Using Quadtrees to Represent Polygonal Maps

H. Samet
R. E. Webber

Computer Science Department and
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

The quadtree data structure is adapted to represent polygonal maps. The presentation is motivated by the problem of point-in-polygon determination. The goal is to store these maps with absolute accuracy and obtain a worst-case execution time that is not overly sensitive to the positioning of the map when a quadtree-like representation is used. A regular decomposition variant of the region quadtree is used to organize the vertices and edges of the maps. A number of appropriate data structures are proposed in an iterative manner until a structure is obtained that meets the stated goals. The result is termed a PM quadtree and is based on a regular decomposition point space quadtree (PR quadtree) that stores additional information about the edges at its terminal nodes.

1 Introduction

Region representation is an important problem in image processing, computer graphics, computerized cartography, and related areas. Recently there has been much interest in hierarchical data structures such as the quadtree (see the overview in [9]). Most of this work has involved regions that are composed of a collection of squares. In this paper we address the problem of representing regions made up of collections of arbitrary polygons. It is useful in cartographic applications (e.g., representing county boundaries, etc.) and also in a host of other disciplines that require partitioning of the plane into regions whose boundaries consist of line segments (e.g., Voronoi diagrams [12]).

In this paper we show how a quadtree-like data structure can be adapted to represent a polygonal map. Our presentation is an evolutionary one in the sense that we start with one data structure and continually refine it until it meets all of our requirements in a satisfactory manner. In general, it is difficult to evaluate a data structure in a vacuum. Thus we shall use the task of point-in-polygon determination on a dynamically changing polygonal map to motivate our discussion and selection of a representation. This task is performed under the assumption that each line segment that forms the boundary of a region contains the names of the regions on both sides of it (as well as which region is on which side). This reduces the point-in-polygon task to the less restricted problem of locating a line segment that borders the region containing the query point.

2 General Considerations

In the following we consider polygonal maps in their most general form, i.e., as a planar embedding of a graph where all the edges are represented by non-intersecting straight line segments. Thus, the polygonal map is a hybrid object in the sense that we are representing three types of data - i.e., points, lines, and regions. Point data are represented by isolated vertices. Linear data are represented by acyclic subgraphs. Region data are represented by cyclic subgraphs that partition the plane. The region labeling of a polygonal map that is viewed as a graph is associated with the vertices and the edges comprising the graph.

The term quadtree [2,4,5] is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition. In two dimensions, a rectangular planar region is recursively subdivided into four parts until each part contains data that is sufficiently simple so that it can be organized by some other data structure (e.g., a single element list, binary tree, etc.). They can be differentiated on the basis of the type of data that they are used to represent, and on the principle guiding the decomposition process. The decomposition may be into equal sized parts (termed a regular decomposition) or it may be arbitrary (i.e., dictated by the input data).

The point quadtree [2] is a decomposition based on the point data being represented. It is a multidimensional analog of the binary search tree [6] where the root node corresponds to the first node that was inserted into the tree. The region quadtree [4,5] is used to represent spatial data. It makes use of a regular decomposition. Subdivision is performed until a square region is obtained which is homogeneous. Edge quadtrees [11] use regular decomposition to represent curves. A region is subdivided repeatedly until the leafs represent regions containing a single curve that can be approximated by a straight line. The line quadtree [10] is similar to the edge quadtree except that a region is repeatedly subdivided (using regular decomposition) until the leafs represent regions that have no line segments passing through their interior. There is also an edge quadtree variant [8] that subdivides, again employing regular decomposition, until the leafs represent regions containing a single curve that can be approximated by k straight lines (where k has been fixed a priori).

The region quadtree can also be adapted to represent point data. We term such a tree a PR quadtree (PR denoting point region). In this case all points are associated with terminal nodes. Regular decomposition is applied until no quadrant contains more than one data point. For example, Figure 2 is a PR quadtree representation of the five vertices of the polygonal map in Figure 1. Note that we have left the edges in the figure. In order to cope with vertices that lie directly on one of the quadrant lines emanating from a subdivision point, we adopt the convention that quadrants NE and SE are closed with respect to the x coordinate and quadrants NW and NE are closed with respect to the y coordinate.

We use the PR quadtree because the shape of point quadtrees is very sensitive to the order in which the data points are inserted into the tree. There does not exist a dynamic balancing algorithm for quadtrees that is analogous to the AVL algorithm [1] for binary trees. In addition, PR quadtrees yield an average-case balancing under the assumption that future data is equally likely in any of the four quarters (called quadrants) of the square.

In our adaptation of the PR quadtree for storing polygonal maps, we view the vertices as being enclosed in a unit square (i.e., both their coordinates lie between 0 and 1). The decomposition criterion is termed C1 and defined below:

C1: At most one vertex can lie in a region represented by a quadtree leaf.

Once we have defined our data structure, the task of point-in-polygon determination is essentially a tree search. Its worst-case execution time depends on the maximum tree depth which is, in turn, a function of the input polygon. For the PR quadtree, the maximum tree depth is obtained as follows. Recall that the polygonal map is embedded in a unit square. As the depth of the PR quadtree increases, the maximum separation between two points in the same node is halved. The maximum separation between any two points in the unit square is the square root of 2. Points that are this far apart require a tree with depth 1 to separate them. Generalizing this observation we have that when vv is the minimum separation between two distinct vertices, then an upperbound on the depth of the corresponding quadtree is

$$D1 = 1 + \log_2 (SQRT(2) / vv)$$

Instead of adapting the PR quadtree, we could have also considered using one of the line quadtree schemes of [10]. This is unsatisfactory for several reasons. First, line quadtrees store only an approximation of a map and certain properties of polygonal maps cannot be directly represented (e.g., it is impossible for five line segments to meet at a vertex because the line quadtree is made up of a collection of square-like regions). Second, shifting or rotating a line quadtree leads to a possible loss of accuracy with respect to the map that was originally approximated. Third, line quadtrees may require much space. For example, Figure 3 is the line quadtree corresponding to the polygonal map of Figure 1. Note that although Figure 1 consists of just five vertices and six edges, its line quadtree requires 105 leaf nodes.

3 Representing Polygonal Maps

In this section we discuss the adaptation of the PR quadtree to represent a polygonal map. During the presentation we will use the term PM quadtree to refer to the evolving structure. We shall use the polygonal map of Figure 1 as a common example to illustrate various claims that we make. Note that the map of Figure 1 partitions the plane into 3 regions which are labeled 1, 2, and 3. In the PM quadtree structure it is assumed that the region labels are associated with the edges. For example edge AD is marked to indicate that region 2 lies to the right of AD and region 1 lies to the left of AD where right and left are with respect to a vantage point at the origin (i.e., the lower left corner of the enclosing square) and lines are viewed as if they have been extended to intersect the enclosing square. In our discussion, we frequently need to refer to segments of edges of the the graph. In particular, we use the term q-edge to refer to a segment of an edge that spans an entire block (e.g., RS in Figure 2) or all of the block of which its corresponding edge is a member (e.g., ER in Figure 2). For example, edge EB consists of the q-edges ER, RS, ST, and TB.

3.1 The Angular Ordering of Edge Segments About the Point Where They Meet

A criterion analogous to C1, called C2, which takes edges into account is given below.

C2: At most one q-edge can lie in a region represented by a quadtree leaf.

Unfortunately C2 is inadequate because there exist polygonal maps that would require a PM quadtree of infinite depth to satisfy it. For example, consider vertex E and q-edge ER and EU in Figure 2 (i.e., the PR quadtree of the polygonal map of Figure 1). Assume that the x and y coordinates of E cannot be expressed as an exact base 2 fraction. This means that E can never be a subdivision point. Thus no matter how many times we subdivide the quadrant containing E, ER, and EU, by the continuity of the q-edges, there will always exist a pair of infinitesimally small q-edges EH and EI which will occupy the same quadtree leaf. Vertices at subdivision points may at times avoid the infinite depth problem by virtue of the conventions adopted in Section 2 with respect to their placement.

One solution to this problem lies in replacing C2 with criteria C2′ and C3 given below.

C2′: If a region contains a vertex, then it can contain no q-edge that does not include that vertex.
C3: If a region contains no vertices, then it can contain at most one q-edge.

A PM quadtree built from the criteria C1, C2′ and C3, representing the polygonal map of Figure 1, is shown in Figure 4.

Since criterion C2′ allows an arbitrary number of q-edges to be stored at one PM quadtree leaf, the question of how these q-edges are organized arises. The simplest approach, consistent with our interest in worst-case analysis, is to store the q-edges in an AVL tree where the q-edges are ordered by the angle that they form with a ray originating at the vertex and parallel to the positive x-axis. Since the number of q-edges passing through a leaf is bounded from above by the number of vertices belonging to the polygonal map, say V, the depth of the AVL tree is proportional to

$$A1 = \log_2 (V)$$

It is appropriate at this point to recall our task of point-in-polygon determination. For PM quadtrees built from C1, C2′, and C3, this problem has three cases which are illustrated by queries with respect to the points x, y, and z in Figure 4.

Point x illustrates the problem of point-in-polygon determination when the point lies in a leaf containing exactly one q-edge. Since region information is stored at each q-edge indicating the regions associated with the q-edge, this reduces to determining the side of the q-edge on which the point lies.

Point y illustrates the case that the query point lies in a leaf containing a vertex, C in this example. This reduces to finding a q-edge in the AVL tree that would neighbor a hypothetical q-edge from C and passing through y. Such a neighboring q-edge must border the region containing y. Thus, once again our task is reduced to determining on which side of a q-edge a point lies (i.e., y).

Point z illustrates the case that the query point lies in a leaf, say q, containing no q-edges. This means that all the points in the region represented by the leaf q lie in the same region of the polygonal map. It also means that one of q′s brothers must be the root of a subtree that contains a q-edge that borders the region containing z. In order to find this (not necessarily unique) brother, we move clockwise among the brothers of q. This prevents us from prematurely considering the diagonally neighboring brother of q (see why in next paragraph). When considering a brother, one of two subcases arises.

Either q′s clockwise neighboring brother, say r, (1) contains a q-edge, say b, lying on the boundary between q and r or (2) it doesn′t. In subcase (1), the problem reduces to determining the side of q-edge b on which z lies. Subcase (2) is slightly more complex. We postulate a hypothetical point z′ in region r that is infinitesimally close to q′s region and recursively reapply the point-in-polygon procedure to z′. Figure 5 shows why we don′t want to prematurely consider a diagonal brother. In this case placement of our hypothetical point z′ in the SE brother of the quadrant containing z will lead us to conclude that z lies in region 2 rather than region 1 by virtue of its relative position to edge segment ST which is the only edge segment in the quadrant. Note that point R is associated with the NE brother of the quadrant containing z by virtue of the conventions adopted in Section 2 with respect to points that lie on quadrant lines emanating from subdivision points.

As an example of the case that z lies in a leaf containing no q-edges, consider Figure 4. Since the leaf containing z, call it q, is empty, we examine its clockwise brother, say r. Since r does not contain a q-edge on the boundary between q and r, subcase (2) applies. Thus we postulate a point z′ that is just across the boundary between q and r. Determining the polygon in which lies z′ (in this example) is equivalent to determining the polygon in which x lies. Note that in subcase (2), if r contains no q-edges, then the algorithm proceeds to examine r′s clockwise brother. It should be clear that one of the brothers must contain a q-edge as otherwise the brothers would have been merged to yield a larger node.

The worst-case execution time of point-in-polygon determination using a PM quadtree constructed with criteria C1, C2′, and C3 is proportional to the depth of the entire structure - i.e., the depth of the quadtree built from C1, C2′, and C3 plus A1, the maximum depth of the AVL trees at the quadtree leafs. The depth of the quadtree can be determined as the maximum of the depth required independently by each of the three criteria for building the quadtree. The factor contributed by criterion C1 has already been noted in Section 2 to be D1. If ev denotes the minimum separation between an edge and a vertex not on that edge (for a given polygonal map), then by reasoning similar to the derivation of D1, the depth of the PM quadtree required to fulfill criterion C2′ is

$$D2' = 1 + \log_2 (\mathrm{SQRT}(2) / ev)$$

Analogously, if ss denotes the minimum separation between two non-intersecting q-edges (i.e., portions of edges bounded by either a vertex or the boundary of a PM quadtree leaf of the PM quadtree of the given polygonal map), then the PM quadtree depth required to fulfill criterion C3 is

$$D3 = 1 + \log_2 (SQRT(2) / ss)$$

Although the factors D1 and D2′ are functions of the polygonal map and are independent of the positioning of the underlying digitization grid, the factor D3 can vary as the polygonal map is shifted. For maps of the complexity of the one shown in Figure 1, the D3 factor can become arbitrarily large. For example, suppose we shift the polygonal map in Figure 1 to the right. As vertex E (see Figure 2) gets closer to the eastern boundary of the quadrant containing it, the minimum separation between q-edges RS and UV (i.e., RU) gets smaller and smaller resulting in the growth of D3 to unacceptable values. While this is better than the impossibility associated with C2, it still behooves us to find a better decomposition criterion than C3.

3.2 The Angular Ordering of Edge Segments About the Points Where Their Edges Meet

In order to remedy the deficiency associated with criterion C3, it is necessary to determine when it dominates the cost of storing a polygonal map. In particular, D3 is greater than D2′ only if ss is smaller than ev, which happens only when the two nearest non-intersecting q-edges are segments of edges that intersect at a vertex. For example, Figure 6 is the PM quadtree for polygonal map ABCD where D3 is greater than D2′, because ss (the distance between q-edges XY and WZ) is smaller than ev (the distance between C and BD). Note that XY is a q-edge of BD, WZ is a q-edge of CD, and BD intersects CD at vertex D. This analysis leads us to replace criterion C3 with criterion C3′ defined below.

C3′: If a region contains no vertices, then it can contain only q-edges that meet at a common vertex exterior to the region.

A PM quadtree built from criteria C1, C2′, and C3′, for the polygonal map of Figure 1, is shown in Figure 6.

Substitution of C3′ for C3 does not lead to significant changes in the point-in-polygon determination procedure. The situation arising when q-edges are ordered about a point exterior to their region is handled in the same way as q-edges that are angularly ordered about their point of intersection. Of course, it is necessary to store with each AVL tree the point about which the ordering is being performed.

The worst-case execution time of the point-in-polygon algorithm is again proportional to the sum of the depth of the quadtree plus A1, the maximum depth of the AVL trees. However, the depth of the quadtree is bounded from above by the maximum of D1 and D2, the factors attributed to criteria C1 and C2′ respectively. Note that by virtue of our definition of C3′, the maximum depth resulting from its use is bounded from above by D2′.

As an example, consider Figure 8, which represents the same polygonal map as Figure 7 except that it uses C3′ instead of C3. The analogue of ss, termed ss′, is defined as the minimum separation between two q-edges that are not segments of two intersecting edges. In this example, D3′ is less than D2′ because ss′ (the distance between UB and SC) is greater than ev (the distance between C and DB). Note that the distance between QR and ST, and the distance between RB and TC are irrelevant to D3′, because, if necessary, these segments could be in the same leaf.

We have now achieved a structure for which point-in-polygon determination has a worst-case execution time that is less sensitive to shift and rotation of the polygonal map. The only question that remains is whether we can do better. Can the factor of criterion C2′ be removed or reduced?

3.3 Ordering Edge Segments With Respect To Their Boundary Intercepts

In this section, we consider a PM quadtree built using only criterion C1, but that could represent any polygonal map. For this version of the PM quadtree we revert to the original PR quadtree (e.g., Figure 1) except that more information is stored at each terminal node corresponding to a vertex of this map. Since the depth factor D1 is always less than or equal to the factor D2′, the quadtree component of the worst-case execution time of point-in-polygon determination is lower than in our previous structures. However, this structure does have the problem that the number of q-edges that can be stored in a leaf is now bounded by the number of q-edges in the graph, instead of the number of vertices. This does not affect the order of the worst-case execution time of point-in-polygon determination, because, in a planar graph, the number of edges is bounded from above by a linear function of the number of vertices (this is a corollary of Euler′s formula [3]).

There still remains the problem of how to organize the q-edges in a leaf′s region and how such an organization interacts with point-in-polygon determination. We propose to partition the q-edges in a leaf′s region into 7 classes, each of which can be ordered by an AVL tree. Note that in any given leaf, some of these classes will often be empty.

The most obvious class of q-edges is the class of q-edges that meet at a vertex within the leaf′s region. This class can be ordered in an angular manner as has been done previously. The remaining q-edges that pass through the leaf′s region must enter at one side and leave via another. This yields six classes: NE, NS, NW, EW, SW, and SE, where NE denotes q-edges that intersect both the North and the East boundary of the leaf′s region. Note that the q-edges are non-directional. For example, the q-edges in class NE (the other 5 classes are handled analogously) are ordered according to whether they lie to the left or to the right of each other when viewing them in an easterly direction from the northern boundary of the leaf's region. Q-edges that coincide with the border of a leaf's region are placed in either NS or EW as is appropriate. Note that any given leaf's boundary can only contain one such q-edge, because if it contained two, then it would have to contain two vertices and violate C1. Point-in-polygon determination is accomplished by finding a bordering q-edge with respect to each of the seven classes and then using the closest of the seven as the true bordering q-edge.

4 Concluding Remarks

We have taken an iterative approach to developing a quadtree-like data structure for storing polygonal maps. We started with the PR quadtree and developed the PM quadtree. Its final formulation uses the same decomposition rule as the PR quadtree but stores a considerable amount of information in the terminal nodes. Note that the PM quadtree enables storing polygonal maps with absolute accuracy and for whom point-in-polygon determination has a worst-case execution time that is less sensitive to the positioning of the polygonal map. Thus unlike the region and/or line quadtrees of [10], the quadtrees presented in Sections 3.2 and 3.3 can be shifted or rotated without distortion or unreasonable change in the size of the structure. Note that the storage requirement are still somewhat dependent on the positioning of the space within which the map is embedded. For example, the polygonal map of Figure 9a requires 7 PR quadtree leafs, while Figure 9b requires only 4 PR quadtree leafs. We also observe that our proposed quadtrees are relatively compact. As a comparison, we note that Figure 3 required 105 quadtree leafs, whereas Figure 7 required 13 quadtree leafs and 21 AVL data nodes (scattered among 11 AVL trees), and Figure 2 (interpreted as a PM quadtree) required 7 quadtree leafs and 17 AVL data nodes (scattered among 9 AVL trees). Note that many of the AVL trees consist of single data nodes.

We have also shown that point-in-polygon determination using the PM quadtree can be done in time proportional to the depth of the structure. While it is true that others (e.g., [6]) have shown that structures exist that have better worst-case analysis for the point-in-polygon task than the PM quadtree described above, the PM quadtree has a major advantage over other structures in that it organizes the data without a dimensional bias. Thus PM quadtrees are generally better for range queries, whose analysis is a natural extension of the above discussion of point-in-polygon for PM quadtrees. The performance analysis of other operations, e.g., shift, rotation, and overlay, has yet to be done.

References

[1] G. M. Adelson-Velskii and Y. M. Landis, An algorithm for the organization of information, Soviet Math. Dokl. 3, 1962, 1259 - 1262.

[2] R. A. Finkel and J. L. Bentley, Quad trees: a data structure for retrieval on composite keys, Acta Informatica 4, 1974, 1 - 9.

[3] F. Harary, Graph Theory, Addison-Wesley Publishing Company, Reading, MA, 1969.

[4] Hunter, G. M. and Steiglitz, K., Operations on images using quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 1979, 145-153.

[5] A. Klinger, Patterns and search statistics, in Optimizing Methods in Statistics, J. S. Rustagi, Ed., Academic Press, New York, 1971.

[6] D. E. Knuth, The Art of Computer Programming: vol. 1/ Fundamental Algorithms, Second Edition, Addison-Wesley, Reading, MA, 1975.

[7] D. T. Lee and F. P. Preparata, Location of a point in a planar subdivision and its applications, Proc. of Eighth Ann. ACM Symp. on Theory of Computing, Hershey, PA, May 1976, 231 - 235.

[8] J. J. Martin, Organization of geographical data with quadtrees and least squares approximations, Proc. Conf. Pattern Recognition and Image Processing, Las Vegas, 1982, 458 - 463.

[9] H. Samet and A. Rosenfeld, Quadtree structures for image processing, Proceedings of Fifth International Conference on Pattern Recognition, Miami Beach, December 1980, 815 - 818.

[10] H. Samet and R. E. Webber, Line quadtrees: a hierarchical data structure for encoding boundaries, Proc. Conf. Pattern Recognition and Image Processing, Las Vegas, 1982, 90 - 92.

[11] M. Shneier, Two hierarchical linear feature representations: edge pyramids and edge quadtrees, Computer Graphics and Image Processing 17, November 1981, 211 - 224.

[12] G. T. Tousssaint, Pattern recognition and geometric complexity, Proceedings of Fifth International Conference on Pattern Recognition, Miami Beach, December 1980, 1324 - 1347.

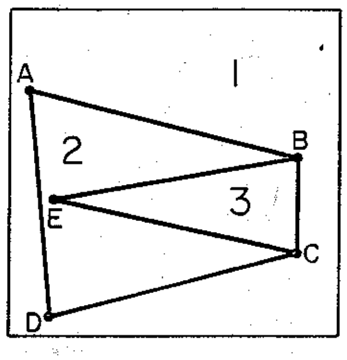

Figure 1: Sample polygonal map.