GRAPHICS AND COMPUTATIONAL GEOMETRY

Jan van Leeuwen

RUU-CS-81-18

December 1981

Rijksuniversiteit Utrecht

Vakgroep informatica
Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–531454
The Netherlands

9

83 3/19

GRAPHICS AND COMPUTATIONAL GEOMETRY

Jan van Leeuwen

Technical Report RUU-CS-81-18
December 1981

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

GRAPHICS AND COMPUTATIONAL GEOMETRY *

Jan van Leeuwen

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

Abstract. We illustrate the paradigms of computational geometry through a number of examples taken from the area of computer graphics. The implications of the use of algorithmic tools like balanced search trees, scan lines and standard conversions (dynamisations) from static to dynamic data structures will be highlighted. Several results and open problems are reviewed.

1. Introduction

The plotter used to be the only graphic output device for a computer. It is a slow and inherently non-interactive device that is easy to program but, through a variety of technical restrictions on the pen movements, hard to use when a flexible drawing capability is required. As computers are getting cheaper and easier to acquire, the applications of computers for display purposes and the need for sophisticated software systems to drive such applications multiply. Teletypes no longer are the only medium for interacting with a system and plotters have been replaced by hardware-driven CR tubes ("screens") in many applications. In fact, the market for interactive terminals with some type of screen or display attached is rapidly expanding and is reaching into almost every professional and non-professional environment. We shall not pursue the problems involved in displaying text, but rather want to study the computational aspects of displaying graphic objects like points, line segments, rectangles and other, perhaps less regular geometric forms on a screen. There is more to it

* Invited paper, to be presented at the AFCET Symposium "Les Mathématiques de l'Informatique", March 16-18, 1982.

than meets the eye (literally ...) and in recent years the area of computer graphics has become a branch of computer science by itself. Newman and Sproull [20] (p. xvii) write that "the graphical display is ... one of the most fascinating devices that computer technology has produced". In this paper we shall illustrate that the need for efficient algorithms to plot or display graphic objects leads to challenging problems in both mathematics and computer science.

Computer graphics involves "the generation, representation, manipulation, processing and/or evaluation of graphic objects by a computer as well as the association of graphic objects with nongraphic information residing in computer files" (Giloi [15], p. 3). Plotters are driven by special channels that execute plot routines written in a simple command language, with primitives for all permissible pen movements. Screens are normally driven by a special display processor which executes a compiled or interpreted display program that is stored in a display file, together with the necessary object data. It should be clear that there are succinct problems for the computer scientist all over, ranging from the design of suitable data structures and algorithms for graphic plots and displays to the design of suitable graphic programming languages and support systems.

The design of efficient algorithms for graphic applications normally requires an augmentation of the classical (mathematical) insights in geometry, where there has traditionally been a greater emphasis on the abstract properties than on the representation and actual computation of geometric objects. Theories of constructability have replaced the practice of construction and proving theorems has replaced the design of proper algorithms (whereas the latter is required in current graphic applications). Only a few mathematicians (e.g. Lemoine [18]) have studied the complexity of finite geometric objects and their construction with the aim of obtaining feasible algorithms. About 1975, Shamos [25] (also [26]) initiated a major study of the computational aspects of elementary geometry, with the primary goal to establish upper- and lowerbounds on the complexity of all common problems in this area. His study included algorithms for e.g. convexity testing, convex hull construction, inclusion and intersection problems, and a variety of closest point problems, and succeeded in proving good lowerbounds on the intrinsic complexity of some of them.

A simple example will illustrate the nature of the results in computational geometry. Suppose $K = (p_1, p_2, \cdots, p_n)$ is a simple n-gon in the plane. Consider the problem of determining for arbitrary points p whether p belongs to the interior or to the exterior (or perhaps ... to the boundary) of K. In mathematics there are two well-known solutions to this problem (cf. Giloi [15], p. 159):

(i) traverse K and sum the angles $< p_i\, p\, p_{i+1}$ for i from 1 to n (with $p_{n+1} \equiv p_1$). Then p is inside of K if the sum is 2π and outside of K if the sum is 0.

(ii) draw a ray from p to infinity and count the number of intersections with K's contour. Then p is inside of K when the number of intersections is odd and outside of K when this number is even.

From the computer scientist's point of view both solutions are easy to program but require $\Theta(n)$ steps of computation. Moreover, whenever we need to solve the problem for another point q, then the same number of steps must be spend again. Shamos [26] has shown that a simple n-gon can be represented differently so containment queries can always be answered in $O(\log n)$ steps of computation. By means of the techniques of Dobkin & Lipton [7] one can show that this bound cannot be improved in order of magnitude (i.e., Shamos' algorithm is essentially optimal).

While studies in computer graphics tend to be device and programming oriented, there is a variety of algorithmic problems in this area for which computational geometry seems to offer the proper paradigms. It is our aim to bring this out, in order to motivate a continued study of the mathematical complexity of these problems. For an introduction to the techniques of algorithm design and analysis, see Aho, Hopcroft & Ullman [1]. An extensive bibliography of the area of computational geometry is given in Edelsbrunner & van Leeuwen [10].

2. The algorithmic setting of computer graphics

The graphic objects in a scene will be stored in a file F. The objects may be stored as individual items (with their characterising coordinates in space) or in some processed order. There are a number of algorithms that need to be designed before there is an image on the display (see figure 1). Algo-

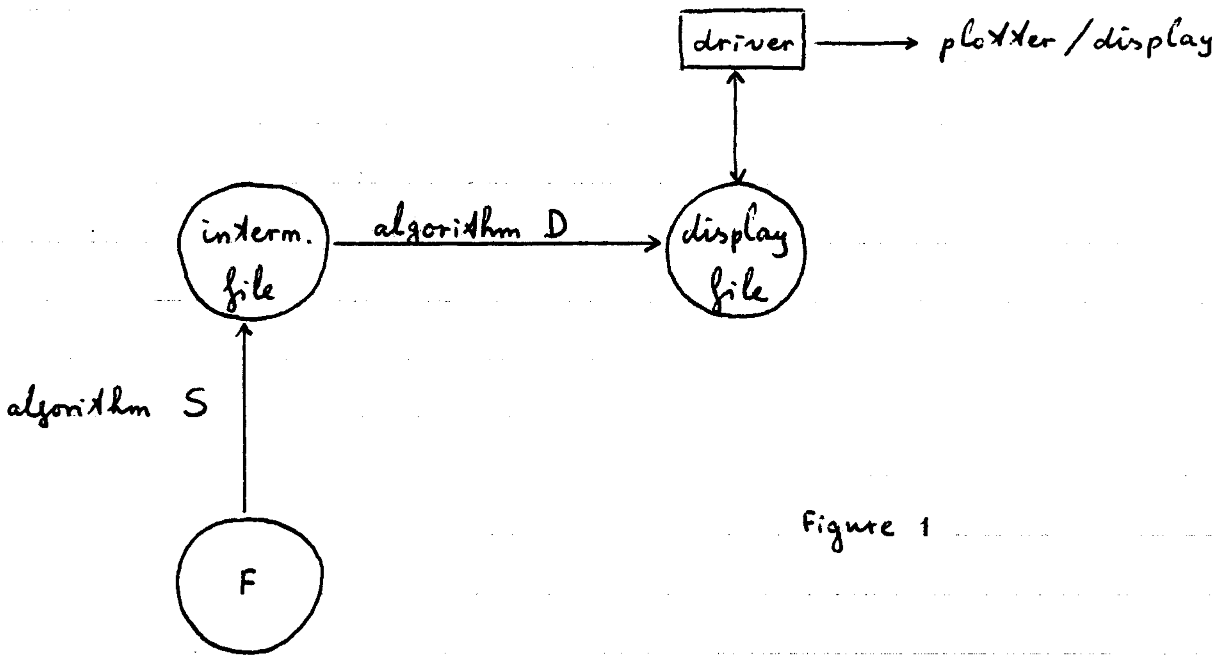

Figure 1

rithm S will select from F the objects we wish to visualise. It commonly are the objects that can be "seen" through a window (a rectangle) of some size, positioned somewhere in space. Algorithm D will structure the resulting subset into a proper display file, which contains the graphic objects in a representation (as a set) that allows for a rapid generation of the image on paper or on the display. In the latter case the output must be continuously regenerated ("refreshed") at a sufficiently high frequency, to maintain a good quality image on the screen. It follows that the problems of computer graphics primarily concern the representation and data structuring of sets of graphic objects for the purpose of rapid display.

There are two fundamental operations on sets of objects in computer graphics:

(i) windowing,

(ii) clipping (or scissoring).

Windowing in 2-space merely consists in determining which objects from

F intersect or are contained in the rectangle representing the window. In 3-space windowing involves the projection of the "view" onto a plane at some stage. Clipping is the elimination of the part of an object that lies outside of the boundary of the window. There are four operations that are typically restricted to objects from the intermediate file :

(iii) rotation,

(iv) translation,

(v) scaling,

(vi) fitting to screen coordinates.

Operations (iii) to (v) are normally performed as simple geometric transforms. (See e.g. Giloi [15], Ch. 3.) In advanced applications there is yet another operation :

(vii) moving.

It involves moving the window through space and updating the image that is displayed on-line. In our further discussion we restrict to 2-space.

3. <u>Plotting a view</u> ; <u>displaying a view</u>

Let an intermediate file I be computed and suppose it contains just a set of points, given by their (x,y)-coordinates. When a point p is plotted, the time to plot a next point q will be proportional to the distance from p to q, measured along the shortest path the plotter head can be made to move from p to q. The composition of an optimum plotting order for I thus reduces to the "geometric traveling salesman problem" (cf. Garey & Johnson [14]), which is known to be NP-complete (hence, computationally intractable for large sets) for several metrics. An algorithm due to Christofides [6] is computationally tractable and produces an ordering that is within a factor $3/2$ from the optimum. Several probabilistic algorithms exist (e.g. Karp [16]) that produce a nearly optimal ordering for random sets. Different optimizations may be of interest as well. For instance, to minimise the changes of directional parameters one may want to know the smallest number of straight lines that collectively contain all points of I. This problem is likely to be NP-complete. No approximation algorithms are known to circumvent its apparent computational inattractiveness.

An entirely different set of problems arises when I consists of a set of line segments. The plotting problem will not be easier than it is for a set of points, but apparently no good heuristics for this problem are known. Assuming line segments are always drawn straight to completion, the composition of an optimum plotting order may be viewed as a special instance of the stacker-crane problem (Frederickson, Hecht & Kim [12]) for which a polynomial time approximation algorithm exists that yields solutions within a factor $9/5$ from the optimum. Better bounds are lacking. No results are known at all when the plotting of a line segment may be interrupted at some halfway point, to plot some nearby segments (or parts of it!) before continuing. An interesting mathematical question would be to prove a (polynomial) bound on the number of "interruptions" required for an optimal plot. In all cases special results for sets of horizontal and/or vertical line segments only would be of interest as well.

When a set of n line segments is plotted, it is important for a good quality drawing that no line fragment is traced over twice (unless specifically desired). Consider this problem for a set of k line segments that happen to lie along the same infinite line. In a balanced search tree one can keep track of the regions covered, i.e., the connected components of the line segments processed. Insertion of another line segment means (i) determining the component(s) containing the left and right endpoints of the segment, (ii) tracing the intermediate gaps (which takes $O(1)$ in computational overhead per gap) and (iii) deleting the row of components connected by the segment from the tree and inserting the new one formed into it. When the right splittable data structure is chosen (see Aho, Hopcroft & Ullman [1], Ch. 4) the operations need only $O(\log k)$ time, excluding plotting time. The total computational overhead for an arbitrary set will be $O(n \log n)$.

When an intermediate file I must be displayed on a screen, the medium imposes different restrictions and different tactics for an algorithmic approach.

Problems also get a succinct dynamic touch as one can, using a light pen or an interactive command language, modify (add/delete) the contents of the display file and (ultimately) the contents of F. Thus, algorithm D (see section 2) may have to structure I for dynamic rather than merely static usage. We assume that the view has been scaled to the size of the screen and ignore the need for rounding to screen coordinates.

The screen is normally subdivided by equidistant horizontal and vertical lines (at least conceptually.....) to create a raster. The electron beam periodically traverses the raster and pulses at every raster point that needs to be displayed (or, refreshed). The process permits about 2000 points to be displayed without flicker. We shall assume that the beam traverses the vertical raster lines from left to right and that the beam-controlling software requires the ordering of the points rather than their precise coordinates along a raster line before traversing it. It follows that a set of n points can be displayed provided we keep the points on every raster line in sorted order. Insertions and deletions take only $O(\log n)$ steps. A different but related method makes use of a centrally located point p (on the screen) and a ray, swinging around p and periodically refreshing the screen. With points given by polar coordinates and a "circular" raster, this method is computationally equivalent to the earlier technique.

The raster scan technique becomes more tedious, both in practice and in theory, once the objects in a view are line segments. It is possible to display points so close together that they appear as a continuous line to the human eye. The display file, however, need not represent a line segment as a collection of points, as long as the software can easily compute the ordered set of points contributed on every raster line while the scan progresses For a set of n line segments an algorithm due to Bentley & Ottmann [4] achieves this using a computational overhead of only $O(n \log n + k \log n)$ steps, where k is the number of intersections of the set. Brown [5] proved that the algorithm can be made to run using only $O(n)$ space. The algorithmic implications of the scan line technique were explored further in e.g.

Nievergelt & Preparata [21].

Straight lines can also be generated using special "vector generation" hardware, which can move the CRT beam continuously from the starting to the ending point of the line. This removes many of the earlier problems, once available.

Given a set of points, it may be required to display it in more particular ways. For example, one may wish to apply a "window" to it (i.e., to file I or file D) and have an enlarged view of the subset selected. Techniques for efficient windowing will be discussed in the next section. It may also be required to display a set together with its convex hull and /or some of its convex layers. There are several techniques to compute the convex hull of a set of n plane points in $O(n \log n)$ or $O(nh)$ steps, where h is the number of points on the hull. For references, see [10]. Overmars & van Leeuwen [23] have shown that a set may be structured so its convex hull can be updated in $O(\log^2 n)$ steps whenever a point is inserted or deleted.

4. Clipping and windowing

Clipping is normally applied after "windowing" has determined which objects from F intersect the view. In some applications clipping may occur while the view is being displayed (analog scissoring, Giloi [15] p. 237). Efficient algorithms are known for clipping line segments when the necessary calculations must be performed on a minicomputer (Newman & Sproull [20] Ch 7, Giloi [15] sect 3.2). Aside from these results, clipping has largely remained unexplored. Using results from [23] one can show that a convex n-gon may be represented so it can be clipped in only $O(\log n)$ steps, regardless the size or relative position of the window. The clipping of an arbitrary simple n-gon will be more complex, as it may result in up to $\Omega(n)$ disjoint "pieces" within the view. It is unknown whether some form of preprocessing can speed up the clipping in this case to $O(\log n)$ as well. Also, no results are known for the clipping of sets of objects as a whole, at a small cost per object.

Windowing is the operation of determining which objects from a given file F intersect a rectangular query region or "window". Windowing is normally facilitated by a suitable structuring of F. In general the results are harder for dynamic sets of objects than they are for static sets. (See also section 5.) Time bounds always include a term "+ A", where A denotes the number of answers reported with the operation. Several interesting algorithmic results are known when the window is rectilinearly oriented with respect to the cartesian coordinate axes of the set. When it is not, the computational difficulty is to avoid a rotation and/or translation of the set for convenient "windowing". Willard [31] has shown that a set of n points in the plane may be organised within $O(n)$ space so windowing (with an arbitrary polygonal window) takes only $O(n^{0.77} + A)$ time. At the expense of a much greater preprocessing and storage cost, Edelsbrunner, Kirkpatrick & Maurer [9] have shown that the windowing of any set of n polygonal objects with a bounded number of edges may be performed in only $O(\log n + A)$ steps. It is unknown whether this bound can be maintained with reduced preprocessing or storage costs. Results for special cases, e.g. for sets of line segments in a fixed number of directions, are also lacking.

If we restrict to rectilinearly oriented windows, then windowing essentially reduces to the "range query" problem known from the study of multi-dimensional data structures. For sets of points Bentley et al. [2], [11] has proposed several (static) generalizations of the ordinary search tree, including quad-trees and k-d trees. The (static) range tree proposed in Bentley [3] leads to a simple solution that requires only $O(\log^2 n + A)$ time per query. Much effort was given to extend the result for higher dimensional point sets and, especially, to obtain dynamic data structures with similar time bounds on the searching algorithms. The best results are due to Willard [30] and Lueker [19], who provided data structures for range searching that have a query time of

$O(\log^2 n + A)$ and an update time of $O(\log^2 n)$. By a technique of Overmars & van Leeuwen [22] (reported again in [28]) the deletion time can be further reduced to $O(\log n)$. An intricate mathematical analysis of the problem due to Fredman [13] proves that these bounds are essentially optimal, at least under some cost measures.

Less is known for windowing different sets of objects. Edelsbrunner [8] addressed the problem for sets of rectilinearly oriented rectangles, and designed a special data structure (the rectangle tree) to handle it both for static and dynamic sets. He proves a bound of $O(\log^3 n + A)$ on the query time, and an update time of $O(\log^3 n)$ in the (slightly more complex) dynamic case. It is intriguing to note that "rectangle intersection" querying in 2 (or: d) dimensions can be reduced to, and thus solved by, a 4 (or: 2d)-dimensional range query problem. A similar observation was made by Lee & Wong [17]. The complexity of windowing collections of other and not necessarily rectilinearly oriented objects remains largely unassessed. Even for the case of arbitrary line segments the problem appears open (but certainly solvable in polylogarithmic time).

The objects reported in a windowing operation must be subjected to clipping before they can be displayed, and perhaps to some further operations. In particular, the collection may have to be structured so it can again be "windowed" efficiently. No results are known (except for the 1-dimensional case) that would indicate e.g. that the required search structure need not be built from scratch and can be obtained by "splitting" the data structure of the set at a lower cost, preferably no more than windowing took in the first place. The problem seems particularly interesting for (orthogonal) range querying. An entirely different set of problems arises if we want to move a window and update the "view" on-line at a low cost. One may want to move e.g. over fixed distances (up or down, left or right) or to the nearest spot (again, in some direction) where some new object enters the view. Again, no algorithmic results are known for these types of iterated and dynamic windowing (or for their combination!).

In 3 dimensions clipping and windowing become a good deal more complex, but there is all reason to believe that there still are computationally attractive alternatives to the "naive" algorithms. Except for (orthogonal) range querying, little is known for the problems we discussed in this case. In graphics there has traditionally been much attention for the problem of hidden line- and hidden surface elimination that now emerges (see e.g. Sutherland, Sproull & Schumacker [27] or Giloi [15] sect 5.3). An interesting algorithmic study of the problem in 2 dimensions was made by Schmitt [24].

5. Dynamization

In most graphic applications the file of objects F is required to be dynamic. The recurring difficulty is to maintain F in a form suitable for efficient searching and/or windowing. While a specific dynamic data structure may exist for the problem at hand, Bentley [3] advocated a more general approach and suggested to look for standard transformations that yield dynamic structures from static building blocks. The approach, termed "dynamization" by van Leeuwen & Wood [29], has led to a number of surprisingly powerful results that only require an efficient static solution to a searching problem as "input". The approach only works for certain classes of searching problems. Let $Q(x, F)$ denote the answer to a searching problem Q (e.g. windowing) with object x, on a file F. A searching problem Q is called decomposable (Bentley [3]) if and only if for every partition $F_1 \cup F_2$ of F one has $Q(x,F) = \square(Q(x,F_1), Q(x,F_2))$, for a constant time computable operator $\square$. For decomposable searching problems one can avoid storing all objects in one large, and presumably static data structure. Instead, the file can be maintained as a dynamic system of disjoint subfiles of smaller size, with each subfile organized in the "static" way. There are several ways to proceed from here. We refer to van Leeuwen & Overmars [28] for an account of the results and developments concerning the dynamization of (static structures for) decomposable searching problems.

Working with the techniques from [23], Overmars [21] has shown that an efficient dynamization exists for many other problems of interest to graphics as well.

6. References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman, The design and analysis of computer algorithms, Addison-Wesley Publ. Comp., Reading, Mass., 1974.

[2] Bentley, J.L., Multidimensional binary search trees used for associative searching, C.ACM 18 (1975) 509-517.

[3] Bentley, J.L., Decomposable searching problems, Inf. Proc. Lett. 8 (1979) 244-251.

[4] Bentley, J.L. and Th. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. C-28 (1979) 643-647.

[5] Brown, K.Q., Comments on "Algorithms for reporting and counting geometric intersections", IEEE Trans. Comput. C-29 (1981) 147-148.

[6] Christofides, N., The traveling salesman problem, in: N. Christofides al. (editors), Combinatorial optimization, J. Wiley & Sons, Chichester, 1979, Chapter 6, pp. 131-149.

[7] Dobkin, D. and R.J. Lipton, On the complexity of computations under varying sets of primitives, J. Comput. Syst. Sci. 18 (1979) 86-91.

[8] Edelsbrunner, H., Dynamic rectangle intersection searching, Bericht 47, Inst. f. Informationsverarb., TU Graz, Graz, 1980.

[9] Edelsbrunner, H., D.G. Kirkpatrick and H.A. Maurer, Polygonal intersection searching, Bericht 64, Inst. f. Informationsverarb., TU Graz, Graz, 1981.

[10] Edelsbrunner, H. and J. van Leeuwen, Multidimensional algorithms and data structures - a bibliography, EATCS Bulletin nr. 11 (1980) 46-74, supplement in: EATCS Bulletin nr. 13 (1981) 79-85.

[11] Finkel, R.A. and J.L. Bentley, Quad trees - a data structure for retrie-

val on composite keys, Acta Inf. 4 (1974) 1-9.

[12] Frederickson, G.N., M.S. Hecht and C.E. Kim, Approximation algorithms for some routing problems, Proc. 17th Ann. IEEE Symp. Found. of Comput Sci., Houston, 1976, pp. 216-227.

[13] Fredman, M.L., A lower bound on the complexity of orthogonal range queries, J. ACM 28 (1981) 696-705.

[14] Garey, M.R. and D.S. Johnson, Computers and intractability - a guide to the theory of NP-completeness, W.H. Freeman & Comp., San Francisco, 1979.

[15] Giloi, W.K., Interactive computer graphics - data structures, algorithm and languages, Prentice-Hall Inc., Englewood Cliffs, NJ., 1978.

[16] Karp, R.M., Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane, Math. Oper. Res. 2 (1977) 209-244.

[17] Lee, D.T. and C.K. Wong, Finding intersections of rectangles by range search, preprint, Dept. of Electr. Engin. and Comput. Sci., Northwestern Univ, Evanston, Ill., 1980.

[18] Lemoine, E., Geometrographie, Paris, 1907.

[19] Lueker, G.S., A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems, Techn. Rep., Dept of Inf. and Comput. Sci., University of California at Irvine, Irvine, 1978.

[20] Newman, W.M. and R.F. Sproull, Principles of interactive computer graphics, McGraw-Hill Book Comp., New York, NY, 1973.

[21] Nievergelt, J. and F.P. Preparata, Plane-sweep algorithms for intersecting geometric figures, ACT-18/Report R-863, Coord. Sci. Lab., University of Illinois at Urbana-Champaign, Urbana, Ill., 1979.

[21] Overmars, M.H., Dynamization of order decomposable set problems, J. Algor. 2 (1981) 245-260.

[22] Overmars, M.H. and J. van Leeuwen, Worst case optimal insertion

and deletion methods for decomposable searching problems, Inf. Proc. Lett. 12 (1981) 168-173.

[23] Overmars, M.H. and J. van Leeuwen, Maintenance of configurations in the plane, J. Comput. Syst. Sci. 23 (1981) 166-204.

[24] Schmitt, A., Reporting geometric inclusions with an application to the hidden line problem, Bericht 18/81, Fak. f. Informatik, Universität Karlsruhe, Karlsruhe, 1981.

[25] Shamos, M.I., Geometric complexity, Proc. 7th Ann. ACM Symp. on Theory of Comput., Albuquerque, 1975, pp. 224-233.

[26] Shamos, M.I., Computational geometry, Ph.D. Thesis (unpubl.), Dept. of Comput. Sci., Yale University, New Haven, Conn., 1978.

[27] Sutherland, I.E., R.F. Sproull and R.A. Schumacker, A characterisation of ten hidden-surface algorithms, Comput. Surv. 8 (1974) 1-55.

[28] van Leeuwen, J. and M.H. Overmars, The art of dynamizing, in: J. Gruska and M. Chytil (eds.), Mathematical Foundations of Computer Science 1981, Lecture Notes in Comput. Sci., vol 118, Springer Verlag, Heidelberg, 1981, pp. 121-131.

[29] van Leeuwen, J. and D. Wood, Dynamization of decomposable searching problems, Inf. Proc. Lett. 10 (1980) 51-56.

[30] Willard, D.E., The super B-tree algorithm, TR-03-79, Aiken Comput. Lab., Harward University, Cambridge, Mass., 1979.

[31] Willard, D.E., Polygon retrieval, Techn. Rep. 79-01, Dept. of Comput. Sci., University of Iowa, Iowa City, 1979.