HIERARCHICAL DATA STRUCTURES FOR SPATIAL REASONING

Hanan Samet
Computer Science Department,
Center for Automation Research, and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742

Abstract

An overview, with an emphasis on recent results, is presented of the use of hierarchical data structures such as the quadtree for spatial reasoning. They are based on the principle of recursive decomposition. The focus is on the representation of data used in image databases. There is a greater emphasis on region data (i.e., 2-dimensional shapes) and to a lesser extent on point, curvilinear, and 3-dimensional data.

NATO ASI Series, Vol. F 65
Mapping and Spatial Modelling for Navigation
Edited by L. F. Pau

1. INTRODUCTION

The successful implementation of applications in spatial reasoning requires paying attention to the representation of spatial data. In particular, an integrated and uniform treatment of different spatial features is necessary in order to enable the reasoning to proceed quickly. Currently, the most prevalent features are points, rectangles, lines, regions, surfaces, and volumes. As an example of a reasoning task consider a query of the form "find all cities with population in excess of 5,000 in wheat growing regions within 10 miles of the Mississippi River." Note that this query is quite complex. It requires processing a line map (for the river), creating a corridor or buffer (to find the area within 10 miles of the river), a region map (for the wheat), and a point map (for the cities).

Spatial reasoning is eased by spatially sorting the data (i.e., a spatial index). In this paper we show how hierarchical data structures can be used to facilitate this process. They are based on the principle of recursive decomposition (similar to *divide and conquer* methods). In essence, they are used primarily as devices to sort data of more than one dimension and different spatial types. The term *quadtree* is often used to describe this class of data structures. In this paper, we focus on recent developments in the use of quadtree methods. We concentrate primarily on region data. For a more extensive treatment of this subject, see [Same84a, Same88a, Same88b, Same88c, Same89a, Same89b].

Our presentation is organized as follows. Section 2 briefly reviews the historical background of the origins of hierarchical data structures. Section 3 discusses improvements in the performance of some key operations on region data. Sections 4 and 5 describe hierarchical representations for point and line data, respectively, as well as give examples of their utility. Section 6 contains concluding remarks in the context of a geographic information system that makes use of these concepts.

2. HISTORICAL BACKGROUND

The term *quadtree* is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. They can be differentiated on the following bases: (1) the type of data that they are used to represent, (2) the principle guiding the decomposition process, and (3) the resolution (variable or not). Currently, they are used for points, rectangles, regions, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (termed a *regular decomposition*), or it may be governed by the input. The resolution of the decomposition (i.e., the number of times that the decomposition process is applied) may be fixed beforehand or it may be governed by properties of the input data.

The most common quadtree representation of data is the *region quadtree*. It is based on the successive subdivision of the image array into four equal-size quadrants. If the array does not consist entirely of 1s or entirely of 0s (i.e., the region does not cover the entire array), it is then subdivided into quadrants, subquadrants, etc., until blocks are obtained (possibly single pixels) that consist entirely of 1s or entirely of 0s. Thus, the region quadtree can be characterized as a variable resolution data structure.

As an example of the region quadtree, consider the region shown in Figure 1a which is represented by the $2^3 \times 2^3$ binary array in Figure 1b. Observe that the 1s correspond to picture elements (termed *pixels*) that are in the region and the 0s correspond to picture elements that are outside the region. The resulting blocks for the array of Figure 1b are shown in Figure 1c. This process is represented by a tree of degree 4.

In the tree representation, the root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, SE) of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. A leaf node is said to be BLACK or WHITE, depending on whether its corresponding block is entirely inside or entirely outside of the represented region. All non-leaf nodes are said to be GRAY. The quadtree representation for Figure 1c is shown in Figure 1d.

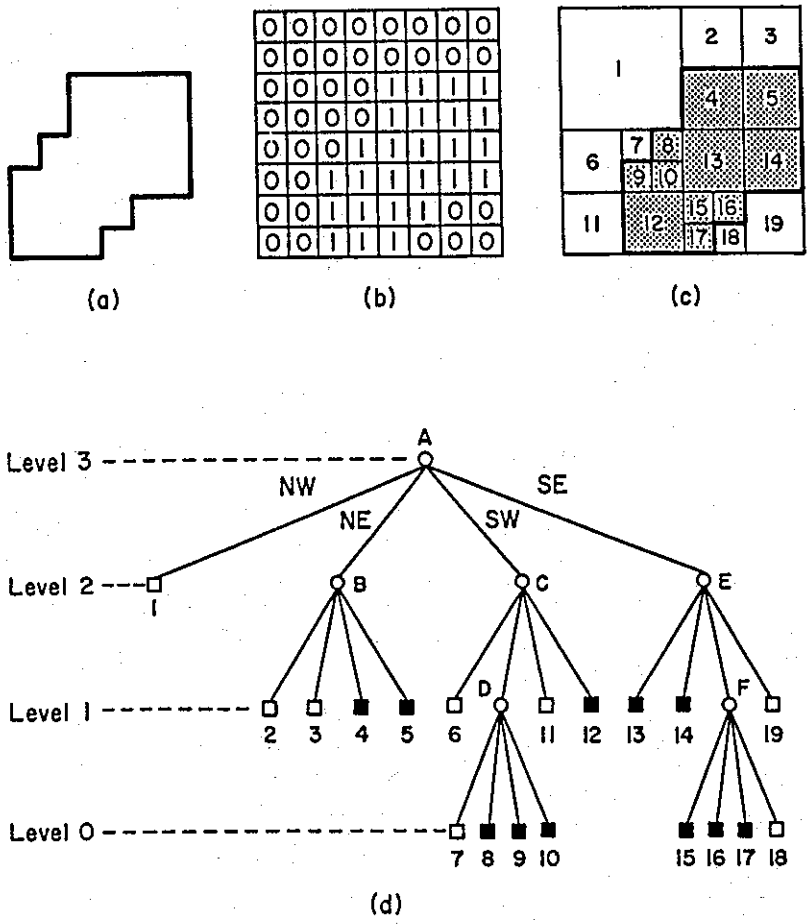

Figure 1. A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region. (b) Binary array. (c) Block decomposition of the region in (a). Blocks in the region are shaded. (d) Quadtree representation of the blocks in (c).

Quadtrees can also be used to represent non-binary images. In this case, we apply the same merging criteria to each color. For example, in the case of a landuse map, we simply merge all wheat growing regions, and likewise for corn, rice, etc. This is the approach taken by Samet *et al.* [Same84b].

Unfortunately, the term *quadtree* has taken on more than one meaning. The region quadtree, as shown above, is a partition of space into a set of squares whose sides are all a power of two long. This formulation is due to Klinger [Klin71] who used the term Q-tree [Klin76], whereas Hunter [Hunt78] was the first to use the term quadtree in such a context. A similar partition of space into rectangular quadrants, also termed a quadtree, was used by Finkel and Bentley [Fink74]. It is an adaptation of the binary search tree to two dimensions (which can be easily extended to an arbitrary number of dimensions). It is primarily used to represent multidimensional point data.

The origin of the principle of recursive decomposition is difficult to ascertain. Below, in order to give some indication of the uses of the quadtree, we briefly trace some of its applications to image data. Morton [Mort66] used it as a means of indexing into a geographic database. Warnock [Warn69] implemented a hidden surface elimination algorithm using a recursive decomposition of the picture area. The picture area is repeatedly subdivided into successively smaller rectangles while searching for areas sufficiently simple to be displayed. Horowitz and Pavlidis [Horo76] used the quadtree as an initial step in a "split and merge" image segmentation algorithm.

The pyramid of Tanimoto and Pavlidis [Tani75] is a close relative of the region quadtree. It is a multiresolution representation which is is an exponentially tapering stack of arrays, each one-quarter the size of the previous array. It has been applied to the problems of feature detection and segmentation. In contrast, the region quadtree is a variable resolution data structure.

Quadtree-like data structures can also be used to represent images in three dimensions and higher. The octree [Hunt78, Jack80, Meag82, Redd78] data structure is the three-dimensional analog of the quadtree. It is constructed in the following manner. We start with an image in the form of a cubical volume and recursively subdivide it into eight congruent disjoint cubes (called octants) until blocks are obtained of a uniform color or a predetermined level of decomposition is reached. Figure 2a is an example of a simple three-dimensional object whose raster octree block decomposition is given in Figure 2b and whose tree representation is given in Figure 2c.

One of the motivations for the development of hierarchical data structures such as the quadtree is a desire to save space. The original formulation of the quadtree encodes it as a tree structure that uses pointers. This requires additional overhead to encode the internal nodes of the tree. In order to further reduce the space requirements, two other approaches have been proposed. The first treats the image as a collection of leaf nodes where each leaf is encoded by a base 4 number termed a *locational code*, corresponding to a sequence of directional codes that locate the leaf along a path from the root of the quadtree. It is analogous to taking the binary representation of the x and y coordinates of a designated pixel in the block (e.g., the one at the lower left corner) and interleaving them (i.e., alternating the bits for each coordinate). It is difficult to determine the origin of this method (e.g., [Abel83, Garg82, Klin79, Mort66]).

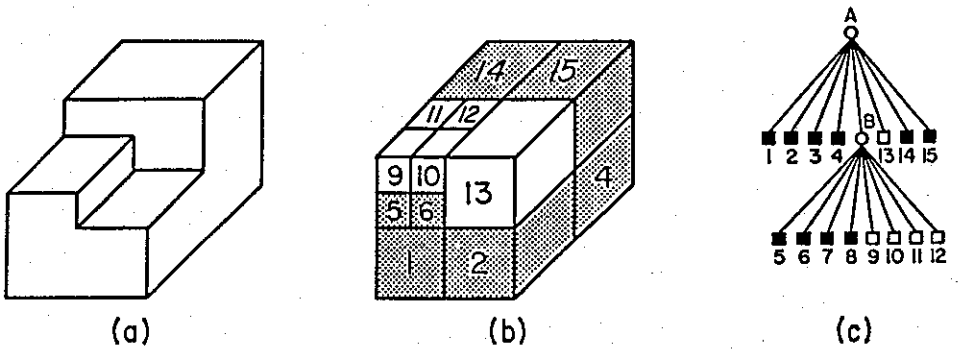

Figure 2. (a) Example three-dimensional object; (b) its octree block decomposition; and (c) its tree representation.

The second, termed a *DF-expression*, represents the image in the form of a traversal of the nodes of its quadtree [Kawa80]. It is very compact as each node type can be encoded with two bits. However, it is not easy to use when random access to nodes is desired. Recently, Samet and Webber [Same89d] showed that for a static collection of nodes, an efficient implementation of the pointer-based representation is often more economical spacewise than a locational code representation. This is especially true for images of higher dimension.

Nevertheless, depending on the particular implementation of the quadtree we may not necessarily save space (e.g., in many cases a binary array representation may still be more economical than a quadtree). However, the effects of the underlying hierarchical aggregation on the execution time of the algorithms are more important. Most quadtree algorithms are simply preorder traversals of the quadtree and, thus, their execution time is generally a linear function of the number of nodes in the quadtree. A key to the analysis of the execution time of quadtree algorithms is the *Quadtree Complexity Theorem* [Hunt78, Hunt79] which states that:

> For a quadtree of depth q representing an image space of $2^q \times 2^q$ pixels where these pixels represent a region whose perimeter measured in pixel-widths is p, then the number of nodes in the quadtree cannot exceed $16 \cdot q - 11 + 16 \cdot p$.

Since under all but the most pathological cases (e.g., a small square of unit width centered in a large image), the region perimeter exceeds the base 2 logarithm of the width of the image containing the region, the Quadtree Complexity Theorem means that the size of the quadtree representation of a region is linear in the perimeter of the region.

The Quadtree Complexity Theorem holds for three-dimensional data [Meag80] where perimeter is replaced by surface area, as well as higher dimensions for which it is stated as follows.

> The size of the k-dimensional quadtree of a set of k-dimensional objects is proportional to the sum of the resolution and the size of the $(k-1)$-dimensional interfaces between these objects.

The Quadtree Complexity Theorem also directly impacts the analysis of the execution time of algorithms. In particular, most algorithms that execute on a quadtree representation of an image instead of an array representation have an execution time

pixels. In its most general case, this means that the application of a quadtree algorithm to a problem in d-dimensional space executes in time proportional to the analogous array-based algorithm in the $(d-1)$-dimensional space of the surface of the original d-dimensional image. Therefore, quadtrees act like dimension-reducing devices.

3. ALGORITHMS USING QUADTREES

In this section, we describe how a number of basic image processing algorithms can be implemented using region quadtrees. In particular, we discuss point and object location, set operations, and quadtree construction.

3.1. POINT AND OBJECT LOCATION

The simplest task to perform on region data is to determine the color of a given pixel. In the traditional array representation, this is achieved by exactly one array access. In the region quadtree, this requires searching the quadtree structure. The algorithm starts at the root of the quadtree and uses the values of the x and y coordinates of the center of its block to determine which of the four subtrees contains the pixel. For example, if both the x and y coordinates of the pixel are less than the x and y coordinates of the center of the root's block, then the pixel belongs in the southwest subtree of the root. This process is performed recursively until a leaf is reached. It requires the transmission of parameters so that the center of the block corresponding to the root of the subtree currently being processed can be calculated. The color of that leaf is the color of the pixel. The execution time for the algorithm is proportional to the level of the leaf node containing the desired pixel.

The object-location operation is closely related to the point-location task. In this case, the x and y coordinates of the location of a pointing device (e.g., representing a mouse, tablet, lightpen, etc.) must be translated into the name of a nearby appropriate object (e.g., the nearest region corresponding to a specified feature). The leaf corresponding to the point is located as described above. If the leaf does not contain the feature, then we must investigate other leaf nodes.

Finding the nearest leaf node containing a specific feature (also known as the *nearest neighbor problem*) is achieved by a top-down recursive algorithm. Initially, at each level of the recursion, we explore the subtree that contains the location of the pointing device, say P. Once the leaf containing P has been found, the distance from P to the nearest feature in the leaf is calculated (empty leaf nodes have a value of infinity). Next, we unwind the recursion and, as we do so, at each level we search the subtrees that represent regions that overlap a circle centered at P whose radius is the distance to the closest feature that has been found so far. When more than one subtree must be searched, the subtrees representing regions nearer to P are searched before the subtrees that are further away (since it is possible that one of them may contain the desired feature thereby making it unnecessary to search the subtrees that are further away).

For example, suppose that the features are points. Consider Figure 3 and the task of finding the nearest neighbor of P in node 1. If we visit nodes in the order NW, NE, SW, SE, then as we unwind for the first time, we visit nodes 2 and 3 and the subtrees of the eastern brother of 1. Once we visit node 4, there is no need to visit node

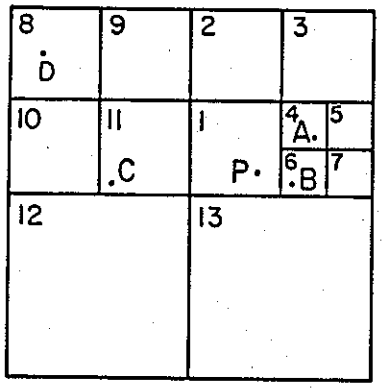

Figure 3. Example illustrating the neighboring object problem. P is the location of the pointing device. The nearest object is represented by point B in node 6.

5 since node 4 contained A. Nevertheless, we still visit node 6 which contains point B which is closer than A, but now there is no need to visit node 7. Unwinding one more level finds that due to the distance between P and B, there is no need to visit nodes 8, 9, 10, 11, and 12. However, node 13 must be visited as it could contain a point that is closer to P than B.

3.2. SET OPERATIONS

For a binary image, set-theoretic operations such as union and intersection are quite simple to implement [Hunt78, Hunt79, Shne81]. For example, the intersection of two quadtrees yields a BLACK node only when the corresponding regions in both quadtrees are BLACK. This operation is performed by simultaneously traversing three quadtrees. The first two trees correspond to the trees being intersected and the third tree represents the result of the operation. If any of the input nodes are WHITE, then the result is WHITE. When corresponding nodes in the input trees are GRAY, then their sons are recursively processed and a check is made for the mergibility of WHITE leaf nodes. The worst-case execution time of this algorithm is proportional to the sum of the number of nodes in the two input quadtrees. Note that as a result of actions (1) and (3), it is possible for the intersection algorithm to visit fewer nodes than the sum of the nodes in the two input quadtrees.

The above implementation assumes that the images are in registration (i.e., they are with respect to the same origin). However, at times, being able to perform set operations on images that are not in registration is very convenient as it enables the execution of many other operations [Same85a]. For example, windowing can be achieved by treating the image and the window as two distinct images, say I_1 and I_2, that are not in registration and performing a set intersection operation. In this case, I_1 is the image from which the window is being extracted and I_2 is a BLACK image with the same size and origin as the window to be extracted. The quadtree corresponding to the result of the windowing operation has the size and position of I_2, where each pixel of I_2 has the value of the corresponding pixel of I_1.

Using the same analogy, we can also shift an image. Specifically, shifting an image is equivalent to extracting a window that is larger than the input image and having a different origin than that of the input image. If the image to be shifted has an origin at (x,y), then shifting it by Δx and Δy means that the window is a BLACK block with an origin at $(x-\Delta x, y-\Delta y)$. Similar paradigms can also be applied to rotations of images by arbitrary amounts (not just multiples of 90°) [Same88b].

3.3. CONSTRUCTING QUADTREES

There are many algorithms for building a region quadtree. They differ in part on the representation of the input data. When the data is presented in raster scan order (i.e., the image is processed row by row), one algorithm [Same81] uses neighbor-finding [Same82, Same89c] to move through the quadtree in the order in which the data is encountered. Such an algorithm takes time proportional to the number of pixels in the image. Its execution time is dominated by the time necessary to check if nodes should be merged. This can be avoided by using a recently developed predictive technique [Shaf87] which assumes the existence of a homogeneous node of maximum size whenever a pixel that can serve as an upper left corner of a node is scanned (assuming a raster scan from left to right and top to bottom). In such a case, the need for merging is reduced and the algorithm's execution time is dominated by the number of blocks in the image rather than by the number of pixels. However, this algorithm does require the use of an auxiliary one-dimensional array of size equal to the width of the image.

In order to see how the predictive algorithm works, we briefly examine the construction of the quadtree corresponding to Figure 1. We say that a node is *active* if at least one pixel (but not all of the pixels) covered by the node has been processed and it differs in color from a node containing it. Assume the existence of a data structure to keep track of the active nodes. For each pixel in the raster scan traversal (starting at the first row), do the following. If the pixel is the same color as the appropriate active node, then do nothing. Otherwise, insert the largest possible node for which this is the first (i.e., upper leftmost) pixel, and (if it is not a 1×1 pixel node) add it to the set of active nodes. Remove any active nodes for which this is the last (lower right) pixel.

Table 1 is a trace of the values of the active nodes as the pixels that lead to the creation of nodes (or their removal from the active list of nodes) are processed. Each row in the table indicates the node that has become active (as well as its size), or the nodes that have been removed from the set of active nodes. In addition, the active nodes are tabulated according to their level. The pixel identifier (a,b) means that the pixel is in row a and column b relative to an origin at the upper left corner of the image. Figure 4 illustrates some of the nodes created during the building process by using their names to label the pixel that caused their creation. Figures 5a-d indicate in greater detail some of the steps in the construction of the quadtree.

When the first pixel of the array is processed, the entire quadtree is represented by a single WHITE node (node A in Figure 5a). No other insertions occur while processing rows 0 and 1. When the first BLACK pixel (2,4) is processed, node B of Figure 5a becomes active. The insertion of node B causes node A to be split and its NE son is split again. When BLACK pixel (2,5) is processed, node B will be located in the active node table since it is the smallest active node containing that pixel.

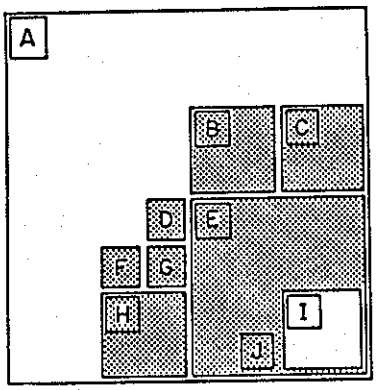

Figure 4. Correlation between some of the nodes of the quadtree corresponding to Figure 1 and the pixels that led to their creation.

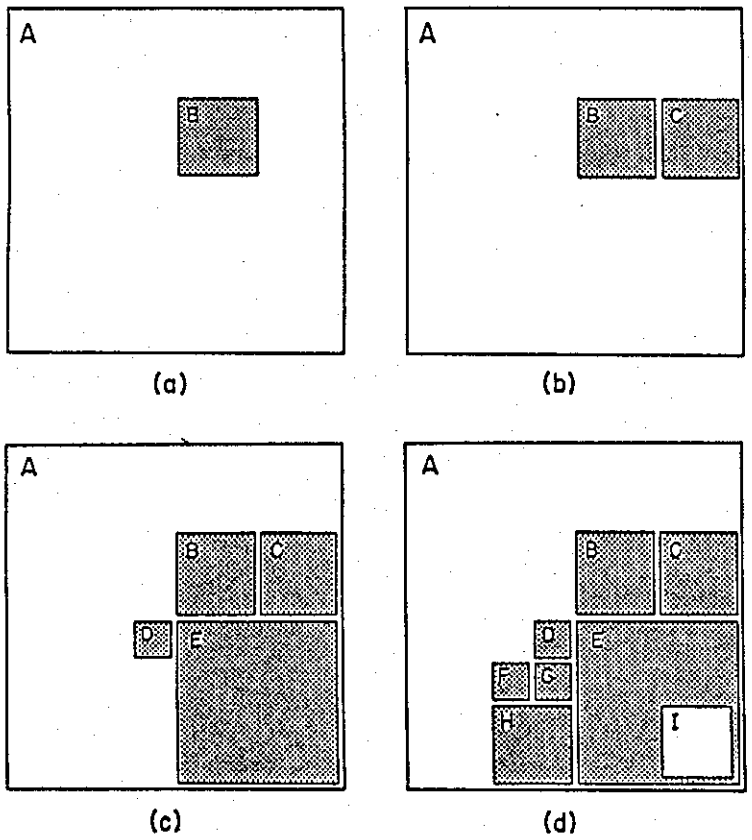

Figure 5. Intermediate states of the quadtree as the predictive algorithm is used to build the quadtree corresponding to Figure 1. (a) State after processing pixel (2,4); (b) state after processing pixel (2,6); (c) state after processing pixel (4,4); (d) state after processing pixel (6,6).

When BLACK pixel (2,6) is processed, node C of Figure 5b becomes active, since only active WHITE node A contains it at that point. As row 3 is processed, nodes B and C are deactivated when their lower right pixels are processed (i.e., pixels (3,5) and (3,7) respectively). Pixel (4,3) causes the insertion of node D which resulted in two node splitting operations. When pixel (4,4) is processed, node E is inserted as shown in Figure 5c. The nodes previously labeled B and C are not active. Similarly, pixel-sized node D at (4,3) is not active since it contains no unprocessed pixels. Therefore, nodes A and E are the only active nodes.

Table 1. Trace of the active nodes as the quadtree for Figure 1a is built.

Pixel	Action	Size	Active Nodes by Level		
			3	2	1
(0,0)	insert WHITE node A	8×8	A		
(2,4)	insert BLACK node B	2×2	A		B
(2,6)	insert BLACK node C	2×2	A		B C
(3,5)	remove B from active		A		C
(3,7)	remove C from active		A		
(4,3)	insert BLACK node D	1×1	A		
(4,4)	insert BLACK node E	4×4	A	E	
(5,2)	insert BLACK node F	1×1	A	E	
(5,3)	insert BLACK node G	1×1	A	E	
(6,2)	insert BLACK node H	2×2	A	E	H
(6,6)	insert WHITE node I	2×2	A	E	H I
(7,3)	remove H from active		A	E I	
(7,5)	insert WHITE node J	1×1	A	E I	
(7,7)	remove I, E, A from active				

Pixels (5,2) and (5,3) cause the insertion of nodes F and G. Node H becomes active after processing pixel (6,2). Pixel (6,6) is WHITE and since the smallest node containing it had been BLACK, a WHITE node, I, has been inserted as well as made active (see Figure 5d). When processing pixel (6,7) we find that three active nodes (i.e., A, E, and I) contain it, with the smallest being node I. Pixel (7,3) causes node H to be removed from the set of active nodes. Pixel (7,5) results in the insertion of node J. Since pixel (7,7) is the lower rightmost pixel in the image, it causes the removal of all active nodes from the active node list (i.e., nodes I, E, and A). The final result of the quadtree building process is shown in Figure 4.

Use of the predictive quadtree construction algorithm for 512×512 images resulted in execution time speedups of factors as high as 40 and usually at least one order of magnitude [Shaf87]. This is a very important result because it means that like all other quadtree operations, the execution time of building a quadtree is also proportional to the complexity of the image. In other words, the building time is competitive with the time necessary to perform a set operation.

4. POINT DATA

Multidimensional point data can be represented in a variety of ways. The representation ultimately chosen for a specific task will be heavily influenced by the type of operations to be performed on the data. Our focus is on dynamic files (i.e., the number of data can grow and shrink at will) and on applications involving search. In Section 2 we briefly mentioned the point quadtree of Finkel and Bentley [Fink74]. In this section we discuss the PR quadtree (P for point and R for region) [Oren82, Same84a]. It is an adaptation of the region quadtree to point data which associates data points (that need not be discrete) with quadrants. The PR quadtree is organized in the same way as the region quadtree. The difference is that leaf nodes are either empty (i.e., WHITE) or contain a data point (i.e., BLACK) and its coordinates. A quadrant contains at most one data point. For example, Figure 6 is a PR quadtree corresponding to some point data.

Data points are inserted into PR quadtrees in a manner analogous to that used to insert in a point quadtree - i.e., a search is made for them. Actually, the search is for the quadrant in which the data point, say A, belongs (i.e., a leaf node). If the quadrant is already occupied by another data point with different x and y coordinates, say B, then the quadrant must repeatedly be subdivided (termed *splitting*) until nodes A and B no longer occupy the same quadrant. This may result in many subdivisions, especially if the Euclidean distance between A and B is very small. The shape of the resulting PR quadtree is independent of the order in which data points are inserted into it. Deletion of nodes is more complex and may require collapsing of nodes - i.e., the direct counterpart of the node splitting process outlined above.

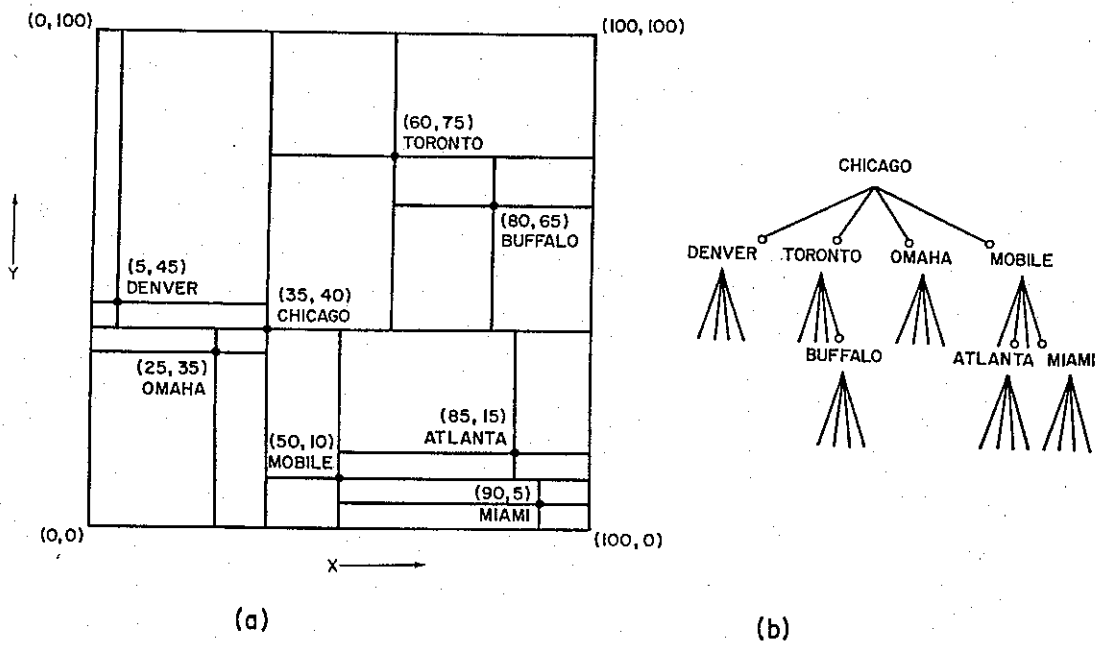

Figure 6. A PR quadtree (b) and the records it represents (a).

PR quadtrees, as well as other quadtree-like representations for point data, are especially attractive in applications that involve search. A typical query is one that requests the determination of all records within a specified distance of a given record - i.e., all cities within 100 miles of Washington, DC. The efficiency of the PR quadtree lies in its role as a pruning device on the amount of search that is required. Thus many records will not need to be examined. For example, suppose that in the hypothetical database of Figure 6 we wish to find all cities within 8 units of a data point with coordinates (84,10). In such a case, there is no need to search the NW, NE, and SW quadrants of the root (i.e., (50,50)). Thus we can restrict our search to the SE quadrant of the tree rooted at root. Similarly, there is no need to search the NW, NE, and SW quadrants of the tree rooted at the SE quadrant (i.e., (75,25)). Note that the search ranges are usually orthogonally defined regions such as rectangles, boxes, etc. Other shapes are also feasible as the above example demonstrated (i.e., a circle).

5. LINE DATA

Section 3 was devoted to the region quadtree, an approach to region representation that is based on a description of the region's interior. In this section, we focus on a representation that specifies the boundaries of regions. We concentrate on use of the PM quadtree family [Same85b, Nels86] (see also edge-EXCELL [Tamm81]) in the representation of collections of polygons (termed *polygonal maps*). There are a number of variants of the PM quadtree. These variants are either vertex-based or edge-based. They are all built by applying the principle of repeatedly breaking up the collection of vertices and edges (forming the polygonal map) until obtaining a subset that is sufficiently simple so that it can be organized by some other data structure.

The PM quadtrees of Samet and Webber [Same85b] are vertex-based. We illustrate the PM_1 quadtree. It is based on a decomposition rule stipulating that partitioning occurs as long as a block contains more than one line segment unless the line segments are all incident at the same vertex which is also in the same block (e.g., Figure 7).

Samet, Shaffer, and Webber [Same87] show how to compute the maximum depth of the PM_1 quadtree for a polygonal map in a limited, but typical, environment. They consider a polygonal map whose vertices are drawn from a grid (say $2^n \times 2^n$), and do not permit edges to intersect at points other than the grid points (i.e., vertices). In such a case, the depth of any leaf node is bounded from above by $4n+1$. This enables a determination of the maximum amount of storage that will be necessary for each node.

A similar representation has been devised for three-dimensional images [Ayal85, Carl85, Fuji85, Hunt81, Nava86, Quin82, Tamm81, Vand84]. The decomposition criteria are such that no node contains more than one face, edge, or vertex unless the faces all meet at the same vertex or are adjacent to the same edge. For example, Figure 8b is a PM_1 octree decomposition of the object in Figure 8a. This representation is quite useful since its space requirements for polyhedral objects are significantly smaller than those of a conventional octree.

The PMR quadtree [Nels86] is an edge-based variant of the PM quadtree. It makes use of a probabilistic splitting rule. A node is permitted to contain a variable

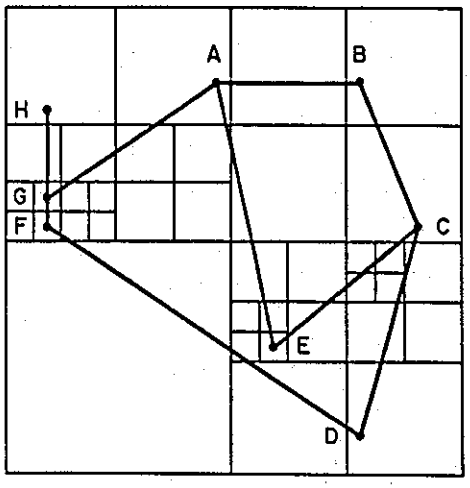

Figure 7. Example PM_1 quadtree.

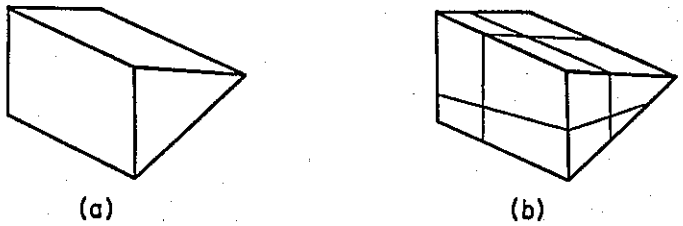

Figure 8. (a) Example three-dimensional object; and (b) its corresponding PM_1 octree.

number of line segments. A line segment is stored in a PMR quadtree by inserting it into the nodes corresponding to all the blocks that it intersects. During this process, the occupancy of each node that is intersected by the line segment is checked to see if the insertion causes it to exceed a predetermined *splitting threshold*. If the splitting threshold is exceeded, then the node's block is split *once*, and only once, into four equal quadrants.

On the other hand, a line segment is deleted from a PMR quadtree by removing it from the nodes corresponding to all the blocks that it intersects. During this process, the occupancy of the node and its siblings is checked to see if the deletion causes the total number of line segments in them to be less than the predetermined splitting threshold. If the splitting threshold exceeds the occupancy of the node and its siblings, then they are merged and the merging process is reapplied to the resulting node and its siblings. Notice the asymmetry between the splitting and merging rules.

Members of the PM quadtree family can be easily adapted to deal with fragments that result from set operations such as union and intersection so that there is no data degradation when fragments of line segments are subsequently recombined. Their use yields an exact representation of the lines - not an approximation. To see how this is

achieved, let us define a *q-edge* to be a segment of an edge of the original polygonal map that either spans an entire block in the PM quadtree or extends from a boundary of a block to a vertex within the block (i.e., when the block contains a vertex).

Each q-edge is represented by a pointer to a record containing the endpoints of the edge of the polygonal map of which the q-edge is a part [Nels86]. The line segment descriptor stored in a node only implies the presence of the corresponding q-edge - it does not mean that the entire line segment is present as a lineal feature. The result is a consistent representation of line fragments since they are stored exactly and, thus, they can be deleted and reinserted without worrying about errors arising from the roundoffs induced by approximating their intersection with the borders of the blocks through which they pass.

6. CONCLUDING REMARKS

The use of hierarchical data structures in spatial reasoning enables the focussing of computational resources on the interesting subsets of data. Thus, there is no need to expend work where the payoff is small. Although many of the operations for which they are used can often be performed equally as efficiently, or more so, with other data structures, hierarchical data structures are attractive because of their conceptual clarity and ease of implementation.

When the hierarchical data structures are based on the principle of regular decomposition, we have the added benefit of a spatial index. All features, be they regions, points, rectangles, lines, volumes, etc., can be represented by maps which are in registration. In fact, such a system has been built [Same84b] for representing geographic information. In this case, the quadtree is implemented as a collection of leaf nodes where each leaf node is represented by its locational code. The collection is in turn represented as a B-tree [Come79]. There are leaf nodes corresponding to region, point, and line data.

The disadvantage of quadtree methods is that they are shift sensitive in the sense that their space requirements are dependent on the position of the origin. However, for complicated images the optimal positioning of the origin will usually lead to little improvement in the space requirements. The process of obtaining this optimal positioning is computationally expensive and is usually not worth the effort [Li82].

The fact that we are working in a digitized space may also lead to problems. For example, the rotation operation is not generally invertible. In particular, a rotated square usually cannot be represented accurately by a collection of rectilinear squares. However, when we rotate by 90°, then the rotation is invertible. This problem arises whenever one uses a digitized representation. Thus, it is also common to the array representation.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grant IRI-8802457. I would like to acknowledge the many valuable discussions that I have had with Michael B. Dillencourt, Randal C. Nelson, Azriel Rosenfeld, Clifford A. Shaffer, Markku Tamminen, and Robert E. Webber.

REFERENCES

1. [Abel83] - D.J. Abel and J.L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, *Computer Vision, Graphics, and Image Processing 24*, 1(October 1983), 1-13.

2. [Ayal85] - D. Ayala, P. Brunet, R. Juan, and I. Navazo, Object representation by means of nonminimal division quadtrees and octrees, *ACM Transactions on Graphics 4*, 1(January 1985), 41-59.

3. [Carl85] - I. Carlbom, I. Chakravarty, and D. Vanderschel, A hierarchical data structure for representing the spatial decomposition of 3-D objects, *IEEE Computer Graphics and Applications 5*, 4(April 1985), 24-31.

4. [Come79] - D. Comer, The Ubiquitous B-tree, *ACM Computing Surveys 11*, 2(June 1979), 121-137.

5. [Fink74] - R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, *Acta Informatica 4*, 1(1974), 1-9.

6. [Fuji85] - K. Fujimura and T.L. Kunii, A hierarchical space indexing method, *Proceedings of Computer Graphics'85*, Tokyo, 1985, T1-4, 1-14.

7. [Garg82] - I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM 25*, 12(December 1982), 905-910.

8. [Horo76] - S.L. Horowitz and T. Pavlidis, Picture segmentation by a tree traversal algorithm, *Journal of the ACM 23*, 2(April 1976), 368-388.

9. [Hunt78] - G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ, 1978.

10. [Hunt81] - G.M. Hunter, Geometrees for interactive visualization of geology: an evaluation, System Science Department, Schlumberger-Doll Research, Ridgefield, CT, 1981.

11. [Hunt79] - 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.

12. [Jack80] - C.L. Jackins and S.L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Computer Graphics and Image Processing 14*, 3(November 1980), 249-270.

13. [Kawa80] - E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression, *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 1(January 1980), 27-35.

14. [Klin71] - A. Klinger, Patterns and Search Statistics, in *Optimizing Methods in Statistics*, J.S. Rustagi, Ed., Academic Press, New York, 1971, 303-337.

15. [Klin76] - A. Klinger and C.R. Dyer, Experiments in picture representation using regular decomposition, *Computer Graphics and Image Processing 5*, 1(March 1976), 68-105.

16. [Klin79] - A. Klinger and M.L. Rhodes, Organization and access of image data by areas, *IEEE Transactions on Pattern Analysis and Machine Intelligence 1*, 1(January 1979), 50-60.

17. [Li82] - M. Li, W.I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, *Computer Graphics and Image Processing 20*, 1(September 1982), 72-81.

18. [Meag80] - D. Meagher, Octree encoding: a new technique for the representation, The manipulation, and display of arbitrary 3-d objects by computer, Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, Troy, New York, October 1980.

19. [Meag82] - D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing 19*, 2(June 1982), 129-147.

20. [Mort66] - G.M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM Ltd., Ottawa, Canada, 1966.

21. [Nava86] - I. Navazo, Contribució a les tecniques de modelat geometric d'objectes poliedrics usant la codificació amb arbres octals, Ph.D. dissertation, Escola Tecnica Superior d'Enginyers Industrials, Department de Metodes Informatics, Universitat Politechnica de Barcelona, Barcelona, Spain, January 1986.

22. [Nels86] - R.C. Nelson and H. Samet, A consistent hierarchical representation for vector data, *Computer Graphics 20*, 4(August 1986), pp. 197-206 (also *Proceedings of the SIGGRAPH'86 Conference*, Dallas, August 1986).

23. [Oren82] - J.A. Orenstein, Multidimensional tries used for associative searching, *Information Processing Letters 14*, 4(June 1982), 150-157.

24. [Quin82] - K.M. Quinlan and J.R. Woodwark, A spatially-segmented solids database - justification and design, *Proceedings of CAD 82 Conference*, Butterworth, Guildford, United Kingdom, 1982, 126-132.

25. [Redd78] - D.R. Reddy and S. Rubin, Representation of three-dimensional objects, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, April 1978.

26. [Same81] - H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 1(January 1981), 93-95.

27. [Same82] - H. Samet, Neighbor finding techniques for images represented by

quadtrees, *Computer Graphics and Image Processing 18*, 1(January 1982), 37-57.

28. [Same84a] - H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2(June 1984), 187-260.

29. [Same88a] - H. Samet, Hierarchical representations of collections of small rectangles, to appear in *ACM Computing Surveys* (also University of Maryland Computer Science TR-1967).

30. [Same89a] - H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1989.

31. [Same89b] - H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*, Addison-Wesley, Reading, MA, 1989.

32. [Same89c] - H. Samet, Neighbor finding in images represented by octrees, *Computer Vision, Graphics, and Image Processing 46*, 3(June 1989), 367-386.

33. [Same85a] - H. Samet, A. Rosenfeld, C.A. Shaffer, R.C. Nelson, Y-G. Huang, and K. Fujimura, Application of hierarchical data structures to geographic information systems: phase IV, Computer Science TR-1578, University of Maryland, College Park, MD, December 1985.

34. [Same84b] - H. Samet, A. Rosenfeld, C.A. Shaffer, and R.E. Webber, A geographic information system using quadtrees, *Pattern Recognition 17*, 6 (November/December 1984), 647-656.

35. [Same87] - H. Samet, C.A. Shaffer, and R.E. Webber, Digitizing the plane with cells of non-uniform size, *Information Processing Letters 24*, 6(April 1987), 369-375.

36. [Same85b] - H. Samet and R.E. Webber, Storing a collection of polygons using quadtrees, *ACM Transactions on Graphics 4*, 3(July 1985), 182-222 (also *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, DC, June 1983, 127-132; and University of Maryland Computer Science TR-1372).

37. [Same89d] - H. Samet and R.E. Webber, A comparison of the space requirements of multi-dimensional quadtree-based file structures, to appear in *The Visual Computer* (also University of Maryland Computer Science TR-1711).

38. [Same88b] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics. Part I. Fundamentals, *IEEE Computer Graphics and Applications 8*, 3(May 1988), 48-68.

39. [Same88c] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics. Part II. Applications, *IEEE Computer Graphics and Applications 8*, 4(July 1988), 59-75.

40. [Shaf87] - C.A. Shaffer and H. Samet, Optimal quadtree construction algorithms, *Computer Vision, Graphics, and Image Processing 37*, 3(March 1987), 402-419.

41. [Shne81] - M. Shneier, Calculations of geometric properties using quadtrees, *Computer Graphics and Image Processing 16*, 3(July 1981), 296-302.

42. [Tamm81] - M. Tamminen, The EXCELL method for efficient geometric access to data, *Acta Polytechnica Scandinavica*, Mathematics and Computer Science Series No. 34, Helsinki, 1981.

43. [Tani75] - S. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, *Computer Graphics and Image Processing 4*, 2(June 1975), 104-119.

44. [Vand84] - D.J. Vanderschel, Divided leaf octal trees, Research Note, Schlumberger-Doll Research, Ridgefield, CT, March 1984.

45. [Warn69] - J.L. Warnock, A hidden surface algorithm for computer generated half tone pictures, Computer Science Department TR 4-15, University of Utah, Salt Lake City, June 1969.