AN OVERVIEW OF QUADTREES, OCTREES, AND RELATED HIERARCHICAL DATA STRUCTURES*

Hanan Samet
Computer Science Department
University of Maryland
College Park, Maryland 20742

ABSTRACT

An overview of hierarchical data structures for representing images, such as the quadtree and octree, is presented. They are based on the principle of recursive decomposition. The emphasis is on the representation of data used in applications in computer graphics, computer-aided design, robotics, computer vision, and cartography. There is a greater emphasis on region data (i.e., 2-dimensional shapes) and to a lesser extent on point, line, and 3-dimensional data.

Keywords: quadtrees, octrees, hierarchical data structures, computer graphics

1. INTRODUCTION

Hierarchical data structures are becoming increasingly important representation techniques in the domains of computer graphics, computer-aided design, robotics, computer vision, and cartography. They are based on the principle of recursive decomposition (similar to *divide and conquer* methods). One such data structure is the quadtree. As we shall see, the term *quadtree* has taken on a generic meaning. In this overview it is our goal to show how a number of data structures used in different domains are related to each other and to quadtrees. Our presentation concentrates on these different representations and illustrates how some basic operations which use them are performed. Whenever possible, we give examples from the domain of computer graphics. For a more extensive treatment of this subject, including a comprehensive set of references, see [Same84a, Same87].

This overview is organized as follows. Section 2 discusses the historical background of hierarchical data structures. Section 3 points out the key properties of hierarchical space decompositions. Section 4 shows how some basic operations are performed on region data. Section 5 describes hierarchical representations for point data while Section 6 does the same for line data. In each section we also mention how the techniques can be applied to 3-dimensional data. Section 7 contains concluding remarks that include a summary of the advantages and disadvantages of hierarchical methods.

*The support of the National Science Foundation under Grant DCR-86-05557 is gratefully acknowledged.

NATO ASI Series, Vol. F40
Theoretical Foundations of Computer Graphics and CAD
Edited by R.A. Earnshaw

2. 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 represent, (2) the principle guiding the decomposition process, and (3) the resolution (variable or not). Currently, they are used for point data, regions, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (i.e., regular polygons, termed a *regular decomposition*), or it may be governed by the input. In computer graphics this distinction is often phrased in terms of image-space hierarchies versus object-space hierarchies, respectively. 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.

Our first example of quadtree representation of data is concerned with the representation of region data. The most studied quadtree approach to region representation, termed a *region quadtree*, is based on the successive subdivision of the image array into four equal-size quadrants. If the array does not consist entirely of 1's or entirely of 0's (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 1's or entirely of 0's, i.e., each block is entirely contained in the region or entirely disjoint from it. Thus the region quadtree can be characterized as a variable resolution data structure. As an example, 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 1's correspond to picture elements (termed *pixels*) that are in the region and the 0's 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 (i.e., each non-leaf node has four sons). 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. Non-leaf nodes are GRAY. The quadtree representation for Figure 1c is shown in Figure 1d. Although the example described in Figure 1 corresponds to a binary image, the extension to non-binary images is straightforward.

The region quadtree is easily extended to represent 3-dimensional data and the resulting data structure is termed an *octree*. 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 of a uniform color are obtained, or a predetermined level of decomposition is reached. Figure 2a is an example of a simple 3-dimensional object whose octree block decomposition is given in Figure 2b and whose tree representation is given in Figure 2c.

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, whereas Hunter [Hunt78] first used the term quadtree in this context. A similar partition of space into rectangular quadrants, also termed a quadtree, was used

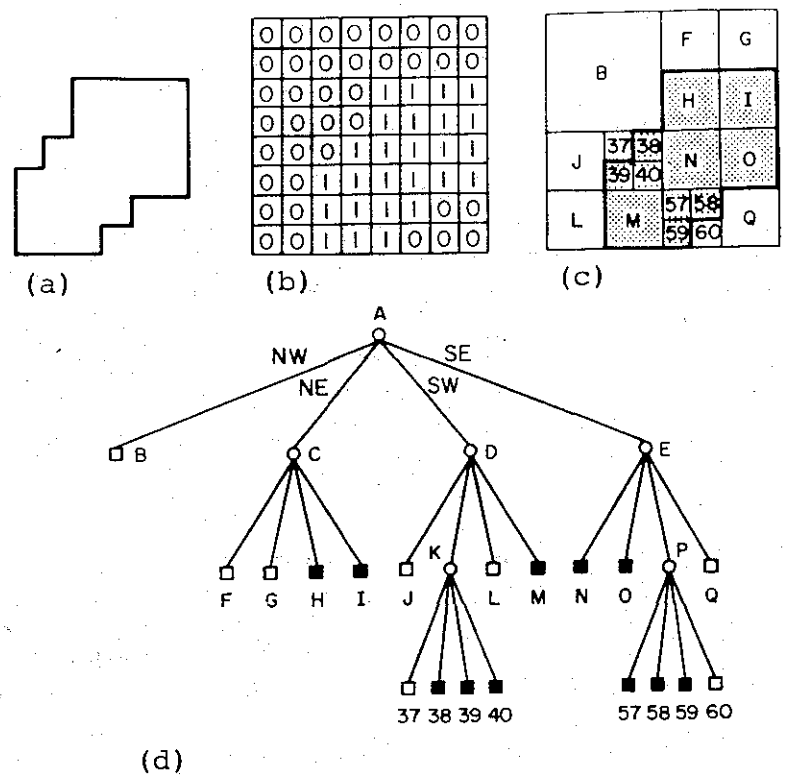

Figure 1. A region, its binary array, its maximal blocks, and the corresponding quadtree. (a) Region. (b) Binary array. (c) Block decomposition of the region in (a). Blocks in the region are shaded. (d) Quadtree representation of the blocks in (c).

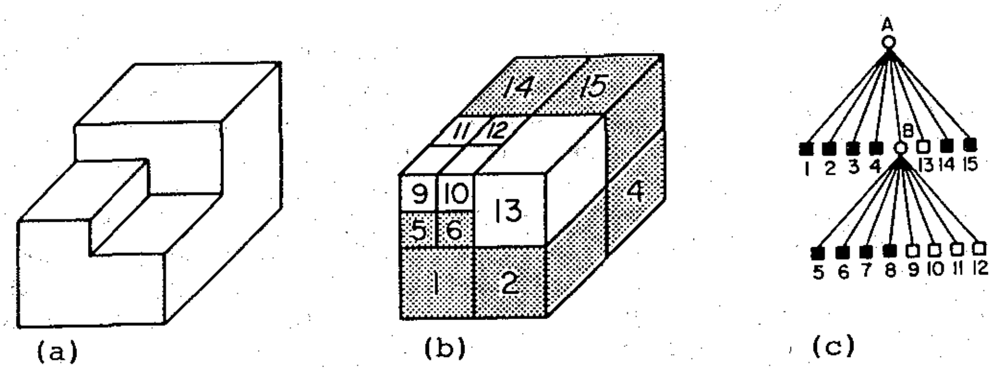

Figure 2. (a) Example 3-dimensional object; (b) its octree block decomposition; and (c) its tree representation.

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. We shall refer to it as a *point quadtree* when confusion with a region quadtree is possible.

The origin of the principle of recursive decomposition is difficult to ascertain. Below, in order to give some indication of the origins of quadtrees and octrees, we briefly, and incompletely, trace some of their applications. Morton [Mort66] used it as a means of indexing into a geographic database. Warnock [Warn68] 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. The pyramid of Tanimoto and Pavlidis [Tani75] is a close relative of the region quadtree. It is an exponentially tapering stack of arrays, each one-quarter the size of the previous array. The pyramid is a multiresolution representation whereas the region quadtree is a variable resolution data structure.

The octree was developed independently by various researchers. Hunter [Hunt78] mentioned it as a natural extension of the quadtree. Reddy and Rubin [Redd78] proposed the octree as one of three representations for solid objects. The second is a 3-dimensional generalization of the point quadtree of Finkel and Bentley [Fink74] - i.e., a decomposition into rectangular parallelepipeds (as opposed to cubes) with planes perpendicular to the x, y, and z axes. The third breaks the object into rectangular parallelepipeds of an arbitrary size that are not necessarily aligned with an axis. Jackins and Tanimoto [Jack80] adapted Hunter's and Steiglitz's quadtree translation algorithm to objects represented by octrees. Meagher [Meg82] developed numerous algorithms for performing solid modeling where the octree is the underlying representation.

3. PROPERTIES OF QUADTREE AND OCTREE SPACE DECOMPOSITIONS

A number of different planar decomposition methods exist. We use a quadtree in the form of squares because it is a planar decomposition that satisfies these two properties: (1) Since it yields a partition that is an infinitely repetitive pattern, it can be used for images of any size. (2) It yields a partition that is infinitely decomposable into increasingly finer patterns (i.e., higher resolution).

A quadtree-like decomposition into four equilateral triangles also satisfies these criteria. However, unlike the decomposition into squares, it does not have a uniform orientation - i.e., all tiles with the same orientation cannot be mapped into each other by translations of the plane that do not involve rotation or reflection. In contrast, a decomposition into hexagons has a uniform orientation but does not satisfy property (2). Nevertheless, triangular quadtrees have been used - e.g., Yamaguchi *et al.* [Yama84] use them to generate an isometric view from an octree representation of an object. For more details on properties of decompositions see Bell *et al.* [Bell83].

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. To further reduce the space requirements, two other approaches have been proposed. The first, termed the *linear quadtree* [Garg82], 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). 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 always easy to use when random access to nodes is desired. Interestingly, Samet and Webber [Same86] show that for a static collection of nodes, an efficient implementation of the pointer-based representation will often be 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. In this case, our discussion assumes a tree representation in the sense that the number of nodes in the quadtree includes the internal nodes.

A key to the analysis of the execution time of quadtree algorithms is the Quadtree Complexity Theorem [Hunt78, Hunt79], which states that, except for pathological cases, the number of nodes in the quadtree representation of a region is proportional to the perimeter of the region. An alternative interpretation of this result is that for a given image, if the resolution doubles and hence the perimeter doubles (ignoring fractal effects), then the number of nodes will double. On the other hand, for the 2-dimensional array representation, when the resolution doubles, the size of the array quadruples. The Quadtree Complexity Theorem holds for 3-dimensional data [Meag80] where perimeter is replaced by surface area, as well as higher dimensions.

Aside from its implications on storage requirements, 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 that is proportional to the number of blocks in the image rather than the number of pixels. Generally, 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. Thus quadtrees are somewhat like dimension-reducing devices.

4. ALGORITHMS USING QUADTREES AND OCTREES FOR REGION DATA

In this section we describe how a number of operations can be implemented using quadtrees and octrees. In particular, we discuss set operations, quadtree and octree construction, polygon coloring, and display. We also expand on the concept of neighbor finding which serves as a basis for many algorithms using quadtrees and octrees.

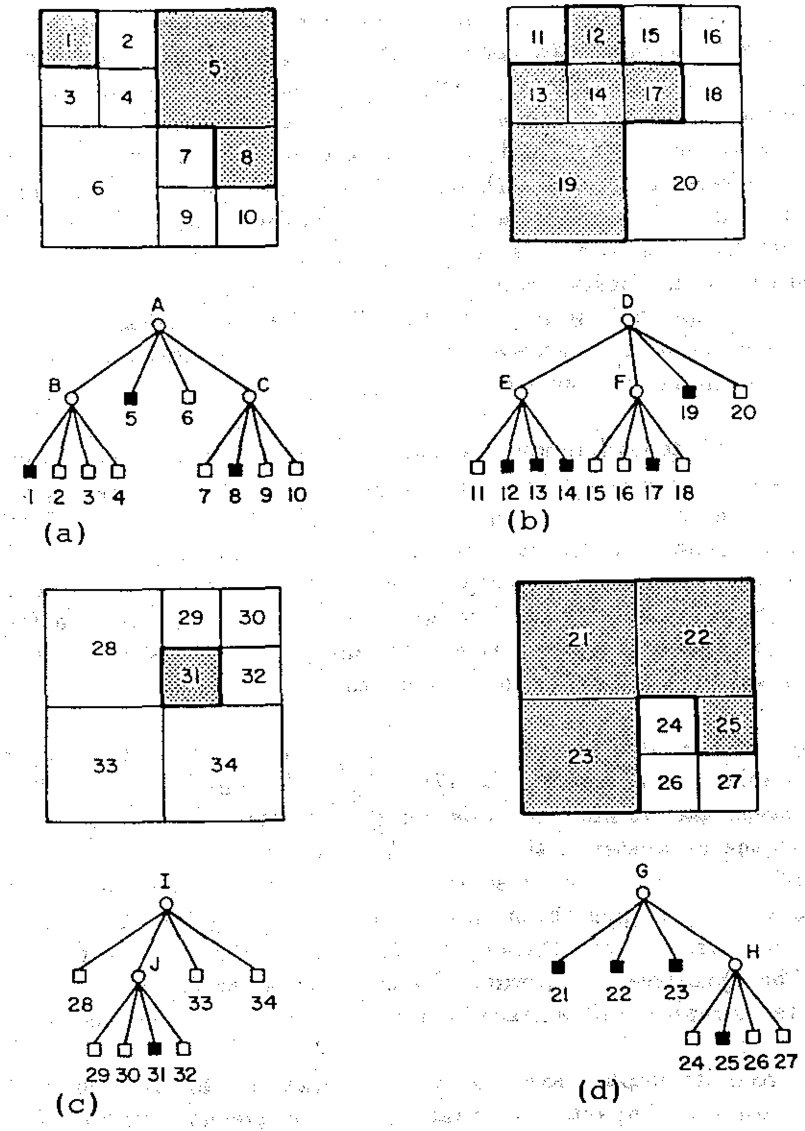

Figure 3. Example of set-theoretic operations. (a) sample image and its quadtree; (b) sample image and its quadtree; (c) intersection of the images in (a) and (b); union of the images in (a) and (b).

4.1. SET OPERATIONS

For a binary image, set-theoretic operations such as union and intersection are quite simple to implement [Hunt79, Shne81]. For example, the intersection of two quadtrees yields a BLACK node only when the corresponding regions in both quadtrees are BLACK. Figure 3c is the result of the intersection of the quadtrees of Figures 3a and 3b. This operation is performed by simultaneously traversing three quadtrees. The first two trees correspond to the trees being intersected while the third tree represents the result of the operation. At each step in the traversal one of the following actions is taken:

(1) If either input quadtree node is WHITE, then the output quadtree node is WHITE.

(2) If both input quadtree nodes are BLACK, then the output quadtree node is BLACK.

(3) If one input quadtree node is BLACK while the other input quadtree node is GRAY (i.e., an internal node), then the GRAY node's subtree is copied into the output quadtree.

(4) If both input quadtree nodes are GRAY, then the output quadtree node is GRAY, and these four actions are recursively applied to each pair of corresponding sons. Once the sons have been processed, we must check to see if they are all leaf nodes of the same color in which case a merge takes place (e.g., the sons of nodes B and E in Figures 3a and 3b respectively). Note that for the intersection operation, a merge of four BLACK leaf nodes is impossible and thus we must only check 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 union operation is implemented easily by applying DeMorgan's law to the above intersection algorithm. For example, Figure 3d is the result of the union of the quadtrees of Figures 3a and 3b. When the set-theoretic operations are interpreted as Boolean operations, union and intersection become "or" and "and" operations, respectively. Other operations, such as "exclusive or" and "set-difference", are coded in an analogous manner with linear-time algorithms. The extension of these algorithms to octrees is straightforward.

The ability to perform set operations quickly is one of the primary reasons for the popularity of quadtrees over alternative representations such as the chain code or vectors. The chain code can be characterized as a local data structure, since each segment of the chain code conveys information only about the part of the image to which it is adjacent - i.e., that the image is to its right. Performing an overlay operation on two images represented by chain codes thus requires a considerable amount of work. In contrast, the quadtree is a hierarchical data structure that yields successive refinements at lower levels in the tree.

4.2. BOTTOM-UP NEIGHBOR FINDING

Many quadtree algorithms involve more work than just traversing the tree. In particular, in several applications we must perform a computation at each node that depends on the values of its adjacent neighbors. Thus we must be able to locate these neighbors. There are several techniques for achieving this result. One approach [Klin79] makes use of the coordinates and the size of the node whose neighbor is being sought in order to compute the location of a point in the neighbor. For a $2^n \times 2^n$ image, this can require n steps corresponding to the path from the root of the quadtree to the desired neighbor. An alternative approach described below only makes use of father links and computes a direct path to the neighbor by following links in the tree. This method,

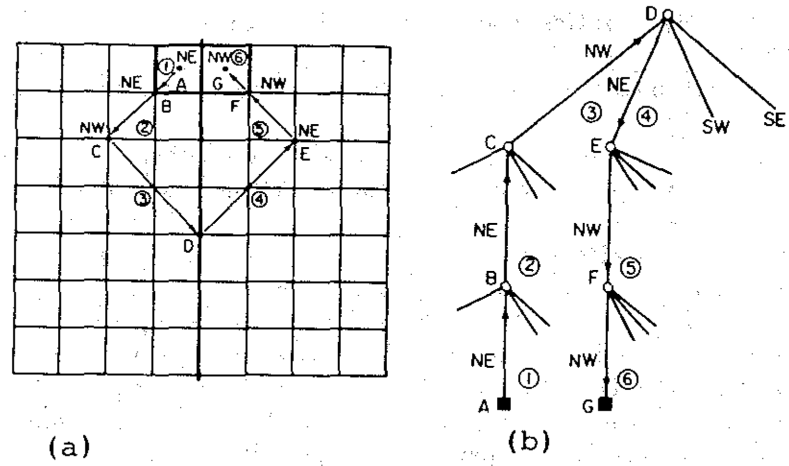

Figure 4. The process of locating the eastern neighbor of node A (i.e., G).
(a) Block decomposition; and (b) Tree representation.

termed *bottom-up neighbor finding,* has been shown to require an average of four links to be followed for each neighbor sought [Same82].

In this section we limit ourselves to neighbors in the horizontal and vertical direction that are of size equal to or greater than the node whose neighbor is being sought. This requires us to follow father links until a common ancestor of the two nodes is found. Once the common ancestor is located, we descend along a path that retraces the previous path with the modification that each step is a reflection of the corresponding prior step about the axis formed by the common boundary between the two nodes. The general flow of such an algorithm is given in Figure 4. For example, when attempting to locate the eastern neighbor of node A (i.e., node G) in Figure 4, node D is the common ancestor of nodes A and G, and the eastern edge of the block corresponding to node A is the common boundary between node A and its neighbor. The main idea behind bottom-up neighbor finding is understood by examining more closely how the nearest common ancestor of a node, say P, and its eastern neighbor of greater than or equal size, say Q, is located. In particular, the nearest common ancestor has P as one of the eastern-most nodes of one of its western subtrees, and Q as one of the western-most nodes of one of its eastern subtrees. Thus as long as an ancestor X is in a subtree that is not an eastern son (i.e., NE or SE), we must ascend the tree at least one more level before locating the nearest common ancestor. For neighbors in the diagonal direction, see [Same82].

4.3. CONSTRUCTING QUADTREES

Before we can operate on an image represented by a quadtree, we must first build the quadtree. This involves being able to convert between a number of different data formats and the quadtree. In this section we briefly describe the construction of region quadtrees from raster data.

When building a quadtree from raster data presented in raster scan order (i.e., the array is processed row by row) [Same81a] we use bottom-up neighbor-finding 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 use of predictive techniques that assume 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, merging is reduced and the algorithm's execution time is dominated by the number of blocks in the image [Shaf87] rather than by the number of pixels. However, this algorithm does require the use of an auxiliary 1-dimensional array of size equal to the width of the image.

Building an octree from a raster representation is a computationally expensive process because of the large amount of data that must be examined. This can be lessened in part by using the predictive techniques described above. However, the storage requirements for the auxiliary array are as large as a cross-section of the image, but they could be overcome by using a quadtree to represent the cross-section. Most often the octree is built from alternative 3-dimensional representations, such as the boundary method [Tamm84] and CSG trees [Same85c, Wood82], which are more compact.

4.4. POLYGON COLORING

Bottom-up neighbor finding can be used to implement the seed-filling approach to polygon coloring. The idea is to start at a block corresponding to the starting point and then use bottom-up neighbor finding to propagate the color to the remaining blocks in the polygon. Polygon coloring is a special case of connected component labeling. It is analogous to finding the connected components of a graph. In the case of a quadtree, the result is the assignment of a different label to each of the distinct BLACK regions in an image. Given a binary array representation of an image, the traditional method of performing this operation would be a "breadth-first" approach which scans the image row by row from left to right and assigns the same label to adjacent BLACK pixels that are found to the right and in the downward direction. During this process, pairs of equivalences may be generated, thus necessitating two more steps: one to merge the equivalences and the second to update the labels associated with the various pixels to reflect the merger of the equivalences.

Using a quadtree to perform the same operation involves an analogous three-step process [Same81b]. The first step is a tree traversal where for each BLACK node that is encountered, say A, all adjacent BLACK nodes on the southern and eastern sides of A are found, and assigned the same label as A. The adjacency exploration is done by using bottom-up neighbor finding. At times, the adjacent node may already have been assigned a label, in which case the equivalence is noted. The second step merges all the equivalence pairs generated during step one. The third step performs another traversal of the quadtree and updates the labels on the nodes to reflect the equivalences generated by the first two steps of the algorithm. Often, the first and second steps can be combined into one step [Same85a].

The execution time of this connected component labeling algorithm is almost linear in the number of BLACK blocks. Thus it is dependent only on the number of blocks in the image and not on their size. In contrast, the analogous algorithm for the binary array has an execution time that is proportional to the number of pixels and hence to the area of the blocks. Therefore, we see that the hierarchical structure of the quadtree data structure saves not only space but time.

4.5. DISPLAY

A basic graphics operation is the conversion of an internal model of a 3-dimensional scene into a 2-dimensional scene that lies on the viewplane for the purpose of display on a 2-dimensional screen. This is known as the hidden-surface operation. Although there are many possible mappings between a 3-dimensional space and a 2-dimensional space, in this section we only discuss projections. Each pixel of the viewplane determines a pyramid that is formed by the set of all rays originating at the viewpoint and intersecting the viewplane within the boundary of the pixel. In the simplest case, a color is assigned to each pixel that corresponds to the color of the object that is closest to the viewpoint while also lying within the pixel's pyramid.

There are three approaches to the hidden-surface task that are relevant to our discussion. First, the 3-dimensional scene can be viewed as a sequence of overlays of 2-dimensional scenes each of which is represented by a quadtree [Kauf83]. Second, quadtrees can be used to model the viewplane even when the 3-dimensional scene consists of polygons of arbitrary orientation and placement in the 3-dimensional space. This solution was first proposed by Warnock [Warn68] and is known as Warnock's algorithm. Third, the parametric space of the surface of a 3-dimensional object can be modeled by a quadtree [Catm75].

When an octree is used to represent a collection of objects, displaying it is straightforward. The easiest method is the parallel projection [Doct81]. Of course, implicit in this task is the solution of the hidden-surface problem for the interaction among the objects. Since the octree imposes a spatial ordering on objects, in this case the hidden-surface problem can be solved more efficiently than the general hidden-surface problem for arbitrary polygons. In particular, any opaque object in the four front octants of an octree will occlude any opaque object in the back four octants. This property holds recursively within each of the suboctants. The display of the scene is facilitated by the construction of a *display quadtree* which corresponds to a partial 2-dimensional view of the scene. The display quadtree is updated as the nodes of the octree are traversed from back-to-front. Of course, the nodes could also be processed from front-to-back, thereby allowing for the possibility of visiting fewer nodes. Generalizations of the parallel projection to planes of arbitrary position and orientation are described by Meagher [Meag82] and Yau [Yau84].

Although projection display techniques are suitable for computer-aided design, realistic modeling of lighting effects generally requires using some variant of raytracing. The necessary intersection calculations can be speeded up by using octrees or their variants. The key operation is to locate a neighboring node that contains a particular point. One approach is to calculate a point that lies in the neighbor and then search the octree for that point [Glas84, Kapl85, Wyvi85, Fuji86]. Alternatively, bottom-up neighbor-finding can be used [Jans86]. As the complexity of the scene increases, bottom-up neighbor-finding methods should be more efficient.

5. 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 points can grow and shrink at will) and on applications involving search. Although 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 data and R for region) [Oren82, Same84a]. It is an adaptation of the region quadtree to point data which associates data points with quadrants. The PR quadtree (see Figure 5) 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.

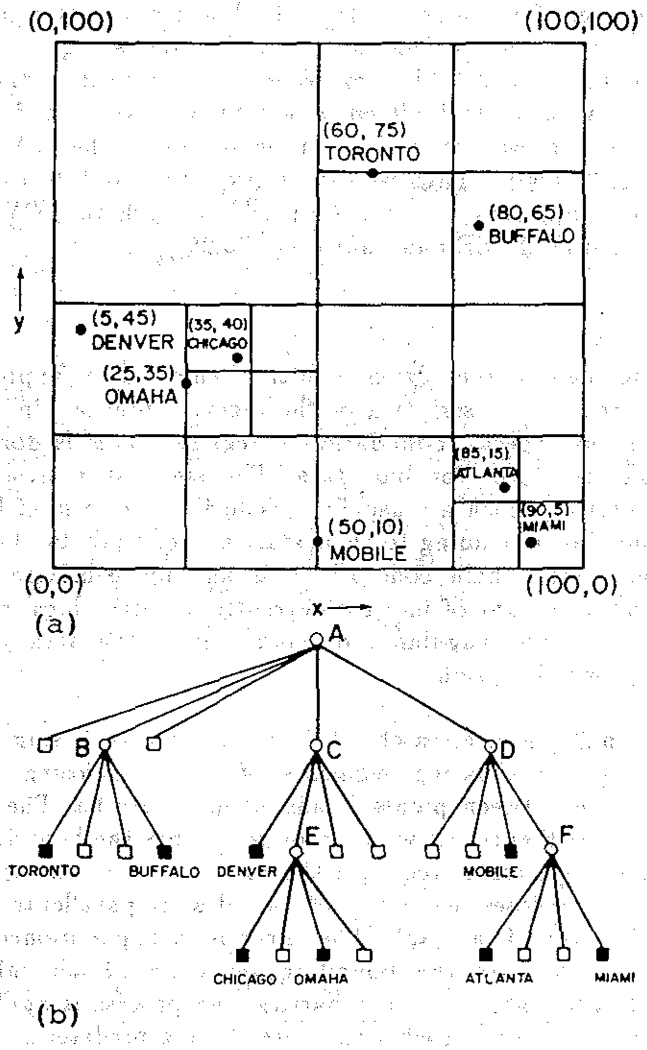

Figure 5. A PR quadtree (b) and the records it represents (a).

Data points are inserted into PR quadtrees by searching 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.

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 5 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)).

6. LINE DATA

Section 4 was devoted to the region quadtree which is an approach to region representation that is based on a description of the region's interior. In this section we focus on representations that specify boundaries of regions. This is done in the more general context of data structures for line data. The simplest representation is the polygon in the form of vectors which are usually specified in the form of lists of pairs of x and y coordinate values corresponding to their start and end points. One of the most common representations is the chain code which is an approximation of a polygon. There also is a considerable amount of interest currently in hierarchical representations. These are primarily based on rectangular approximations to the data as well as on a regular decomposition in two dimensions.

The *strip tree* [Ball81] is a hierarchical representation of a single curve that is obtained by successively approximating segments of it by enclosing rectangles. For example, consider the curve between points P and Q in Figure 6a. The data structure consists of a binary tree (see Figure 6b) whose root represents the bounding rectangle of the entire curve. The rectangle associated with the root, A in this example, corresponds to a rectangular strip, that encloses the curve, whose sides are parallel to the line joining the endpoints of the curve (i.e., P and Q). The curve is then partitioned in two at one of the locations where it touches the bounding rectangle. Each subcurve is then surrounded by a bounding rectangle and the partitioning process is applied recursively. This process stops when the width of each strip is less than a predetermined value.

Like point and region quadtrees, strip trees are useful in applications involving search and set operations. For example, suppose we wish to determine whether a road crosses a river. Representing such features with a strip tree, answering this query

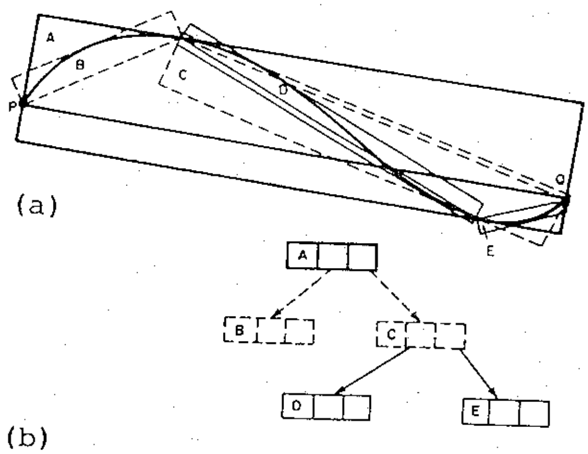

Figure 6. A curve between points P and Q. (a) Its decomposition into strips; and (b) the corresponding strip tree.

requires performing an intersection of the corresponding strip trees. Three cases are possible as shown in Figure 7. Figures 7a and 7b correspond to the answers NO and YES respectively while Figure 7c requires us to descend further down the strip tree. This method saves a lot of work when an intersection is impossible.

The PM quadtree of Samet and Webber [Same85b] (see also edge-EXCELL [Tamm81]) is an adaptation of the region quadtree to represent collections of polygons (termed *polygonal maps*). There are a number of variants of the PM quadtree. The one described is based on a decomposition rule that stipulates that partitioning occurs as long as a quadrant contains more than one line segment unless the line segments are all incident at the same vertex which is also in the same quadrant (e.g., Figure 8). We 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 a block contains a vertex). In such a case, each q-edge is

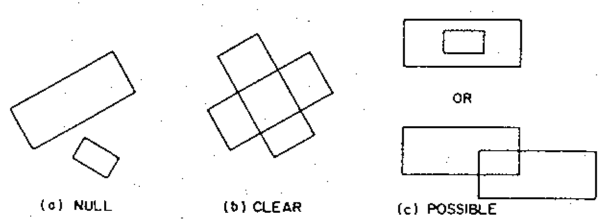

Figure 7. Three possible results of intersecting two strip trees.
(a) Null. (b) Clear. (c) Possible.

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. 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 that they pass through.

The PM quadtree has also been adapted to represent polyhedra [Ayal85, Carl85, Fuji85]. Decomposition stops when a node is intersected by exactly one edge, or one face, or all edges that intersect the node meet at a common vertex in the node. For example, Figure 9b is the PM octree corresponding to the 3-dimensional object in Figure 9a. This representation is considerably more compact than the raster octree.

7. CONCLUDING REMARKS

The use of hierarchical data structures enables us to focus computational resources on the interesting subsets of data. Thus there is no need to expend work where the payoff is small. Moreover, algorithms based on such methods are easy to develop and maintain. When the hierarchical data structures are based on the principle of recursive decomposition, we have a spatial index. All features, be they regions, points, or lines, are represented with respect to a common origin. A system representing geographic information that makes use of the region quadtree, PR quadtree, and PM quadtree to represent region, point, and line data respectively has been built [Same84b].

The disadvantage of quadtree methods is that they are shift sensitive in 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].

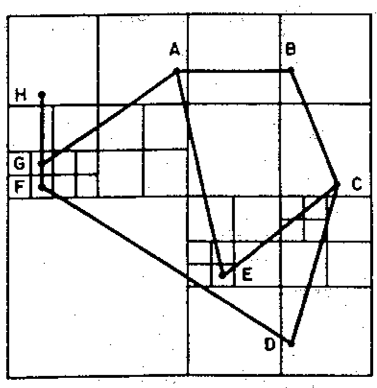

Figure 8. A PM quadtree.

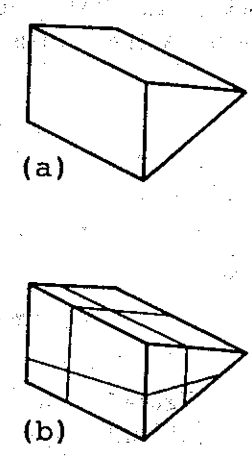

Figure 9. (a) Example 3-dimensional object; (b) its corresponding PM octree.

Another disadvantage for some applications (not display) is that some variants of the quadtree and octree are approximations. The fact that we are working in a digitized space also may 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. Interestingly, the impact of both the digitization and shifting problems is lessened considerably by use of representations such as the PM quadtree for 2-dimensional and 3-dimensional data, in which case the representation is not an approximation.

ACKNOWLEDGMENTS

I have benefited greatly from discussions with Robert E. Webber.

REFERENCES

1. [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.

2. [Ball81] - 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).

3. [Bell83] - S.B.M. Bell, B.M. Diaz, F. Holroyd, and M.J. Jackson, Spatially referenced methods of processing raster and vector data, *Image and Vision Computing 1*, 4(November 1983), 211-220.

4. [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.

5. [Catm75] - E. Catmull, Computer display of curved surfaces, *Proceedings of the Conference on Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, May 1975, 11-17.

6. [Doct81] - L.J. Doctor and J.G. Torborg, Display techniques for octree-encoded objects, *IEEE Computer Graphics and Applications 1*, 1(July 1981), 39-46.

7. [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.

8. [Fuji86] - A. Fujimoto, T. Tanaka, and K. Iwata, ARTS: Accelerated ray-tracing system, *IEEE Computer Graphics and Applications 6*, 4(April 1986), 16-26.

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

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

11. [Glas84] - A.S. Glassner, Space subdivision for fast ray tracing, *IEEE Computer Graphics and Applications 4*, 10(October 1984), 15-22.

12. [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.

13. [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.

14. [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.

15. [Jans86] - F.W. Jansen, Data structures for ray tracing, *Data Structures for Raster Graphics* (F.J. Peters, L.R.A. Kessener, and M.L.P. van Lierop, Eds.), Springer Verlag, Berlin, 1986, 57-73.

16. [Kapl85] - M.R. Kaplan, Space-tracing: a constant time ray-tracer, SIGGRAPH'85 Tutorial on the Uses of Spatial Coherence in Ray-Tracing, San Francisco, ACM, July 1985.

17. [Kauf83] - A. Kaufman, D. Forgash, and Y. Ginsburg, Hidden surface removal using a forest of quadtrees, *Proceedings of Conference on Image Processing, Computer Graphics, and Pattern Recognition*, Beer-Sheva, Israel, June 1983, 85-89.

18. [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.

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

20. [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.

21. [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.

22. [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.

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

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

25. [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).

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

27. [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.

28. [Same81a] - H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 3*, 1(January 1981), 93-95.

29. [Same81b] - H. Samet, Connected component labeling using quadtrees, *Journal of the ACM 28*, 3(July 1981), 487-501.

30. [Same82] - H. Samet, Neighbor finding techniques for images represented by quadtrees, *Computer Graphics and Image Processing 18*, 1(January 1982), 37-57.

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

32. [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.

33. [Same85a] - H. Samet and M. Tamminen, Efficient component labeling of images of arbitrary dimension, Computer Science TR-1480, University of Maryland, College Park, MD, February 1985 (subsumes TR-1420).

34. [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).

35. [Same85c] - H. Samet and M. Tamminen, Bintrees, CSG trees, and time, *Computer Graphics 19*, 3(July 1985), 121-130 (also *Proceedings of the SIGGRAPH'85 Conference*, San Francisco, July 1985; and University of Maryland Computer Science TR-1472).

36. [Same86] - H. Samet and R.E. Webber, A comparison of the space requirements of multi-dimensional quadtree-based file structures Computer Science TR-1711, University of Maryland, College Park, MD, September 1986.

37. [Same87] - H. Samet and R.E. Webber, Hierarchical data structures and algorithms for computer graphics, Computer Science TR-1752, University of Maryland, College Park, MD, January 1987.

38. [Shaf87] - C.A. Shaffer and H. Samet, Optimal quadtree construction algorithms, to appear in *Computer Vision, Graphics, and Image Processing.*

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

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

41. [Tamm84] - M. Tamminen and H. Samet, Efficient octree conversion by connectivity labeling, *Computer Graphics 18*, 3(July 1984), pp. 43-51 (also *Proceedings of the SIGGRAPH'84 Conference*, Minneapolis, July 1984).

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

43. [Warn68] - J.E. Warnock, A hidden line algorithm for halftone picture representation, Computer Science Department TR 4-5, University of Utah, Salt Lake City, May 1968.

44. [Wood82] - J.R. Woodwark and K.M. Quinlan, Reducing the effect of complexity on volume model evaluation, *Computer-aided Design 14*, 2(1982), 89-95.

45. [Wyvi85] - G. Wyvill and T.L. Kunii, A functional model for constructive solid geometry, *The Visual Computer 1*, 1(July 1985), 3-14.

46. [Yama84] - K. Yamaguchi, T.L. Kunii, K. Fujimura, and H. Toriya, Octree-related data structures and algorithms, *IEEE Computer Graphics and Applications 4*, 1(January 1984), 53-59.

47. [Yau84] - M. Yau, Generating quadtrees of cross-sections from octrees, *Computer Vision, Graphics, and Image Processing 27*, 2(August 1984), 211-238.

Hanan Samet received the B.S. degree in engineering from the University of California, Los Angeles, and the M.S. Degree in operations research and the M.S. and Ph.D. degrees in computer science from Stanford University, Stanford, CA.

In 1975 he joined the Computer Science Department at the University of Maryland, College Park, where he is now a Professor. He also serves as the Director of the Graduate Program in Computer Science, is a member of the Computer Vision Laboratory of the Center for Automation Research, and has an appointment in the University of Maryland Institute for Advanced Computer Studies.

His research interests are data structures, computer graphics, geographic information systems, computer vision, robotics, programming languages, artificial intelligence, and database management systems.