Computing 12 (1983), pp. 28-35.

[10] Lee, D.T. and F.P. Preparata, Location of a Point in a Planar Subdivision and its Applications, SIAM J. Computing 6 (1977) pp. 594-606.

[11] Lueker, G.S., A Data Structure for Orthogonal Range Queries, Proc. 19th IEEE Symp. on Foundations of Computer Science, 1978, pp. 28-34.

[12] Overmars, M.H., The Design of Dynamic Data Structures, Lect. Notes in Computer Science 156, Springer-Verlag, 1983.

[13] Overmars, M.H., The Locus Approach, in: M. Nagl and J. Perl (ed.), Proc. 9th Conf. on Graphtheoretic Concepts in Computer Science (WG83), Trauner Verlag, 1983, pp. 263-273.

[14] Overmars, M.H., Range Searching in a Set of Line Segments, Techn. Rep. RUU-CS-83-6, Dept. of Computer Science, University of Utrecht, 1983. (to appear in Proc. of the Symp. on Computational Geometry, Baltimore, 1985)

[15] Overmars, M.H. and J. van Leeuwen, Worst-case Optimal Insertion and Deletion Methods for Decomposable Searching Problems, Inform. Proc. Letters 12 (1981), pp. 168-173.

[16] Preparata, F.P., A New Approach to Planar Point Location. SIAM J. Computing 10 (1981), pp. 473-482.

[17] Shamos, M.I., Computational Geometry, Ph.D. thesis, Dept. of Computer Science, Yale University, 1978.

[18] van Leeuwen, J., Graphics and Computational Geometry, Les Mathematiques de l'Informatique, Colloq. AFCET, 1982, pp. 159-165.

[19] Willard, D.E., The Super-B-Tree Algorithm, Techn. Rep. TR-03-79, Aiken Computer Lab., Harvard University, 1979.

[20] Willard, D.E., New Data Structures for Orthogonal Queries, SIAM J. Computing, in press.