Samet, Hanan (Maryland)

Hierarchical Data Structures for Spatial Databases

1. Introduction

Spatial data consists of points, lines, regions, rectangles, surfaces, volumes, and even data of higher dimension which includes time. Spatial databases permit the storage of spatial information about objects. Spatial databases are finding increasing use in applications in environmental monitoring, space, urban planning, resource management, and geographic information systems (GIS) (BUCHMANN et al. 1990, GÜNTHER et al. 1991). In spatial databases it is often desirable to attach attribute information such as elevation heights, city names, etc. to objects appearing in maps. On the other hand, in many standard database applications, it is useful to add spatial attributes to describe different objects in the database such as the extent of a given river, or the boundary of a given county, etc. Many queries can be answered more efficiently when spatial and nonspatial information are combined.

Many prototype systems have been proposed to store spatial data along with the nonspatial data describing it in a relational database management system (ABEL 1988, GÜTING et al. 1989, LORIE et al. 1984, ORENSTEIN 1988, ROUSSOPOULOS et al. 1988, SACKS-DAVIS et al. 1987, WOLF 1990). At the same time, there has been much research in the general database area. Many general techniques have been developed [such as extensible database management systems - e.g., (CAREY et al. 1988, STONEBRAKER et al. 1986), and object-oriented database systems - e.g., (KIM et al. 1990, DEUX et al. 1990)]. However, they are not well-suited for spatial data. The problem is that they often reduce all spatial data to points in a higher dimensional space. For example, a line could be represented by its endpoints. This means that the line is represented by a tuple of four items (a pair of x coordinates and a pair of y coordinates). Thus, in effect, we have constructed a mapping from a two-dimensional space (i.e., the space from which the lines are drawn) to a four-dimensional space (i.e., the space containing the representative point corresponding to the line). These values are then stored in records in the system in the same way as nonspatial attributes. Such an approach is fine for storage purposes. However, it is not good for spatial operations involving search. The problem is that much time is wasted on the retrieval step since both spatial and nonspatial information is retrieved although often only the spatial portion is needed. Moreover, such an approach ignores the geometry inherent in the data (e.g., the fact that a line passes through a particular region).

For example, suppose we want to detect if two lines are near each other, or, alternatively, to find the nearest line to a given line. This is difficult to do in the four-dimensional space since proximity in the two-dimensional space from which the lines are drawn is not necessarily preserved in the four-dimensional space. In other words, although the two lines may be very close to each other, the Euclidean distance between their representative points may be quite large. Thus we need special representations for spatial data.

One approach to the representation of spatial data is to separate it structurally from the nonspatial data while maintaining appropriate links between the two (AREF et al. 1990b, AREF et al. 1991a). This leads to a much higher bandwidth for the retrieval of the spatial data. Thus we will perform the spatial operations directly on the spatial data structures. This gives the freedom to choose a more appropriate spatial structure than the imposed non-spatial structure (e.g., a relational database). In such a case, a spatial processor can be used that is specifically designed for efficiently dealing with the part of the queries that involve proximity relations and search, and a relational database management system for the part of the queries that involve non-spatial data. Its proper functioning depends on the existence of a query optimizer to determine the appropriate processor for each part of the query (AREF et al. 1991b).

As an example of the type of query we would like to be able to answer consider a request to "find the names of the roads that pass through the University of Maryland region". This requires that we extract the region locations of all the database records whose "region name" field has the value "University of Maryland" and build a map, say *A*. Next, we intersect map *A* with the road map, say *B*, to yield a new map, say *C*, with the selected roads. We now create a new relation in our database which has just one attribute which is the relevant road names of the roads in map *C*. There are many approaches to answering the above query. Their efficiency depends on the nature of the data and its volume.

In the rest of this review we concentrate on the data structures used by the spatial processor. In particular, we focus on hierarchical data structures. They are based on the principle of recursive decomposition (similar to *divide and conquer* methods). The term *quadtree* is often used to describe this class of data structures, and this is the focus of this paper. We concentrate primarily on region data. For a more extensive treatment of this subject, see SAMET 1984a, SAMET 1988a, SAMET et al. 1988b, SAMET et al. 1988c, SAMET 1990a, SAMET 1990b, SAMET et al. 1991.

Our presentation is organized as follows. Section 2 briefly reviews the historical background of the origins of hierarchical data structures. Section 3 discusses some key operations on region data. Sections 4, 5, and 6 describe hierarchical representations for point, rectangle, and line data, respectively, as well as give examples of their utility. Section 7 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 1 x 1 blocks) 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.

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. and SHAFFER et al. 1990b.

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 (1971) who used the term Q-tree (KLINGER et al. 1976), whereas HUNTER (1978) 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 (1974). 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 (1966) used it as a means of indexing into a geographic database. WARNOCK (1969) 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 (1976) used the quadtree as an initial step in a "split and merge" image segmentation algorithm.

The pyramid of TANIMOTO and PAVLIDIS (1975) is a close relative of the region quadtree. It is a multiresolution representation which 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. Recall that in contrast, the region quadtree is a variable resolution data structure.

The distinction between a quadtree and a pyramid can be easily seen by considering the types of spatial queries. There are two principal classes (AREF et al. 1990a). The first is location-based. In this case, we are searching for the nature of the feature associated with a particular location or in its proximity. For example, "what is the feature at location X?", "what is the nearest city to location X?", or "what is the nearest road to location X?" The second is feature-based. In this case, we are probing for the presence or absence of a feature, as well as seeking its actual location. For example, "does wheat grow anywhere in California?", "what crops grow in California?", or "where is wheat grown in California?"

Location-based queries are easy to answer with a quadtree representation [e.g., the QUILT system (SHAFFER et al 1990b)]. On the other hand, feature-based queries are more difficult. The problem is that there is no indexing by features. The indexing is only based on spatial occupancy. The goal is to process the query without examining every location in space. The pyramid is useful for such queries since the nodes that are not at the maximum level of resolution (i.e., at the bottom level) contain summary information (AREF et al. 1990a, AREF et al. 1991c). Thus we could view these nodes as feature vectors which indicate whether or not a feature is present at a higher level of resolution. Therefore, by examining the root of the pyramid (i.e., the node that represents the entire image), we can quickly tell if a feature is present without having to examine every location.

Quadtree-like data structures can also be used to represent images in three dimensions and higher. The octree (HUNTER 1978, JACKINS et al. 1980, MEAGHER 1982, REDDY et al. 1978) data structure is the three-dimensional analogon of the quadtree. Therefore 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. ABEL et al. 1983, GARGANTINI 1982, KLINGER et al. 1979, MORTON 1966).

The second, termed a "DF-expression", represents the image in the form of a traversal of the nodes of its quadtree (KAWAGUCHI et al. 1980). 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 (1989) 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* (HUNTER 1978, HUNTER et al. 1979) 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 $16q - 11 + 16p$.

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 two 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 (MEAGHER 1980 and 1982) 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 that is proportional to the number of blocks in the image rather than the number of 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 operations required for answering spatial queries can be implemented using region quadtrees. In particular, we discuss point and object location and set operations. Object location is important in detecting the feature associated with a given location or its neighbors. Set operations form the basis of most complicated queries. For example, to "find the names of the roads that pass through the University of Maryland region," we will need to intersect a region map with a line map.

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 nea-

rest 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 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. For a treatment of the problem for lines, see HOEL and SAMET 1991.

3.2. Set operations

For a binary image, set-theoretic operations such as union and intersection are quite simple to implement (HUNTER 1978, HUNTER et al 1979, SHNEIER 1981). 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 (SHAFFER et al. 1990b). 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°) (SAMET et al. 1988b).

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 (1979). In this section we discuss the PR quadtree (P for point and R for region) (ORENSTEIN 1982, SAMET 1990a).

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 4 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.

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 4 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. Rectangle data

The rectangle data type lies somewhere between the point and region data types. Rectangles are often used to approximate other objects in an image for which they serve as the minimum rectilinear enclosing object. For example, bounding rectangles can be used in cartographic applications to approximate objects such as lakes, forests, hills, etc. (MATSUYAMA et al. 1984). In such a case, the approximation gives an indication of the existence of an object. Of course, the exact boundaries of the object are also stored; but they are only accessed if greater precision is needed. For such applications, the number of elements in the collection is usually small, and most often the sizes of the rectangles are of the same order of magnitude as the space from which they are drawn.

Rectangles are also used in VLSI design rule checking as a model of chip components for the analysis of their proper placement. Again, the rectangles serve as minimum enclosing objects. In this application, the size of the collection is quite large (e.g., millions of components) and the sizes of the rectangles are several orders of magnitude smaller than the space from which they are drawn. Regardless of the application, the representation of rectangles involves two principal issues (SAMET 1988a). The first is how to represent the individual rectangles and the second is how to organize the collection of the rectangles.

The representation that is used depends heavily on the problem environment. If the environment is static, then frequently the solutions are based on the use of the plane-sweep paradigm (PREPARATA et al. 1985), which usually yields optimal solutions in time and space. However, the addition of a single object to the database forces the re-execution of the algorithm on the entire database. We are primarily interested in dynamic problem environments. The data structures that are chosen for the collection of the rectangles are differentiated by the way in which each rectangle is represented.

One representation reduces each rectangle to a point in a higher dimensional space, and then treats the problem as if we have a collection of points. This is the approach of HINRICHS and NIEVERGELT (1983) and HINRICHS (1985). Each rectangle is a Cartesian product of two one-dimensional intervals where each interval is represented by its centroid and extent. The collection of rectangles is, in turn, represented by a grid file (NIEVERGELT et al. 1984), which is a hierarchical data structure for points.

The second representation is region-based in the sense that the subdivision of the space from which the rectangles are drawn depends on the physical extent of the rectangle - not just one point. Representing the collection of rectangles, in turn, with a tree-like data structure has the advantage that there is a relation between the depth of node in the tree and the size of the rectangle(s) that are associated with it. Interestingly, some of the region-based solutions make use of the same data structures that are used in the solutions based on the plane-sweep paradigm. In the remainder of this section, we give an example of a pair of region-based representations.

The *MX-CIF quadtree* of KEDEM (1982) (see also ABEL and SMITH 1983) is a region-based representation where each rectangle is associated with the quadtree node corresponding to the smallest block which contains it in its entirety. Subdivision ceases whenever a node's block contains no rectangles. Alternatively, subdivision can also cease once a quadtree block is smaller than a predetermined threshold size. This threshold is often chosen to be equal to the expected size of the rectangle (KEDEM 1982). For example, Figure 5 is the MX-CIF quadtree for a collection of rectangles. Note that rectangle F occupies an entire block and hence it is associated with the block's father. Also rectangles can be associated with both terminal and non-terminal nodes.

It should be clear that more than one rectangle can be associated with a given enclosing block and, thus, often we find it useful to be able to differentiate

between them. Kedem proposes to do so in the following manner. Let *P* be a quadtree node with centroid (*CX, CY*), and let *S* be the set of rectangles that are associated with *P*. Members of *S* are organized into two sets according to their intersection (or collinearity of their sides) with the lines passing through the centroid of *P*'s block - i.e., all members of *S* that intersect the line *x=CX* form one set and all members of S that intersect the line *y=CY* form the other set.

If a rectangle intersects both lines (i.e., it contains the centroid of *P*'s block), then we adopt the convention that it is stored with the set associated with the line through *x=CX*. These subsets are implemented as binary trees (really tries), which in actuality are one-dimensional analogs of the MX-CIF quadtree. For example, Figure 6 illustrates the binary tree associated with the y axes passing through the root and the NE son of the root of the MX-CIF quadtree of Figure 5. Interestingly, the MX-CIF quadtree is a two-dimensional analog of the interval tree (EDELSBRUNNER 1980, McCREIGHT 1980), which is a data structure that is used to support optimal solutions based on the plane-sweep paradigm to some rectangle problems.

The R-tree (GUTTMAN 1984, BECKMANN et al. 1990) is a hierarchical data structure that is derived from the B-tree (COMER 1979). Each node in the tree is a *d*-dimensional rectangle corresponding to the smallest rectangle that encloses its son nodes which are also *d*-dimensional rectangles. The leaf nodes are the actual rectangles in the database. Often, the nodes correspond to disk pages and, thus, the parameters defining the tree are chosen so that a small number of nodes is visited during a spatial query. Note that rectangles corresponding to different nodes may overlap.

Also, a rectangle may be spatially contained in several nodes, yet it can only be associated with one node. This means that a spatial query may often require several nodes to be visited before ascertaining the presence or absence of a particular rectangle. This problem can be alleviated by using the R^+-tree (FALOUTSOS et al. 1987, SELLIS et al. 1987) for which all bounding rectangles (i.e., at levels other than the leaf) are non-overlapping. This means that a given rectangle will often be associated with several bounding rectangles. In this case, retrieval time is sped up at the cost of an increase in the height of the tree. Note that B-tree performance guarantees are not valid for the R^+-tree - i.e., pages are not guaranteed to be 50 % full without very complicated record update procedures.

6. 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 (SAMET et al. 1985, NELSON et al. 1986), see also edge-EXCELL (TAMMINEN 1981) 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 (1985) 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 (1987) 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 (AYALA et al. 1985, CARLBOM et al. 1985, FUJIMURA et al. 1985, HUNTER 1981, NAVAZO et al. 1986b, QUINLAN et al. 1982, TAMMINEN 1981, VANDERSCHEL 1984). 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. Another approach (NAVAZO et al. 1986b, BRUNET et al. 1987) extends the concepts of face node, edge node, and vertex node to handle faces represented by biquadratic Bezier primitives. The use of bi-

quadratic Bezier patches enables a better fit with fewer primitives than can be obtained with planar faces, thereby reducing the size of the octree.

The PMR quadtree (NELSON et al. 1986) 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 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.

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 (NELSON et al. 1986). 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.

7. Concluding remarks

The use of hierarchical data structures in spatial databases enables the focusing 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, known as QUILT, has been built (SAMET et al. 1984b, SHAFFER et al. 1990b) 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 (COMER 1979). 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 (LI et al. 1982).

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.

8. References

ABEL D J (1988), Relational data management facilities for spatial information systems. In: Proc. of the Third Int. Symp on Spatial Data Handling, pp 9 - 18. Sydney, Australia.

ABEL D J, SMITH J L (1983), A data structure and algorithm based on a linear key for a rectangle

retrieval problem. In: Comp. Vision, Graphics, and Image Processing, 24, 1, pp 1 - 13.

AREF W G, SAMET H (1990a), Efficient processing of window queries in the pyramid data structure. In: Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS), pp 265 - 272. Nashville, Tennessee.

AREF W G, SAMET H (1990b), Design of an integrated database system to support geographical applications. In: Proc. of the Fourth Int. Symp. on Spatial Data Handling, pp 589 - 598. Zurich.

AREF W G, SAMET H (1991a), Extending a DBMS with spatial operations. In: GÜNTHER O, SCHEK H J (ed.), Advances in Spatial Databases - 2nd Symp., SSD'91, Lecture Notes in: Comp. Science, 525, pp 299 - 318. Springer-Verlag, Berlin.

AREF W G, SAMET H (1991c), Optimization strategies for spatial query processing. In: LOHMAN G (ed.), Proc. of the Seventeenth Int. Conf. on Very Large Data Bases, pp 81 - 90. Barcelona.

AREF W G, SAMET H (1991c), Loading spatial features into the incomplete pyramid data structure. In: Proc. of the Int. Workshop on DBMS's for Geographical Applications. Capri, Italy.

AYALA D, BRUNET P, JUAN R, NAVAZO I (1985), Object representation by means of nonminimal division quadtrees and octrees. In: ACM Transactions on Graphics, 4, 1, pp 41 - 59.

BECKMANN N, KRIEGEL H P, SCHNEIDER R, SEEGER B (1990), The R*-tree: an efficient and robust access method for points and rectangles. In: Proc. of the SIGMOD Conf., pp 322 - 331. Atlantic City, NJ.

BRUNET P, AYALA D (1987), Extended octtree representation of free form surfaces. In: Comp.-Aided Geometric Design, 4, 1-2, pp 141 - 154.

BUCHMANN A, GÜNTHER O, SMITH T R, WANG Y-F (eds.) (1990), Design and Implementation of Large Spatial Databases. In: Lecture Notes in Comp. Science, No. 409. Springer-Verlag, Berlin.

CAREY M, DeWITT D, GRAEFE G, HAIGHT D, RICHARDSON J, SCHUH D, SHEKITA E, VANDENBERG S (1988), The EXODUS extensible DBMS project: an overview, Techn. Report 808. Univ. of Wisconsin, Madison, WI.

CARLBOM I, CHAKRAVARTY I, VANDERSCHEL D (1985), A hierarchical data structure for representing the spatial decomposition of 3-D objects. In: IEEE Comp. Graphics and Applications, 5, 4, pp 24 - 31.

COMER D (1979), The Ubiquitous B-tree. In: ACM Comp. Surveys, 11, 2, pp 121 - 137.

DEUX O et al. (1990), The story of O2. In: IEEE Transactions on Knowledge and Data Engineering, 2, 1, pp 91 - 108.

EDELSBRUNNER H (1980), Dynamic rectangle intersection searching. In: Inst. for Inform. Proc. Report, 47. Techn. Univ. of Graz, Graz.

FALOUTSOS C, SELLIS T, ROUSSOPOULOS N (1987), Analysis of object oriented spatial access methods. In: Proc. of the SIGMOD Conf., pp 426 - 439. San Francisco.

FINKEL R A, BENTLEY J L (1974), Quad trees: a data structure for retrieval on composite keys. In: Acta Informatica, 4, 1, pp 1 - 9.

FUJIMURA K, KUNII T L (1985), A hierarchical space indexing method. In: Proc. of Comp. Graphics '85, T1-4, pp 1 - 14. Tokyo.

GARGANTINI I (1982), An effective way to represent quadtrees. In: Communications of the ACM, 25, 12, pp 905 - 910.

GÜNTHER O, SCHEK H J (eds.) (1991), Advances in Spatial Databases - 2nd Symp., SSD'91. In: Lecture Notes in Comp. Science, 525. Springer-Verlag, Berlin.

GÜTING R H (1989), Gral: an Extensible relational system for geometric applications. In: APERS P M G, WIEDERHOLD G (eds.), Proc. of the Fifteenth Int. Conf. on Very Large Data Bases, pp 33 - 44. Amsterdam.

GUTTMAN A (1984), R-trees: a dynamic index structure for spatial searching. In: Proc. of the SIGMOD Conf., pp 47 - 57. Boston.

HINRICHS K (1985), The grid file system: implementation and case studies of applications. Ph.D. diss., Inst. f. Informatik, ETH, Zurich.

HINRICHS K, NIEVERGELT J (1983), The grid file: a data structure designed to support proximity queries on spatial objects. In: NAGL M, PERL J (eds.), Proc. of the WG '83 (Int. Workshop on Graphtheoretic Concepts in Comp. Science), pp 100 - 113. Trauner Verlag, Linz.

HOEL E G, SAMET H (1991), Efficient processing of spatial queries in line segment databases. In: GÜNTHER O, SCHEK H J (eds.), Advances in Spatial Databases - 2nd Symp., SSD'91, Lecture Notes in Comp. Science, 525, pp 237 - 256. Springer-Verlag, Berlin.

HOROWITZ S L, PAVLIDIS T (1976), Picture segmentation by a tree traversal algorithm. In: Journ. of the ACM, 23, 2, pp 368 - 388.

HUNTER G M (1978), Efficient computation and data structures for graphics. Ph.D. diss., Dep. of Electrical Engineering and Comp. Science. Princeton Univ., Princeton, NY.

HUNTER G M (1981), Geometrees for interactive visualization of geology: an evaluation. System Science Dep., Schlumberger-Doll Research, Ridgefield, CT.

HUNTER G M, STEIGLITZ K (1979), Operations on images using quad trees. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 1, 2, pp 145 - 153.

JACKINS C L, TANIMOTO S L (1980), Oct-trees and their use in representing three-dimensional objects. In: Comp. Graphics and Image Proc., 14, 3, pp 249 - 270.

KAWAGUCHI E, ENDO T (1980), On a method of binary picture representation and its application to data compression. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2, 1, pp 27 - 35.

KEDEM G (1982), The Quad-CIF tree: a data structure for hierarchical on-line algorithms. In: Proc. of the Nineteenth Design Automation Conf., pp 352 - 357. Las Vegas.

KIM W W, GARZA J F, BALLOU N, WOELK D (1990), Architecture of the ORION next-generation database system. In: IEEE Transactions on Knowledge and Data Engineering, 2, 1, pp 109 - 124.

KLINGER A (1971), Patterns and search statistics. In: RUSTAGI J S (ed.), Optimizing Methods in Statistics, pp 303 - 337. Academic Press, New York.

KLINGER A, DYER C R (1976), Experiments in picture representation using regular decomposition. In: Comp. Graphics and Image Proc., 5, 1, pp 68 - 105.

KLINGER A, RHODES M L (1979), Organization and access of image data by areas. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, 1, 1, pp 50 - 60.

LI M, GROSKY W I, JAIN R (1982), Normalized quadtrees with respect to translations. In: Comp. Graphics and Image Proc., 20, 1, pp 72 - 81.

LORIE R, MEIER A (1984), Using a relational DBMS for geographical databases. In: Geo-Proc., 2, pp 243 - 257.

MATSUYAMA T, HAO L V, NAGAO M (1984), A file organization for geographic information systems based on spatial proximity. In: Comp. Vision, Graphics, and Image Proc., 26, 3, pp 303 - 318.

McCREIGHT E M (1980), Efficient algorithms for enumerating intersecting intervals and rectangles. Xerox Palo Alto Research Center Report CSL-80-09, Palo Alto, CA.

MEAGHER D (1980), Octree encoding: a new technique for the representation, the manipulation, and display of arbitrary 3-d objects by computer. Electrical and Systems Engineering Techn. Report IPL-TR-80-111. Rensselaer Polytechnic Inst., Troy, NY.

MEAGHER D (1982), Geometric modeling using octree encoding. In: Comp. Graphics and Image Proc., 19, 2, pp 129 - 147.

MORTON G M (1966), A computer oriented geodetic data base and a new technique in file sequencing. IBM Ltd., Ottawa, Canada.

NAVAZO I (1986a), Contribució a les tècniques de modelat geomètric d'objectes polièdrics usant

la codificació amb arbres octals. Ph.D. diss., Escola Tècnica Superior d'Enginyers Industrials, Dep. de Metodes Informatics, Univ. Politècnica de Catalunya, Barcelona, Spain.

NAVAZO I, AYALA D, BRUNET P (1986b), A geometric modeller based on the exact octree representation of polyhedra. In: Comp. Graphics Forum, 5, 2, pp 91 - 104.

NELSON R C, SAMET H (1986), A consistent hierarchical representation for vector data. In: Comp. Graphics, 20, 4, pp 197 - 206 (also in: Proc. of the SIGGRAPH'86 Conf., Dallas).

NIEVERGELT J, HINTERBERGER H, SEVCIK K C (1984), The grid file: an adaptable, symmetric multikey file structure. In: ACM Transactions on Database Systems, 9, 1, pp 38 - 71.

ORENSTEIN J A (1982), Multidimensional tries used for associative searching. In: Inform. Proc. Letters, 14, 4, pp 150 - 157.

ORENSTEIN J A, MANOLA F A (1988), PROBE: spatial data modeling and query processing in an image database application. In: IEEE Transactions on Software Engineering, 14, 5, pp 611 - 629.

PREPARATA F P, SHAMOS M I (1985), Computational Geometry: An Introduction. Springer-Verlag, New York.

QUINLAN K M, WOODWARK J R (1982), A spatially-segmented solids database - justification and design. In: Proc. of CAD'82 Conf., pp 126 - 132. Butterworth, Guildford, Great Britain.

REDDY D R, RUBIN S (1978), Representation of three-dimensional objects. CMU-CS-78-113, Comp. Science Dep., Carnegie-Mellon Univ., Pittsburgh.

ROUSSOPOULOS N, FFALOUTSOS, SELLIS T (1988), An efficient pictorial database system for PSQL. In: IEEE Transactions on Software Engineering, 14, 5, pp 639 - 650.

SACKS-DAVIS R, McDONELL K J, OOI B C (1987), GEOQL - a query language for geographic information systems. Techn. Report, 2. Monash Univ., Victoria, Australia.

SAMET H (1984a), The quadtree and related hierarchical data structures. In: ACM Comp. Surveys, 16, 2, pp 187 - 260.

SAMET H (1988a), Hierarchical representations of collections of small rectangles. In: ACM Comp. Surveys, 20, 4, pp 271 - 309.

SAMET H (1990a), The Design and Analysis of Spatial Data Structures. Addison-Wesley, Reading, MA.

SAMET H (1990b), Applications of Spatial Data Structures: Comp. Graphics, Image Proc., and GIS. Addison-Wesley, Reading, MA.

SAMET H, ROSENFELD A, SHAFFER C A, WEBBER R E (1984b), A geographic information system using quadtrees. In: Pattern Recognition, 17, 6, pp 647 - 656.

SAMET H, SHAFFER A, WEBBER R E (1987), Digitizing the plane with cells of non-uniform size. In: Inform. Proc. Letters, 24, 6, pp 369 - 375.

SAMET H, WEBBER R E (1985), Storing a collection of polygons using quadtrees. In: ACM Transactions on Graphics, 4, 3, pp 182 - 222 (also in: Proc. of Comp. Vision and Pattern Recognition, 83, pp 127 - 132. Washington, DC; and Univ. of Maryland Comp. Science TR-1372).

SAMET H, WEBBER R E (1988b), Hierarchical data structures and algorithms for computer graphics. Part I. Fundamentals. In: IEEE Comp. Graphics and Applications, 8, 3, pp 48 - 68.

SAMET H, WEBBER R E (1988c), Hierarchical data structures and algorithms for computer graphics. Part II. Applications. In: IEEE Comp. Graphics and Applications, 8, 4, pp 59 - 75.

SAMET H, WEBBER R E (1989), A comparison of the space requirements of multi-dimensional quadtree-based file structures. In: Visual Comp., 5, 6, pp 349 - 359.

SAMET H, WEBBER R E (1991), Data structures to support Bézier-based modeling. In: Comp.-Aided Design, 23, 3, pp. 162-176.

SELLIS T, ROUSSOPOULOS N, FALOUTSOS C (1987), The R+-tree: a dynamic index for multi-dimensional objects. Comp. Science, TR-1795. Univ. of Maryland, College Park, MD.

SHAFFER C A, SAMET H (1990a), Set operations for unaligned linear quadtrees. In: Comp. Vision, Graphics, and Image Proc., 50, 1, pp 29 - 49.

SHAFFER C A, SAMET H, NELSON R C (1990b), QUILT: a geographic information system based on quadtrees. In: Int. Journ. of Geogr. Inform. Systems, 4, 2, pp 103 - 131.

SHNEIER M (1981), Calculations of geometric properties using quadtrees. In: Comp. Graphics and Image Proc., 16, 3, pp 296 - 302.

STONEBRAKER M, ROWE L (1986), The design of POSTGRES. In: Proc. of the SIGMOD Conf., pp 340 - 355. Washington, DC.

TAMMINEN M (1981), The EXCELL method for efficient geometric access to data. In: Acta Polytechnica Scandinavica, Mathematics and Comp. Science Series, No. 34. Helsinki.

TANIMOTO S, PAVLIDIS T (1975), A hierarchical data structure for picture processing. In: Comp. Graphics and Image Proc., 4, 2, pp 104 - 119.

VANDERSCHEL D J (1984), Divided leaf octal trees, Research Note. Schlumberger-Doll Research, Ridgefield, CT.

WARNOCK J E (1969), A hidden surface algorithm for computer generated half tone pictures. In: Comp. Science Dep., TR 4-15. Univ. of Utah, Salt Lake City.

WOLF A (1990), The DASDBS GEO-kernel: concepts, experiences, and the second step. In: BUCHMANN A, GÜNTHER O, SMITH T R, WANG Y-F (eds.), Design and Implementation of Large Spatial Databases, Lecture Notes in Comp. Science, No. 409, pp 67 - 88. Springer-Verlag, Berlin.

Acknowledgements

This work was supported by the National Science Foundation under Grant IRI-9017393.

Abstract

An overview is presented of the use of hierarchical spatial data structures such as the quadtree in spatial databases. They are based on the principle of recursive decomposition. The focus is on the representation of data used in geographic information systems. 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.

Zusammenfassung

Der Beitrag bietet einen Überblick über die Verwendung von hierarchischen raumbezogenen Datenstrukturen, wie etwa den Quadtree in raumbezogenen Datenbanken. Sie beruhen auf dem Prinzip der rekursiven Unterteilung. Der Schwerpunkt liegt in der Darstellung von Daten in Geographischen Informationssystemen. Flächenhafte Daten (d.h. 2-dimensionale Gebilde) werden stärker betont als Punkte, Linien und 3-dimensionale Daten.

List of BW-tables (pp 36 - 39)

Page 36

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

Page 37

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.

Figure 5: MX-CIF quadtree. (a) Collection of rectangles and the block decomposition induced by the MX-CIF quadtree. (b) The tree representation of (a).

Page 38

Figure 4: A PR quadtree (b) and the records it represents (a).

Page 39

Figure 6: Binary trees for the y axes passing through (a) the root of the MX-CIF

quadtree in Figure 5 and (b) the NE son of the root of the MX-CIF quadtree in Figure 5.

Figure 7: Example PM_1 quadtree.

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

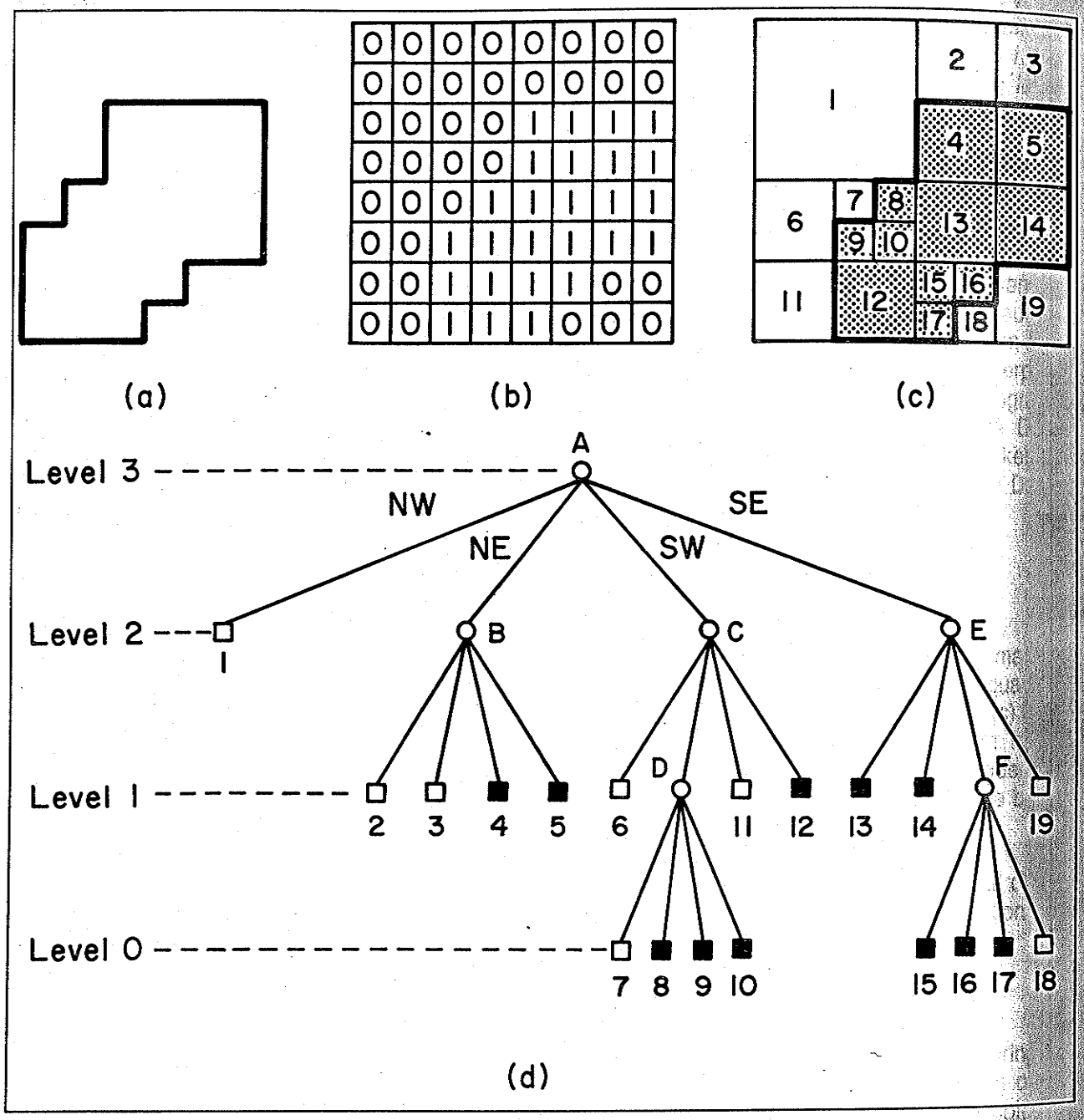

Figure 1

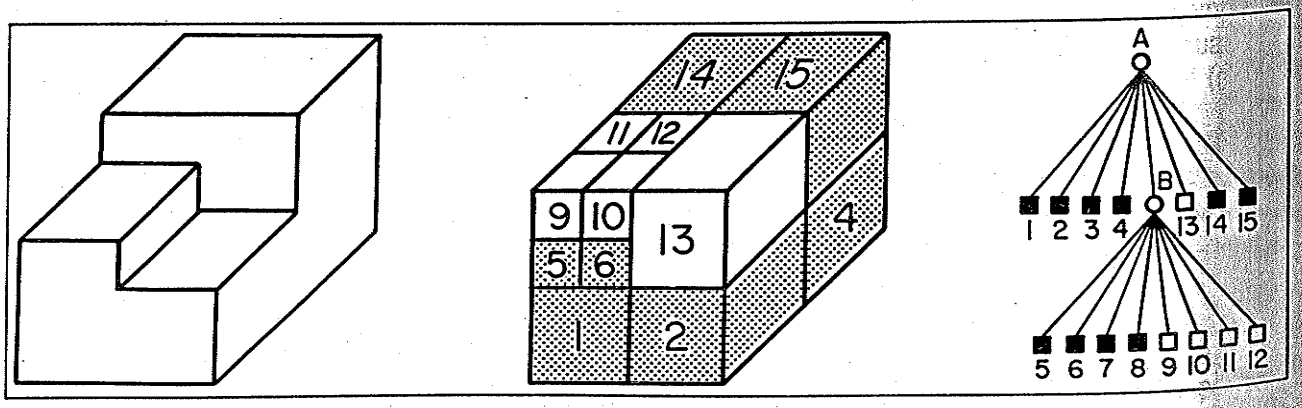

Figure 2

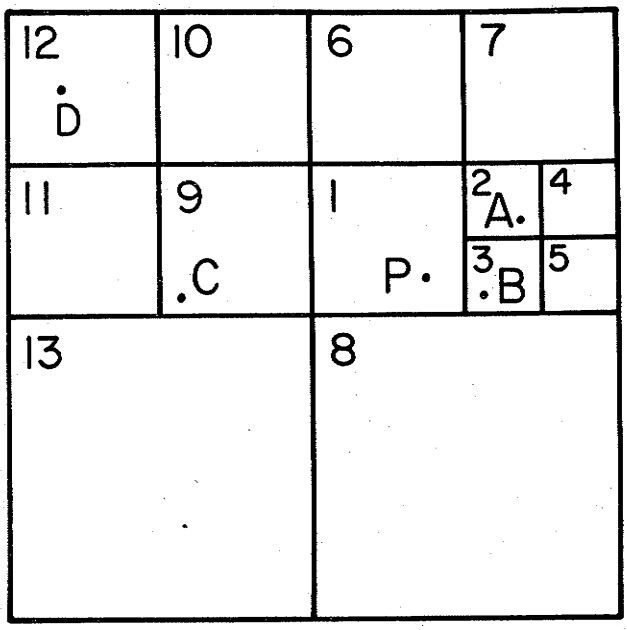

Figure 3

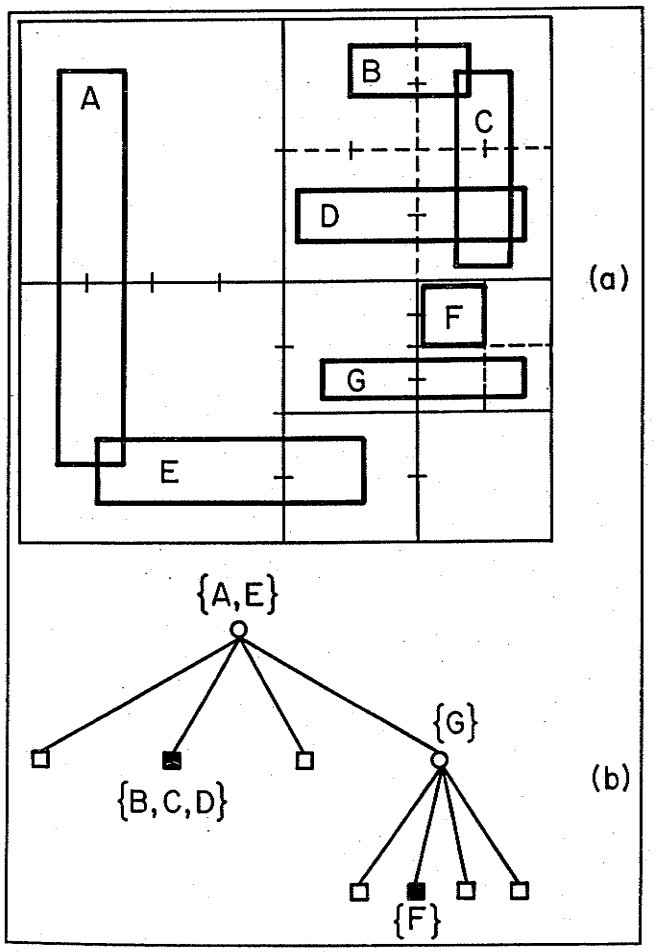

Figure 5

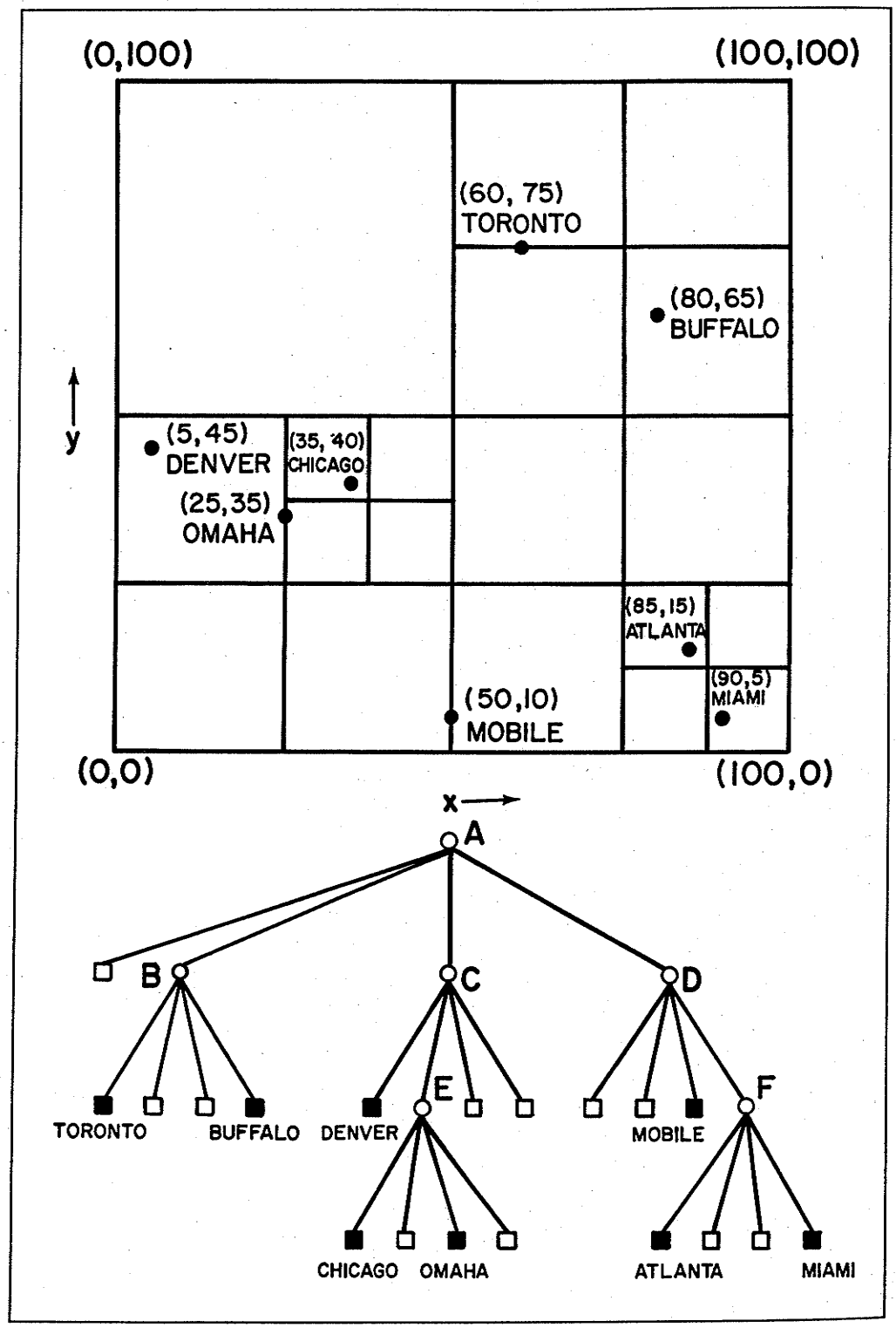
(0,100)
(100,100)
(60, 75)
TORONTO
(80,65)
BUFFALO
y
(5,45)
DENVER
(35, 40)
CHICAGO
(25,35)
OMAHA
(85,15)
ATLANTA
(90,5)
MIAMI
(50,10)
MOBILE
(0,0)
(100,0)
x
A
B
C
D
E
F
TORONTO
BUFFALO
DENVER
MOBILE
CHICAGO
OMAHA
ATLANTA
MIAMI

Figure 4

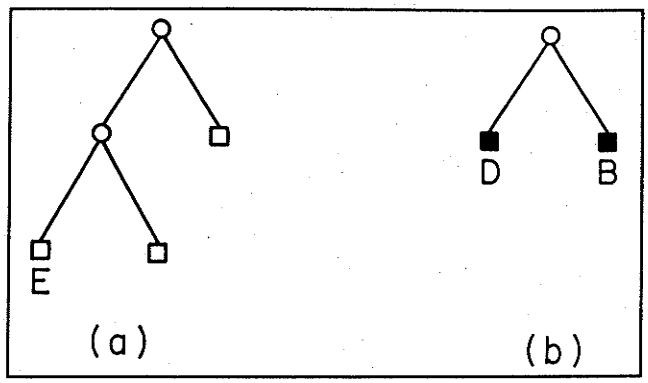

Figure 6

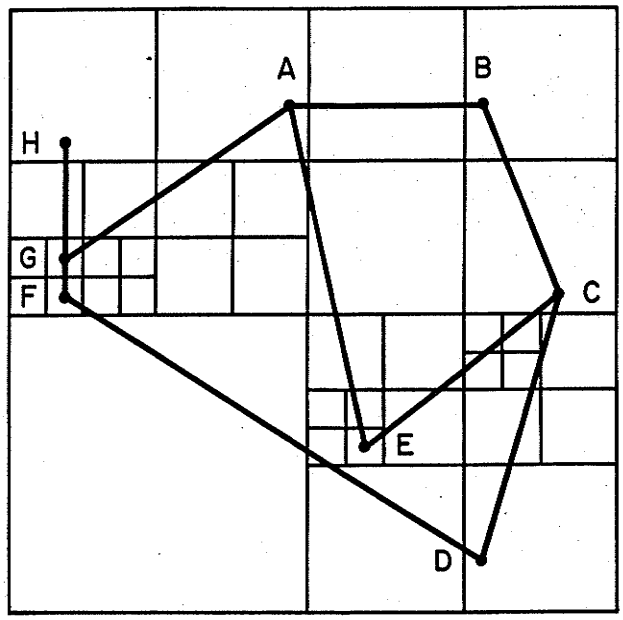

Figure 7

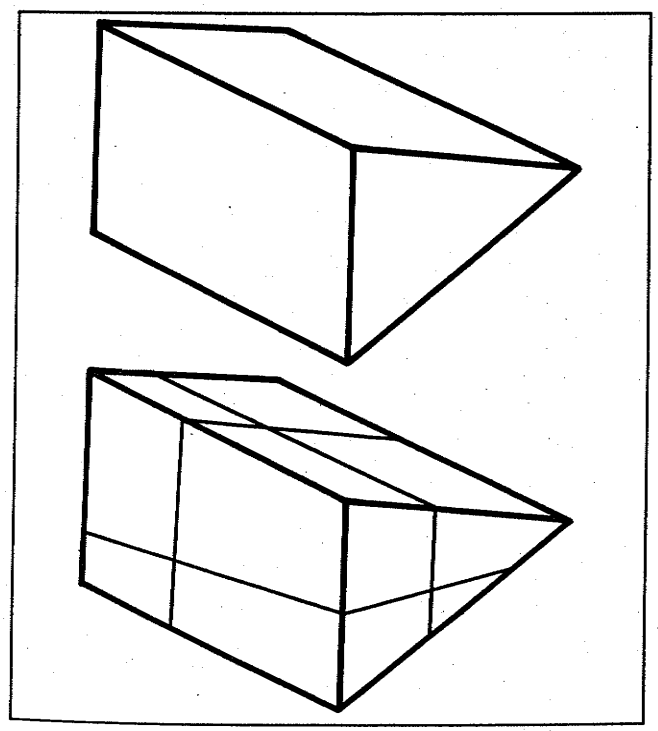

Figure 8