

THE SEGMENT QUADTREE:  
A LINEAR QUADTREE-BASED REPRESENTATION FOR LINEAR FEATURES\*

Hanan Samet  
Clifford A. Shaffer

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

Robert E. Webber

Computer Science Department  
Rutgers University, Busch Campus  
New Brunswick, New Jersey 08903

ABSTRACT

A new linear quadtree-based representation for linear features termed a *segment quadtree* is presented. It uses a fixed amount of storage per node, represents straight lines exactly (not a digitized representation), and enables updates in a consistent manner.

1. INTRODUCTION

The region quadtree representation [4,7] has gained extensive use in image processing, computer graphics, automated cartography, and other fields. It is a hierarchical data structure that is based on the principle of regular decomposition. Due to the large amounts of data involved in such applications, it is useful to use a representation for the quadtree that does not involve pointers. The linear quadtree [2] is one such representation that stores the image as a collection of leaf nodes each of which is encoded by a number corresponding to a sequence of directional codes that locate the leaf along a path from the root of the quadtree.

We are interested in using the linear quadtree as a uniform representation for data corresponding to regions, points, and linear features. The uniformity facilitates the performance of set operations such as intersecting a linear feature with a region within an integrated database. While there are several hierarchical data structures for linear features, they are either not based on regular decomposition (e.g., the strip tree [1]), do not handle more than one linear feature (e.g., the strip tree), or do not cope with the intersection of linear features (e.g., the edge quadtree [10]). For linear features, a good linear quadtree representation must also have the following three properties. First, it must use a fixed amount of storage per node. This rules out the PM quadtree [6]. Second, straight line segments should be represented exactly (not by a digitized representation). Third, updates must be consistent, i.e., when a linear feature is deleted, the data base is restored to a state identical to that which would have resulted if the deleted linear feature had never been added. The line quadtree [8] is eliminated because it only handles rectilinear linear features. Hunter and Steiglitz's [3] adaptation of the quadtree for polygons is likewise of no use since it only approximates the lines. The edge quadtree [10] was designed primarily for representing curves and thus does not handle connected edges in a manner that permits consistent updates.

\* The support of the U. S. Army Engineer Topographic Laboratory under contract 70-81-C-0059 is gratefully acknowledged.

In order to meet the above requirements, we present a new data structure, termed a *segment quadtree*. We show how it can be used in conjunction with linear quadtrees to represent collections of linear features consisting of sets of connected straight line segments termed *polygonal maps*. In particular, we discuss in detail the insertion and deletion of line segments.

2. ALTERNATIVE LINE REPRESENTATIONS

In the *edge quadtree* of Shneier [10], a region containing a linear feature, or part thereof, is repeatedly subdivided into subquadrants until each quadrant contains at most one curve that can be approximated by a single straight line segment. Applying this process leads to quadtrees in which long straight edges can be stored in a few large leaves. However, small leaves are required in the vicinity of corners, intersecting edges, close approaches between curves, or areas of high curvature. As an example of the decomposition that is imposed by the edge quadtree, consider Figure 2 which is the edge quadtree corresponding to the polygonal map of Figure 1 when represented on a  $2^4$  by  $2^4$  grid. A serious drawback of the edge quadtree is its inability to handle the meeting of two or more edges at a single point (i.e., a vertex) except as a pixel corresponding to an edge of minimal length. This means that boundary following as well as deletion of line segments cannot be properly handled in the vicinity of a vertex at which more than one edge meets.

Another quadtree variant which is closely related to the edge quadtree is the formulation of Hunter and Steiglitz [3], termed an *MX quadtree* in [7]. It considers the border of a region as separate from either the inside or the outside of that region. Figure 3 shows the MX quadtree corresponding to the polygonal map of Figure 1. The MX quadtree has problems similar to those of the edge quadtree in handling vertices. Again, a vertex is represented by a single pixel. Furthermore, note that the edge quadtree of Figure 2 contains considerably fewer nodes than the quadtree of Figure 3.

The *linear edge quadtree* is a variant of the edge quadtree that has been adapted for incorporation in a geographic information system [9]. In this scheme, the leaf nodes of the quadtree are stored in a list (maintained by a B-tree) in the order in which they would have been visited by a preorder traversal of the tree. Each node contains three fields: an address, a type, and a value field. The address field describes the size of the node and the coordinates of one of the corners of its corresponding block. The type field indicates whether the node is empty (i.e., WHITE), contains a single point, or contains a line segment. Unlike Shneier's [10] formulation, a line

segment may not end within a node since in the existing implementation the value field is not large enough to contain the location of an interior point as well as a slope. Thus endpoints and intersection points are represented by single pixel-sized point nodes. Vertices are represented by pixel-sized nodes with the degree of the vertex stored in the value field. The value field of a line segment contains the coordinates of its intercepts with the borders of its containing node. Figure 4 illustrates the linear edge quadtree representation. Note the difference in the decomposition of the region containing the vertex H in Figures 2 and 4.

The linear edge quadtree has a number of deficiencies. All vertices and endpoints are stored at the lowest level of digitization, i.e., in nodes deep in the tree. There is no mechanism for following a line segment, as each node describes only that portion of a line segment which is contained within the borders of the node. In particular, given a node that contains a single point, there is no indication as to which of the neighboring nodes are connected to the point by a line segment.

An important criteria for evaluating whether or not a storage representation handles line segments properly is if the successive insertion and removal of the same line segment leaves the map unchanged. Since the edge quadtree nodes store only approximate local slope information, it is extremely difficult to restore nodes, by merging what had been split apart by the original insertion. For example, it is not easy to determine the endpoints of the edges emanating from a given vertex. Thus over time, the representation's compactness can deteriorate until it becomes equivalent to the MX quadtree (i.e., line segments are represented by pixel-sized nodes).

An alternative approach to storing linear feature data is based on the PR quadtree [5,7], which is an adaptation of the region quadtree to handle point data. Given an image representing a set of points, the image is subdivided into quadrants, subquadrants, etc., until each quadrant contains at most one point. The collection of points in Figure 5a is represented by the PR quadtree of Figure 5b. The PM quadtree [6] evolved from a desire to adapt the PR quadtree to store a polygonal map in a manner which preserves the relationship between edges and vertices. In essence, whenever a group of line segments meet at a common point, those segments can be organized by the linear ordering derived from their orientation. There are three variations [6] of PM quadtree, termed  $PM_1$ ,  $PM_2$ , and  $PM_3$ . We are only interested in the  $PM_1$  variant.

The  $PM_1$  quadtree is based on a decomposition rule that permits more than one line segment to be stored at a node only if they meet at a vertex that lies within the borders of that node. Figure 6 shows the  $PM_1$  quadtree corresponding to the polygonal map of Figure 1. From the decomposition of the line segments CD and CE, we observe that the representation of line segments which meet at narrow angles may require a large number of nodes.

### 3. THE SEGMENT QUADTREE

The segment quadtree has three types of nodes: 1) WHITE (or empty), 2) vertex, and 3) line (i.e., a node containing a line segment whose endpoints are boundaries of the block corresponding to the node). A linear quadtree representation stores each node type in the same (fixed) amount of space.

This space should be minimized since WHITE nodes will also be of the same size as line and vertex nodes, even though they need only to store a few bits distinguishing them from line and vertex nodes.

A line node in the segment quadtree is defined to contain precisely one line segment which enters from one side and exits through another side. With each line node we store the coordinates of the vertices of the line of which it is a component. This enables us to determine whether or not two neighboring line nodes are part of a larger line segment. Without this information, we cannot properly merge nodes after deletion of a nearby line segment. Note that recording the coordinates, although requiring more space, is preferable to recording the slope and intercept values for the line because it avoids precision errors as well as keeps all information in integer format.

A vertex node in the segment quadtree is defined to contain precisely one vertex and no line segments which do not intersect the vertex. This is identical to the definition of the  $PM_1$  quadtree. With each vertex node we store the  $x$  and  $y$  coordinates of the vertex that it represents. We do not explicitly store a list of the line segments that meet at the vertex.

In order to be able to perform boundary following, we must be able to follow line segments which extend from a vertex. This requires us to investigate the neighboring nodes of the vertex node. There are three cases. First, if a neighbor is empty, then there are no line segments extending from the vertex in that direction. Second, if a neighbor contains a single line segment, then we can check the slope of that line segment to determine whether or not it intersects the vertex. Finally, if the neighbor is a vertex, then we must be able to detect whether or not there exists an edge joining the two vertices. The solution is to store a four bit descriptor with each vertex node that indicates which of its sides are exited by one or more line segments.\* Using this scheme, whenever two vertices are contained in adjacent nodes, one of three situations can arise:

- (1) The corresponding sides are not exited through;
- (2) the corresponding sides are exited through; and
- (3) the side for one node is marked as exited through, and the corresponding side of the other node is not exited through.

(1) indicates that no line segment joins the two vertices (e.g., the boundary between vertices B and C in Figure 7), while (2) indicates that a line segment does join the vertices (e.g., the boundary between vertices A and B in Figure 7). (3) signifies that no line segment passes between the two vertices and that the vertex node with the exited side is larger than the vertex node with the unexited side (since there must be another node on that side into which a line segment exited). For example, consider vertices B and D in Figure 7. Note that all the nodes neighboring a given node on an exited side must be inspected since more than one line segment may exit from that side (e.g., the east edge of the node containing vertex D in Figure 7).

In summary, segment quadtree nodes may contain either a vertex in which case each line segment in the node intersects the vertex, or at most a single line segment which enters and exits the node. The resulting decomposition is

\*We adopt appropriate conventions with respect to open and closed sides to deal with line segments that exit from a corner of a node's block.

identical to that of the  $PM_1$  quadtree; however, the information stored is different. Figure 8 shows the segment quadtree corresponding to the polygonal map of Figure 1. Compare this with the  $PM_1$  quadtree of Figure 6.

An important aspect of the above decomposition rule is that it does not allow any case where there might be an unbounded decomposition of the tree. This would result if we would have taken as our decomposition rule one that did not permit a quadrant to contain more than one line segment. The maximum depth of the segment quadtree depends on two further factors: the minimum separation between a vertex and a line and the minimum separation between two lines.

In the case of geographic data, images are constructed from sequences of line segments that intersect only at their endpoints. Thus, if a user wishes to specify two intersecting line segments, then four line segments that meet at a common vertex must be specified. One way of analyzing the maximum depth of a segment quadtree is in terms of the number of bits necessary to specify either the  $x$  or  $y$  coordinates of the vertices. For example, if  $n$  bit numbers are used to specify the location of points, then the PR quadtree would have a maximum depth of  $n$ . It can be shown that under the same assumptions, the segment quadtree would have a maximum depth of  $4n$ .

#### 4. INSERTION INTO THE SEGMENT QUADTREE

Insertion of a line segment into a segment tree proceeds as follows. If the line segment is to be inserted into a region represented by an internal node of the quadtree, then it is clipped against each of the quadrants of that region and the appropriate component of the line segment is inserted into the corresponding subquadrants. Once a line segment has been clipped, it is important to remember whether its endpoints were vertices of the original line segment or artifacts of the clipping operation.

Upon encountering a leaf node there are three possible courses of action depending on its type - i.e., empty, line, or vertex. If the leaf is empty and the line segment contains two vertices (i.e., it is an unclipped line segment), then the region is further subdivided. If the leaf is empty and the line segment contains one vertex, then the node becomes a vertex node and is marked exited in the direction of the intercept of the line segment with the border of the node. If the line segment contains no vertices, then the node becomes a line node. If the leaf is a line node, then the nodes are decomposed further until the two line segments are not contained within the same node. If the leaf is a vertex node, and the line segment either contains another vertex or does not intersect the vertex, then the leaf node must be further decomposed. Otherwise, we need only mark the side of the node that intersects the line segment as exited.

#### 5. DELETION FROM THE SEGMENT QUADTREE

Deletion of a line segment from a segment quadtree is achieved by applying a two-step process to each quadtree node through which the line segment passes. First, we must remove the information corresponding to the presence of the line segment. Second, we must consider whether or not any of the modified nodes can be merged with their siblings.

The first step of removing the information corresponding to the presence of a node is quite easy. We simply repeat the decomposition of the line segment as though it were to be inserted, locating those segment quadtree leaf nodes which contain the segment. Line nodes are marked WHITE. Vertex nodes require more work since the side of the containing leaf through which the segment exits may also be exited by another (undeleted) line segment. If not, then the vertex segment bit marking that edge as exited is turned off. If, after this process, no edge of the node's block remains marked as exited, then the line segment being deleted is the only one intersecting that vertex, and the node is marked WHITE.

Merging eligible nodes can be done because it is possible to determine which neighbors of a node are connected by a line segment. Each node which contained a portion of the line segment being deleted must be considered as a potential candidate for merger with its three siblings. A node and its three siblings may be merged if they are either all WHITE, all line nodes (or WHITE) representing portions of the same line, or all represent a single vertex with only the line segments intersecting the vertex. The first two cases are easily determined by comparing the values of the node and its siblings - the siblings in this case will all be at the same depth in the tree. The third case might be difficult to determine, since two line segments may intersect at a vertex which is either not contained within the region covered by the node and its siblings, or stored within a sibling at a level deeper than that of the node to be merged. Therefore, it is necessary to check the subtrees of each sibling of the node to determine if all line segments stored intersect at the same vertex. If the vertex is not contained in these siblings, we must then check the siblings of the node's ancestors until either the vertex is located, or additional line segments or vertices are encountered which would not allow the nodes to merge.

For example, consider the deletion of line segment AE from Figure 1, the result of which is shown in Figure 9a. The leaf node marked I is the result of merging since it now contains only portions of line segment AG. The node containing vertex E is no longer marked as exited on the North. Figure 9b shows the result of deleting segment CE from Figure 9a. In the course of deleting the line segment all leaf nodes in the southeast quadrant will merge to form the node marked II.

#### 6. CONCLUSIONS

The segment quadtree has been presented and shown to be a suitable data structure for representing linear feature data in conjunction with image databases using linear quadtrees. Methods for insertion and deletion of line segments in a segment quadtree have been outlined. Border following is also quite simple. It is achieved by using neighbor finding techniques to determine the next node to be visited. The segment quadtree requires fewer nodes than the edge quadtree to store a collection of linear features since the vertices need not be stored at the lowest level of resolution as in the edge quadtree. This means that fewer nodes will be required to represent corners and intersections. Future work includes its implementation and integration into the geographical information system described in [9], where it will replace the linear edge quadtree.

## REFERENCES

1. D.H. Ballard, Strip trees: A hierarchical representation for curves, *Communications of the ACM* 24, 5(May 1981), 310-321 (see also corrigendum, *Communications of the ACM* 25, 3(March 1982), 213).
2. I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM* 25, 12(December 1982), 905-910.
3. G.M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1, 2(April 1979), 145-153.
4. A. Klinger, Patterns and search statistics, in *Optimizing Methods in Statistics*, J.S. Rustagi, Ed., Academic Press, New York, 1971, 303-337.
5. J.A. Orenstein, Multidimensional tries used for associative searching, *Information Processing Letters* 14, 4(June 1982), 150-157.
6. H. Samet and R. E. Webber, Using quadtrees to represent polygonal maps, *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, DC, June 1983, 127-132.
7. H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys* 16, 2(June 1984), 187-260.
8. H. Samet and R.E. Webber, On encoding boundaries with quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6, 3(May 1984), 365-369.
9. H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber, A geographic information system using quadtrees, *Pattern Recognition*, 6 (November/December 1984), 647-656.
10. M. Shneier, Two hierarchical linear feature representations: edge pyramids and edge quadtrees, *Computer Graphics and Image Processing* 17, 3(November 1981), 211-224.

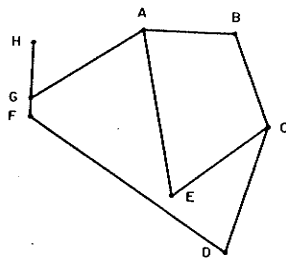


Figure 1: A polygonal map.

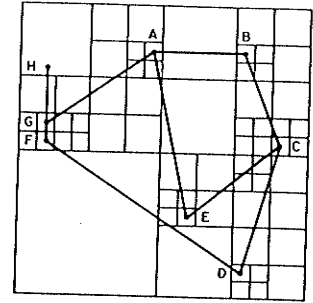


Figure 2: The edge quadtree for the polygonal map of Figure 1.

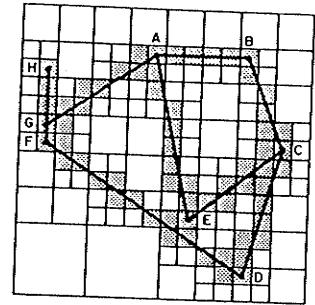


Figure 3: The MX quadtree for the polygonal map of Figure 1.

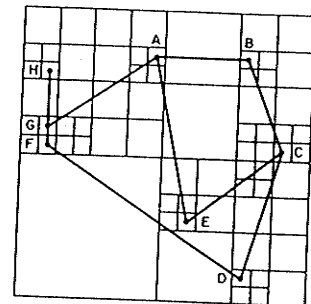
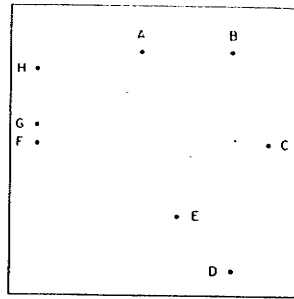
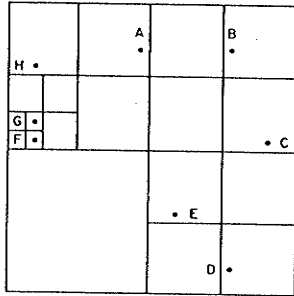


Figure 4: The linear edge quadtree for the polygonal map of Figure 1.



(a) A collection of points



(b) The PR quadtree for the points of (a)

Figure 5: A collection of points and its PR quadtree.

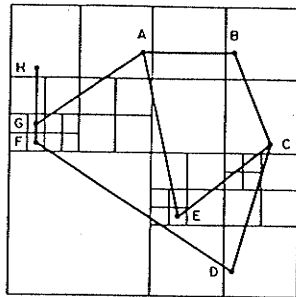


Figure 6: The  $PM_1$  quadtree for the polygonal map of Figure 1.

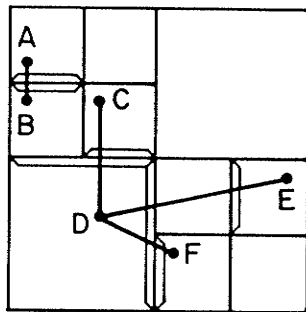


Figure 7: The segment quadtree for a collection of line segments. Vertex nodes have exited edges marked by a bracket.

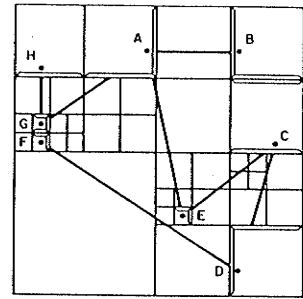
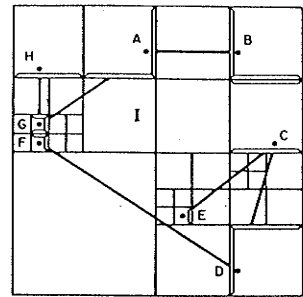
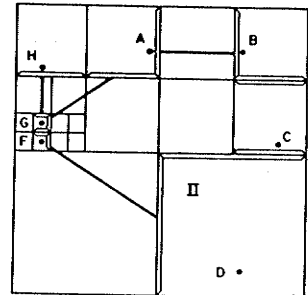


Figure 8: The segment quadtree for the polygonal map of Figure 1.



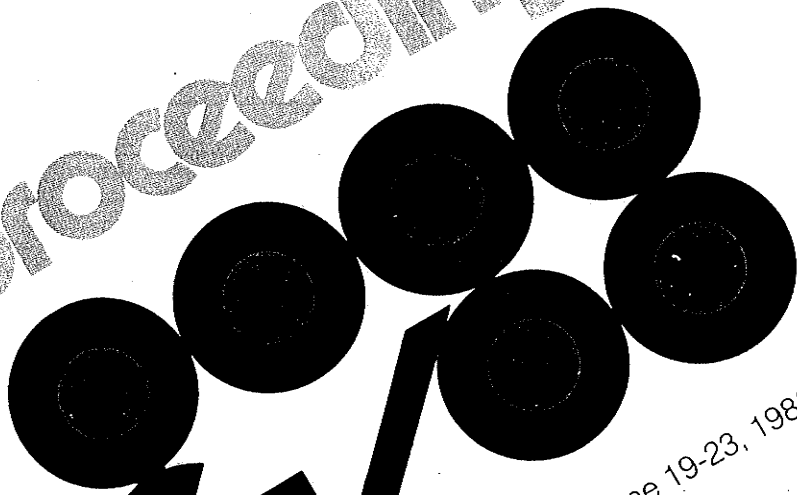
(a) The result of deleting segment AE from the segment quadtree of Figure 8.



(b) The result of deleting segment CE from the segment quadtree of (a).

Figure 9: Examples of deletion of line segments from the segment quadtree.

Proceedings



CV  
PR, 1985


June 19-23, 1985 • San Francisco, California

IEEE Computer Society Conference on

computer  
vision  
and  
pattern  
recognition

ISBN 0-8186-0633-9  
IEEE CATALOG NUMBER 85CH2145-1  
LIBRARY OF CONGRESS NUMBER 85-60198  
IEEE COMPUTER SOCIETY ORDER NUMBER 633

 IEEE COMPUTER SOCIETY

 THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

IEEE  
COMPUTER  
SOCIETY  
PRESS 