Using Convex Differences in Hierarchical Representations of Polygonal Maps

Ari Rappoport
Computer Science Department
The Hebrew University
Jerusalem, Israel 91904
arir@humus.huji.ac.il

Abstract

We discuss adaptive, hierarchical, spatial subdivisions of planar polygonal maps. These maps are extensively used in cartography, GIS, computer graphics and computer vision. Our results are applicable to many subdivision schemes, including quad trees, k-d trees, and binary space partitioning (BSP). We focus on cell splitting rules and leaf organization methods, introducing and comparing three of them. The *simple rule* results in short hierarchy trees but leaf operations are costly. The *convex rule* enjoys efficient leaf operations at the expense of taller trees. The *convex differences rule* combines both short hierarchies and efficient operations to create the *convex differences hierarchy (CDH)*. It utilizes the *convex differences tree (CDT)* representation of simple polygons. We also present *vertex enlargement*, a technique for reducing the map complexity which results in shorter trees and simplified splitting rules.

Keywords: Polygonal subdivisions, Geometric data bases, Hierarchical data structures, Spatial data structures, Adaptive data structures, Convex differences tree (CDT), Convex differences hierarchy, Convex decomposition of polygons.

Introduction

A polygonal map (or a planar polygonal subdivision) is a subdivision of the plane into disjoint regions, each of which is a polygon. The elements of the map are the regions, the edges separating them, and the vertices where edges meet. Polygonal maps arise in many applications, among which are geographic information systems (GIS), cartography, computer aided design, computer graphics and computer vision.

The main computational problem associated with polygonal maps is how to represent them efficiently, storage- and computation-wise. Some of the prevalent operations which are to be performed on polygonal maps are point location (given a point, which region does it belong to), segment cutting (given a line segment, which regions does it intersect), dynamic changes (insertion and deletion of edges or regions), and overlay of one map on another.

The approaches towards the problem can be coarsely classified into theoretic, which aim at theoretical optimality, and practical, which aim at good performance for maps encountered in practice.

The point location problem has received much theoretic attention in the field of computational geometry [Pre85], [Ede87]. Optimal solutions require pre-processing time of $O(n \log n)$, $O(n)$ storage and point location time of $O(\log n)$. n is the size of the map, which is the number of the vertices or edges. It does not matter which of them, as they are linearly related by Eulers' formula [Har69]. Some optimal solutions are the triangular hierarchy of Kirkpatrick [Kir83] and the layered dag of Edelsbrunner, Guibas and Stolfi [Ede86]. Although theoretically optimal, these algorithms have significant disadvantages. First, the constants involved may be too large to make the method practical. This is certainly true for the Kirkpatrick algorithm, and maybe also for the layered dag. Second, it is not clear at all that the structures they use are able to efficiently support operations other than point location. Third is the difficulty of implementation. The practicality of these methods has not yet been demonstrated.

The second class of approaches towards polygonal maps aims at practical usefulness, using schemes which are not theoretically optimal (some times they are not even analyzed thoroughly) but which are claimed to achieve good performance in practice. An advantage of these schemes is that most of them are suited to many kinds of operations on the map, not only for point location. Practical methods can be classified according to a number of orthogonal characteristics. Among them are whether the method organizes the embedding space or the input data, whether the method adapts its data structures to the input data or not, and whether the organization of the data is hierarchical or uni-level.

Here are some examples. The quadtree with its many variants [Sam89a] is hierarchical, adaptive and space organizing. The grid file [Nie84] is uni-level, adaptive, and space organizing. Object hierarchies [Kay86] organize the input data and adapt to it. A uniform grid ([Fra88], [Eda84]) is uni-level, organizes the embedding space, and adapts to the data in a very limited sense or not at all. The two books of Samet [Sam89a],[Sam89b] contain a wealth of information about spatial data structures as well as a large bibliography.

In this paper we discuss representation schemes for polygonal maps which are based upon hierarchical, adaptive, disjoint spatial subdivisions. These are probably the most popular among the above methods. Our focus will be on the point location operation, discussing only static maps. This is a limitation, but static maps are very useful in many application areas, including GIS, cartography and computer vision. Our results are applicable to quadtrees, k-d trees, binary space partitioning (BSP) and similar data structures.

The contributions of the paper include:

- Presentation of the convex differences hierarchy (CDH) as a method for hierarchical storage of polygonal maps. The CDH utilizes the convex differences tree (CDT) representation of simple polygons [Rap89b] both as a splitting rule and as a leaf organization scheme. As a splitting rule it provides a better estimate to the complexity of the map than the number of polygons or the number of edges. As a leaf organization scheme it enables more efficient operations on polygons than the usual vertex list representation or a convex decomposition.

- Introduction of vertex enlargement as a technique for reducing the complexity of a polygonal map around vertices having a high degree. Vertex enlargement simplifies cell splitting rules and results in shorter hierarchy trees.

Hierarchical Methods for Polygonal Maps

In this section we discuss adaptive, hierarchical, exact representations of polygonal maps. We limit ourselves to disjoint hierarchies in the set theoretic sense, which are far more useful than other hierarchies. The discussion is valid for other types of geometric objects in addition to polygons.

Adaptive hierarchical methods organize a map in a tree. Each node contains a region (cell) of the plane. If the part of the map which lies in this region is too complex, the region is decomposed (split, subdivided) into a number of disjoint sub-regions. These are represented by child nodes of the original node. The process continues recursively untill there is no need to subdivide. The internal nodes of the tree hold only structural information, while the leaf nodes hold the real information about the data.

To use a hierarchical representation scheme, three main decisions should be made: the nature of the decomposition into sub-regions, the criterion of when to decompose, and the organization of the data in the leaves of the resulting tree. Following is a discussion on these issues. The paper focuses on the last two.

The decision giving the particular scheme its name is the first one, the nature of the decomposition into sub-regions. The main distinction is between methods in which the decomposition is guided by the data and those in which it is not. The latter are still among the adaptive methods, because the second issue, the splitting criterion, considers the input data.

Three popular methods are the *quad tree* [Sam89a], in which a region is always decomposed into four equal sized sub-regions using a horizontal and a vertical line; the *k-d tree* [Ben75], in which the decomposition is made by horizontal and vertical lines alternately, located so as to balance the load between each of the two sub-regions; and binary space partitioning (BSP) [Fuc80], which is similar to the k-d tree in that it tries to balance the load between sub-regions, however the decomposition is not restricted to be done by lines parallel to the coordinates. The results of this paper are relevant to any of these methods.

The second important decision to be made by a particular hierarchical method is how to determine when a node has to be decomposed. The rule of determination is referred to as the *splitting rule*. This decision is independent of the splitting method used. The general policy is to estimate the complexity of the region and decompose the region if its complexity is greater than some pre-defined, constant threshold value, the *splitting threshold*. A basic distinction between complexity measures is whether they are *region based*, *edge based* or *vertex based*. In this paper we consider only region based rules.

The third decision concerns the organization of the data in the leaves of the tree and how to relate this data to the original data (before the decompositions were performed). Considerations here are efficiency of computations on the leaf node, the amount of storage used, and numerical stability.

The three issues addressed above affect the structure and nature of the hierarchy. There are other issues, such as the physical storage of the tree and algorithmic issues concerning the efficient creation of the tree. These issues are not dealt with in the paper.

An important observation concerning polygonal maps which is relevant to the rest of the paper is that the number of edges in the map can be much greater than the number of polygons. This is the case in many maps of interest, such as maps obtained by digitization of cartographic maps or maps in which the polygonal regions are approximations for curved regions.

Vertex Enlargement

An implicit assumption of the general hierarchical scheme outlined in the previous section is that decomposing a node reduces its complexity, otherwise the decomposition could go on forever. A very natural region based splitting rule is to subdivide the region if the number of polygons incident in it is too large. However, a vertex of the polygonal map may have an arbitrary number of polygons meeting in it. No decomposition reduces this number unless the splitting line passes through the vertex. There is always a cell containing this vertex, and this cell will be redecomposed.

One way of overcoming this difficulty is to include the case of a single vertex in a cell in the splitting rule [Sam89a]. The *modified splitting rule* is 'if the number of polygons in the cell exceeds the threshold, decompose it, unless there is only one vertex in the cell'. The rule is very simple, but it may result in a very fine decomposition at the neighbourhood of vertices. In some types of maps, like road maps, the degree of a vertex is not likely to be high, and the drawback vanishes. In other types it may incur an inadmissible cost.

We suggest a new way of treating this situation. Before the decomposition process is started, all vertices having a high degree are found. Call such a vertex a *center vertex.* Every center vertex is replaced by a polygon which intersects only the polygons adjacent to the vertex. The polygon is called the *new polygon.* The edges of this polygon are formed by connecting points on adjacent edges incident on the problematic vertex, unless the angle between the edges is more than 180. The connecting points are called *new vertices* and the segments connecting them *new edges*. The new polygons are marked as such, with a pointer to the center vertex.

In Figure 1 we see a center vertex adjacent to four polygons. The new edges are dashed. The polygon on the left does not cause a new edge because the angle is more than 180.

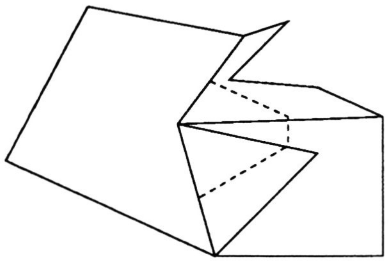
Figure 1: Vertex enlargement.

If the new vertices are sufficiently close to the center vertex, then the new edges will not cross existing edges and the process will result in a new legal polygonal map. Taking care of that is easy. In every polygon incident on the center vertex, we find the vertex closest to the center vertex by examining all vertices of the polygon. We take the new vertex has to be closer to the center vertex than the two closest vertices of the two polygons neighbouring the new vertex. In Figure 1 the upper new vertex obeys this restriction while the one below it does not. Note that if it is desired that the new polygon be convex, the new vertices can be taken at a same distance from the center, the minimum of all distances computed.

If the new polygon is very small the area around it will again be finely decomposed, but this would result from high map complexity in the vicinity of the vertex. This is so because a new polygon is small only if there are many original vertices in the neighbourhood of the center vertex. In this case finer decomposition around the vertex is not objectionable, as this is the policy of hierarchical schemes.

The vertex enlargement method results in trees which are shorter than trees created using the modified splitting rule. Generally, the map complexity near a new polygon is reduced when the splitting line passes through it. This happens much sooner than when the center vertex becomes the only vertex in the cell.

The Simple and Convex Splitting Rules

In this section we detail two natural region based splitting rules, the *simple rule* and the *convex rule*, and in the next section a new one, the *convex differences rule*. The simple rule estimates the cell complexity according to the number of polygons in it. The convex rule is the same, but with a preprocessing that decomposes all polygons into convex pieces. For each of the rules we discuss its advantages and disadvantages, give an accompanying leaf structure and analyze its efficiency.

1. The Simple Splitting Rule

The simple splitting rule adopts the simplest criterion for the complexity of a cell. The number of polygons which fall in the cell serves as the complexity estimate. If this number exceeds the splitting threshold the cell is subdivided. As explained in the previous section, vertex enlargement ensures that the recursive splitting will stop. The resulting structure is called the *simple hierarchy (SH).*

Leaf cells are not organized in any special way. Every leaf stores a list of the polygons in it. To locate a polygon containing a specific point, the tree is traversed top down until the leaf that contains the point is found. Recall that the spatial subdivision is disjoint, so a point can be located only in regions which are sub-regions of a cell that contains the point. Then a sequential search for the point is made in all the polygons in the list. For every polygon, a point-polygon classification algorithm [Pre85] is performed. The algorithm takes time linear in the size of the polygon, so the cost of searching in the leaf is linear in the number of edges of the polygons in it.

As stated earlier, the number of edges in a cell can be much greater than the number of polygons in it. This means that the complexity measure was inaccurate: the number of polygons in a cell is not a good measure for the complexity of operations on the cell.

2. The Convex Splitting Rule

Operations on convex polygons are more efficient than operations on general or simple polygons. Point-polygon classification, intersection with a line, and intersection detection between two convex polygons take logarithmic time ([Cha87], [Rap89a]). Intersecting two convex polygons takes linear time [Oro82].

This motivates an improved splitting rule. As a preliminary step, all polygons are subdivided into convex parts. This is easy to achieve, e.g. by triangulation [1]. The complexity measure is again the number of polygons incident in the cell, and leaf cells hold a list of these polygons. This results in the *convex hierarchy (CH)*.

The advantage of the convex rule is that operations on a leaf are much faster. The number of polygons in a cell is now a more accurate measure for the complexity of the cell.

The disadvantage of the convex scheme is that the total number of polygons can be much greater after their subdivision into convex parts. In Figure 2 there are only two polygons. The lower one is convex. In every convex subdivision of the upper one (an example convex subdivision is shown dashed; ignore the dotted edge for the moment) the number of convex parts is at least the number of edges, because every edge participates in a different convex part.

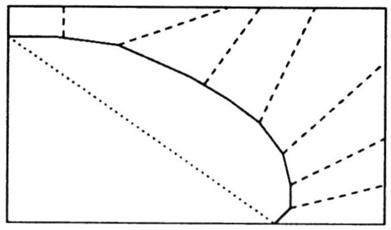

Figure 2: Two polygons (solid lines) and a convex decompostion (dashed lines). The dotted line is a CDT edge of the upper polygon.

Note that the convex decomposition can be deferred to the end of the process, with successive splittings if it results in more polygons than the threshold value. It is not clear which option is better.

The Convex Differences Hierarchy (CDH)

The *convex differences hierarchy (CDH)* results from the convex differences splitting rule and leaf organization method. The rule utilizes the efficiency of operations upon convex polygons without significantly deepening the hierarchy tree. It is based on the *convex differences tree (CDT)* representation for simple polygons [Rap89b], [Tor84]. A simple polygon is represented as a tree. the root of the tree contains the convex hull of the polygon. For every connected component of the difference between the convex hull and the polygon there is a son node (child) of the root. This difference is viewed as a simple polygon, to be represented recursively by the child node. An example is given in Figure 3.

[1]Algorithms for optimal convex decomposition with and without the introduction of new vertices (Steiner points) are given in [Cha85] and [Gre83] respectively.

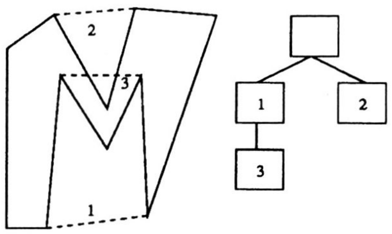

Figure 3: An example of a polygon and its CDT.

```
PointClassify(p,N)
    /* p is a point, N is a node, initially the root */
    if p is outside the convex hull of N
        return OUT;
    for all sons of N
       if PointClassify(p,son) = IN
          return OUT;
    return IN;
```

Figure 4: Point-polygon classification with the CDT.

In a previous work [Rap89b] we showed that the space occupied by the CDT is linear in the number of vertices of the polygon. An efficient $O(k \log k)$ algorithm for converting to the CDT representation from the vertex list representation was given, where k is the number of edges of the polygon. It was also shown how to perform commonplace operations on the CDT. These operations include point classification, segment intersection, Boolean operations between two polygons and sorting of the vertices in an arbitrary direction.

The operations are done by performing a tree traversal on the CDT, calling efficient convex polygon algorithms to operate on the nodes. For example, an algorithm for point-polygon classification with the CDT is given in Figure 4. The algorithm uses the fact that a point is inside a node of the CDT if and only if it is inside the convex hull of the node and outside of all the children of the node.

For certain families of polygons, some of the operations were shown to be much more efficient than when using the standard vertex-list representation.

These families include polygons which obey one or both of the following conditions. The *bounded depth condition* is obeyed when the depth of the CDT is bounded by a small constant (e.g. 5). In general, the depth of a CDT can be linear in the number of vertices of the polygon [Rap89b], but it takes highly complicated shapes, such as fractals developed into deep recursion, to achieve that.

The *bounded node condition* is obeyed whenever the number of nodes in a CDT is bounded by a constant. This condition is much stronger than the previous one, since every level of the CDT can have many nodes. Many of the polygons which arise in computer graphics or computer aided design obey this condition.

For polygons which obey the bounded depth condition, the construction time of the CDT is linear in the number of vertices. For polygons which obey the bounded node condition, the operations of point-polygon classification and segment-polygon intersection require logarithmic time [Rap89b].

The reason for the efficiency of the operations on the CDT is two-fold. The polygon is represented in a hierarchy, which enables fast pruning of redundant paths. For example, if the point to be classified is outside of the convex hull of the polygon, this will be detected at the first step of testing against the root of the CDT. The second reason is that each level of the hierarchy is represented by a convex polygon, which permits very efficient operations.

We can now explain the convex differences splitting rule and leaf structure. As a preliminary step we perform vertex enlargement, the replacement of high degree vertices by polygons around them. The decomposition process is initiated as in the simple scheme, with the splitting rule being the number of polygons in a cell (any decomposition process can easily produce this number). When this number is under the splitting threshold T, the CDT representation of all polygons in the cell is computed. Now the decomposition process is resumed, the splitting threshold being *the total number of CDT nodes in the cell*. Decomposing a cell entails a recalculation of the CDT's of all polygons in the new sub cells. For leaf nodes we again keep a list of all polygons in them, similar to the simple and convex schemes, but represented as CDT's.

Note that the number of CDT nodes is never smaller than the number of polygons, because every polygon has at least one CDT node. Hence, it never happens that a cell should be split according to the first (simple) rule and is found to be too simple in the second (convex differences) rule. There is no need to reunite decomposed cells.

When the original map is convex, i.e. all the participating polygons are convex, the three splitting rules coincide, because a polygon is convex if and only if its CDT has one node.

As an example for a CDH see Figure 2. The figure shows one cell which contains two polygons. The lower one is convex and its CDT has one node. The CDT of the upper polygon has two nodes, the root containing the convex hull of the polygon (it has one edge not belonging to the polygon, shown dotted), and a child node containing the part of the lower polygon bounded by the dotted edge. The total number of polygons belonging to the CDH of the cell is three, all of them convex. Note that they are not disjoint. As noted earlier, the number of polygons in a convex decomposition of the cell is arbitrarily large.

Comparison

We have already noted that for general hierarchical schemes the splitting criterion is independent of the nature of the decomposition. Its major effect is on the depth of the tree. The leaf organization policy, which is somehow related to the splitting rule, affects the time to perform the desired operation on the leaf. Hence, the only parameters we need for comparing the presented splitting rules, apart from the splitting threshold T, are the depth of the hierarchy tree and the average number of edges in a polygon after the decomposition.

1. Depth of the Hierarchy Tree

Among the three rules, the simple rule results in the shortest trees. This is a consequence of the already stated fact that the number of CDT nodes and the number of convex parts of a cell are not smaller than the number of polygons in it.

Next in depth of its trees is the convex differences hierarchy. It is hard to determine exactly how much deeper are its trees compared to the simple rule. This depends on how far the polygons are from being convex and how noisy are their edges. If the polygons obey the bounded node condition there is not much of a difference. If they obey the bounded depth condition we can expect the difference to be a small constant. Note that considering the total number of CDT nodes in the splitting rule results in averaging the effect of complex shaped polygons with this of simpler polygons in the cell.

The deepest trees result from convex decomposition. In Figure 2, the CDT has three nodes as compared to an arbitrarily large number of convex parts. This phenomenon is very common in maps whose edges are a result of an approximation process.

2. Storage and Construction Time

The arguments concerning storage are the same as those for the depth of the tree. The best rule is the simple rule and the worst is the convex rule. The storage occupied by a CDT is linear in the number of polygon vertices, and the depth of the CDH is close to that of the SH. Hence the storage requirements of the CDH are not much greater than those of the SH, and are certainly smaller than those of the CH.

Regarding construction time, the arguments are less clear. A convex decomposition takes time, but it enables the decomposition process to be carried more efficiently by using the logarithmic time algorithm for intersecting a line against a convex polygon [Rap89a]. This is also true for the CDT. However, we discuss static maps, and construction time is the least important factor of our comparison.

3. Point Location in a Leaf

In all three schemes, point location is done by first traversing the hierarchy tree to reach the cell containing the given point and then searching the list of polygons incident in the cell and classifying the point against each. The time to reach the cell is a function of the depth of the tree, which we have already discussed.

The splitting threshold T gives an upper bound on the number of point-polygon classifications. Denote by e, e_c, e_{cd} the average number of edges of a polygon in a cell in the simple, convex, and convex differences hierarchies respectively.

In the simple rule, classifying a point against a polygon is done by firing a ray from the point to infinity and counting the number of intersections with the edges of the polygon [Pre85]. This costs $O(e)$ operations. Hence the cost of point location in a leaf is $O(Te)$.

In the convex rule, the cost of point location is $O(T \log e_c)$, because of the logarithmic time algorithm for classifying a point against a convex polygon[Pre85], [Rap89a]. e_c is expected to be much smaller than e. In the case of triangulation, $e_c = 3$, the cost of the classification against a single polygon is constant, and the point location cost is $O(T)$.

In the convex differences hierarchy, the worst case cost of point location is $O(T \log e_{cd})$, but there is a potential saving due to the hierarchical nature of the CDT representation of a polygon. Specifically, if the point was found to be outside of a node of the CDT, there is no need to test for inclusion in any of its descendants. e_{cd} is also smaller than e, but larger than e_c.

In summary, the point location operation is most efficient in the CH, with no significant difference from the CDH. It is most costly in the SH.

Note that in all methods, if the point is located within a new polygon (one which has resulted from the vertex enlargement preprocess), the point should be classified against the neighbourhood of the center vertex using the original data. This can be done in time logarithmic in the number of edges meeting at the center vertex using binary search for the angle, as shown in [Rap89a].

4. Conclusion

Many maps are obtained by manual design, approximation, or sampling of smooth data. In such cases, the polygons in the map are not very complex, i.e. their CDT's have a limited number of nodes and a bounded depth. Hence, the depth of the hierarchy tree resulting from the CD splitting rule is almost the same as the depth resulting from the simple rule. The operations on a leaf are much more efficient, especially when e, the average number of edges per polygon, is large. In this case the CDH is superior to the SH.

The convex splitting rule results in much deeper hierarchy trees occupying a lot more storage. It enables fast operations on the leaves, but there is not much of a difference from leaf operation time in the convex differences scheme. As a result the CD rule is better than the convex rule, since the depth of the tree becomes the important factor.

The advantage of the CDH diminishes when dealing with maps whose polygons have very complicated boundaries, e.g. fractals. In this extreme case the depth of the CDT of a polygon, as well as the number of polygons in a convex decomposition, are large, and we can expect the SH to be better than the CH and the CDH.

To focus the discussion, point location was the only operation dealt with. The nature of the conclusions holds for other operations also. Most operations done on polygons possess efficient algorithms when the polygons are convex. As a result, the leaf organization in the convex and convex differences rules facilitates more efficient operations than this of the simple rule, while the depth of the CD trees is smaller than this of the convex trees.

Future Research

An obvious direction to continue this research is to extend our results to dynamic situations. In particular, vertex enlargement should be adapted to changing maps.

The CDT cell organization can be applied to non-adaptive schemes simply by computing the CDT's of all regions in the cell. Operations on the cell are more efficient than when keeping the regions in the vertex list representation. Of course, we no longer have control on the number of CDT nodes in the cell because non-adaptive schemes do not decompose cells.

An algorithm to efficiently split a CDT along a line can accelerate the CDH construction time. Such an algorithm is useful at the stage when we already have the CDT's of all polygons in the cell and it has to be decomposed.

We have not compared our region based approach to edge based approaches. Edge based approaches generally decompose the space finer than region based approaches. The choice between them probably depends on the nature of the data. Various experiments with large size real data should be performed.

Acknowledgements

I want to sincerely thank Prof. Hanan Samet, who motivated this research, commented on a preliminary version, and provided continuous support.

References

[Ben75] Bentley J.L., Multidimensional binary search trees used for associative searching, CACM 18(9):509-517, 1975.

[Cha85] Chazelle, B., Dobkin, D., Optimal convex decompositions, in: Computational Geometry, G. Toussaint (editor), North-Holland, 1985.

[Cha87] Chazelle, B., Dobkin, D., Intersection of convex objects in two and three dimensions, *Journal of the ACM* 34(1):1-27, 1987.

[Eda84] Edahiro, M., Kokubo, I., Asano, T., A new point location algorithm and its practical efficiency – comparison with existing algorithms, *ACM Transaction On Graphics*, 3(2):86-109, 1984.

[Ede86] Edelsbrunner, H., Guibas, L.J., Stolfi, J., Optimal point location in a monotone subdivision, *SIAM Journal on Computing*, 15(2):317-340, 1986.

[Ede87] Edelsbrunner, H., Algorithms in Combinatorial Geometry, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, 1987.

[Fra88] Franklin, W.R., Akman, V., An adaptive grid for polyhedral visibility in object space: an implementation, *Computer Journal*, 31(1):56-60, 1988.

[Fuc80] Fuchs H., Kedem, Z.M., Naylor, B.F., On visible surface generation by a priority tree structure, *Computer Graphics* 14(3):124-133, 1980 (SIGGRAPH '80).

[Gre83] Greene, D.H., The decomposition of polygons into convex parts, in: Advances In Computing Research, F. Preparata (editor), vol.1, JAI Press Inc., London England, 1983.

[Har69] Harary, F., Graph Theory, Addison-Wesley, Reading, Mass. 1969.

[Kay86] Kay, T.L., Kajiya, J.T., Ray tracing complex scenes, *Computer Graphics* 20-(4):269-278, 1986 (SIGGRAPH '86).

[Kir83] Kirkpatrick, D., Optimal search in planar subdivisions, *SIAM Journal on Computing*, 12(1):28-35, 1983.

[Nie84] Nievergelt, J., Hinterberger H., Sevcik, K.C., The grid file: an adaptable, symmetric multikey file structure, *ACM Transaction on Data Base Systems*, 9(1):38-71, 1984.

[Oro82] O'Rourke, J., Chen, C.B, Olson, T, and Naddor, D., A new linear algorithm for intersecting convex polygons, *Computer Vision, Graphics and Image Processing*, 19:384-391, 1982.

[Pre85] Preparata, F., Shamos, M. I., Computational Geometry: An Introduction, Springer-Verlag, New-York, 1985.

[Rap89a] Rappoport, A,. An efficient algorithm for line and polygon clipping, Technical Report 89-27, Computer Science Department, The Hebrew University, 1989.

[Rap89b] Rappoport, A., The convex differences tree representation for simple polygons and its use in computer graphics, Technical Report 89-28, Computer Science Department, The Hebrew University, 1989.

[Sam89a] Samet, H., The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, Mass. 1989.

[Sam89b] Samet, H., Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Addison-Wesley, Reading, Mass. 1989.

[Tor84] Tor S. B., Middleditch A. E., Convex decomposition of simple polygons, *ACM Transactions on Graphics*, 3(4):244-265, 1984.