Geol. Jb.	A 104	99–115	6 Fig.	6 Tab.	Hannover 1988

An Empirical Comparison of Vectors, Arrays and Quadtrees for Representing Geographic Data

CLIFFORD A. SHAFFER

Quadtrees, hierarchical data structures, cartography, boundary representations, area representations, vectors, picture arrays, data storage

Abstract: Until recently geographic data was typically stored using edge representations such as a list of points representing a series of vectors which form the boundaries of regions. The pixel array has been a popular alternative to the vector representation, particularly for storing elevation data. Much interest has developed in the use of hierarchical data structures such as the quadtree. A comparison of vector, array, and quadtree data representations is presented. In particular, this paper contains a discussion of the relative strengths and weaknesses of each for various geographic applications. Storage comparisons and empirical timing results for selected geographic database functions are presented.

[Ein empirischer Vergleich der Verwendung von Vektoren, Rastern und Quadtrees zur Speicherung geographischer Daten]

Kurzfassung: Bisher wurden geographische Daten in Form von Punkt-Tabellen, die die flächenumgrenzenden Vektoren darstellten oder, besonders bei Höhendaten, als Pixel-Felder (Raster) abgespeichert. Von steigendem Interesse sind mittlerweile hierarchische Datenstrukturen wie der Quadtree. In diesem Beitrag wird die Speicherung geographischer Daten in Form von Vektoren, Rastern oder Quadtrees miteinander verglichen und die Stärken bzw. Schwächen der jeweiligen Methode anhand verschiedener Anwendungen diskutiert. Speicherplatzbedarf und Zeitverhalten einzelner graphischer Datenbankfunktionen werden erläutert.

Contents

Page
1. Introduction 100
2. Region Representation 101
3. Point and Linear Feature Representation 105
4. Empirical Comparisons 106
4.1 Storage Comparisons 106
4.2 Computation Cost Comparisons 109
5. Conclusions 113
6. References 114

Author's address: Dr. C.A. SHAFFER, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, U.S.A.

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

1. Introduction

Traditionally, polygonal maps in computerized geographic databases have been stored by means of boundary representations. This stems from the fact that it is the natural way for a person to hand-draw a map (PEUQUET 1979). A typical boundary representation will represent a polygon by a list of points which represents a chain of straight line segments forming the boundary of the polygon. Such techniques will be referred to as vector representations. Many commercial cartographic systems, such as ARC/INFO, are based on such methods. Vector representations store maps efficiently, but are inefficient when performing key operations such as determining the value of a point (referred to as the point-in-polygon operation) and polygonal overlay (i.e., intersection of two sets of polygons). Vector representations are also not a good method for storing elevation data (i.e., were each stored point has a z-value associated with it).

In the past several years, area representations have gained acceptance as a body of cartographic algorithms has been developed for their use (PEUQUET 1979). In particular, the array has been applied to cartography, its use driven by the availability of algorithms developed by the image processing community. Arrays are good for storing elevation data, and have been used to represent region maps. However, arrays require much more space to represent region maps than do boundary representations, and are not well suited to representing point and linear feature data. The vector and array structures illustrate the basic differences between boundary and area representations.

Recent interest has developed in hierarchical data structures; in particular we focus on the region quadtree (SAMET 1984b). The quadtree is based on successive subdivision of an image into quadrants. If the image is not homogeneous, it is subdivided into quadrants, subquadrants, ..., until we obtain homogeneous square blocks (see Fig. 1 for an example of quadtree decompostion). The region quadtree is an area representation, with the area representations' advantages over boundary representations with respect to efficient processing of many cartographic operations. An important feature of the quadtree is that it maintains a spatial index for the data, regardless of the data type. It also allows aggregation of array pixels, giving a potential savings in storage and processing time over the array for certain classes of data. The quadtree is good for representing polygonal regions, linear features, and point features, but no better than an array for elevation data.

Many algorithms have recently been developed for efficient manipulation of quadtrees in geographic applications. Early algorithms which initated interest in quadtrees included area and centroid calculation (SHNEIER 1981) and aligned map intersection (polygon overlay) (HUNTER 1978). For these operations, the complexity of the quadtree algorithm is related to the number of nodes in the quadtree representation of the input image(s). A number of conversion algorithms (quadtrees to/from arrays and chaincodes) were produced early on (DYRER 1980, SAMET 1980, 1981, 1984a). Efficient algorithms for perimeter and connected component analysis were then developed based on active border tables (SAMET & TAMMINEN 1985). Recently, a new generation of algorithms allowing efficient array to quadtree conversion, and (more importantly) unaligned map intersection and map transformations have been developed (SHAFFER 1986, SHAFFER & SAMET 1987). It is reasonable to expect further advances in quadtree algorithms in the future.

With the ongoing maturation of quadtree algorithms and the advent of researchers experimenting with the use of quadtrees in geographic information systems (e.g. the

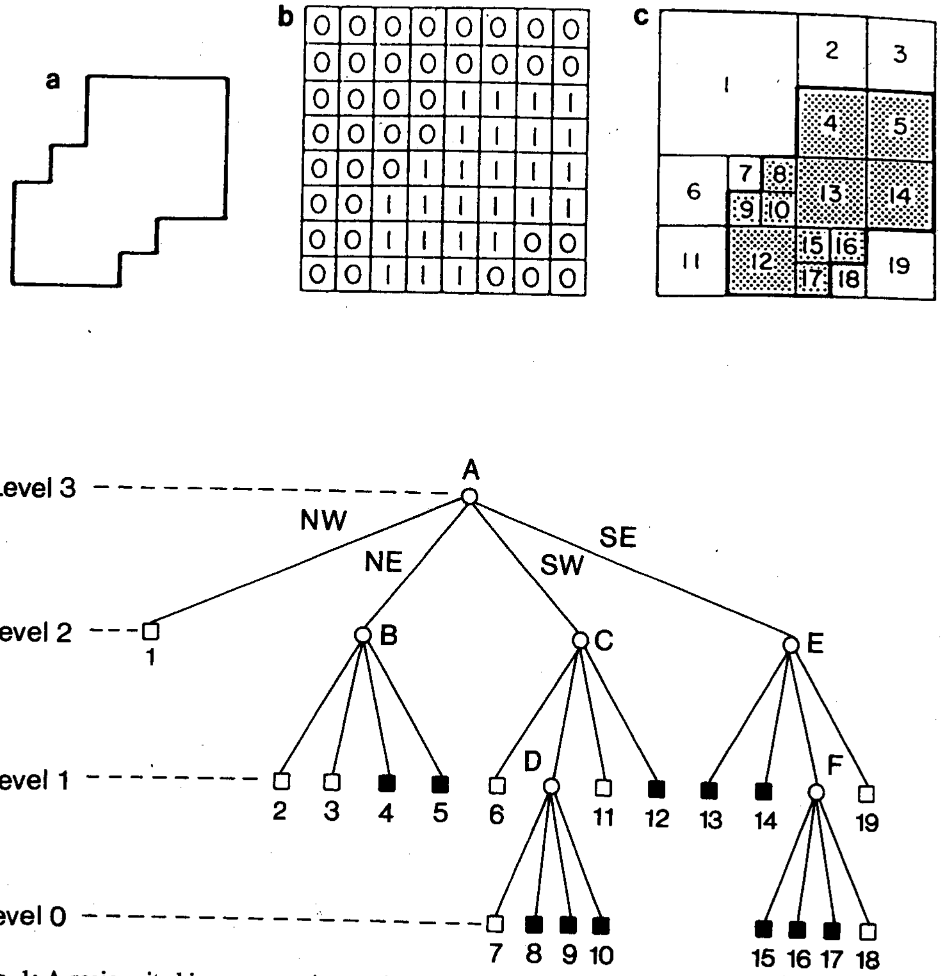

Fig. 1: A region, its binary array, its maximal blocks and the corresponding quadtree.

QUILT system, SAMET et al. 1984, SHAFFER et al. 1985), it seems appropriate at this time to compare the quadtree with the more traditional vector and array representations. This paper discusses the relative merits of these three data structures in the context of a geographic information system. Empirical studies of storage requirements and relative operational efficiency are presented.

2. Region Representation

In this section we discuss the advantages and disadvantages of the vector, array, and quadtree representations for storing region data (sometimes referred to as polygonal data). An example polygonal map is shown in Figure 2.

Vector representations store polygonal maps by means of one or more chains of line segments that represent each polygon's border, as illustrated by Figure 2b. Since only the

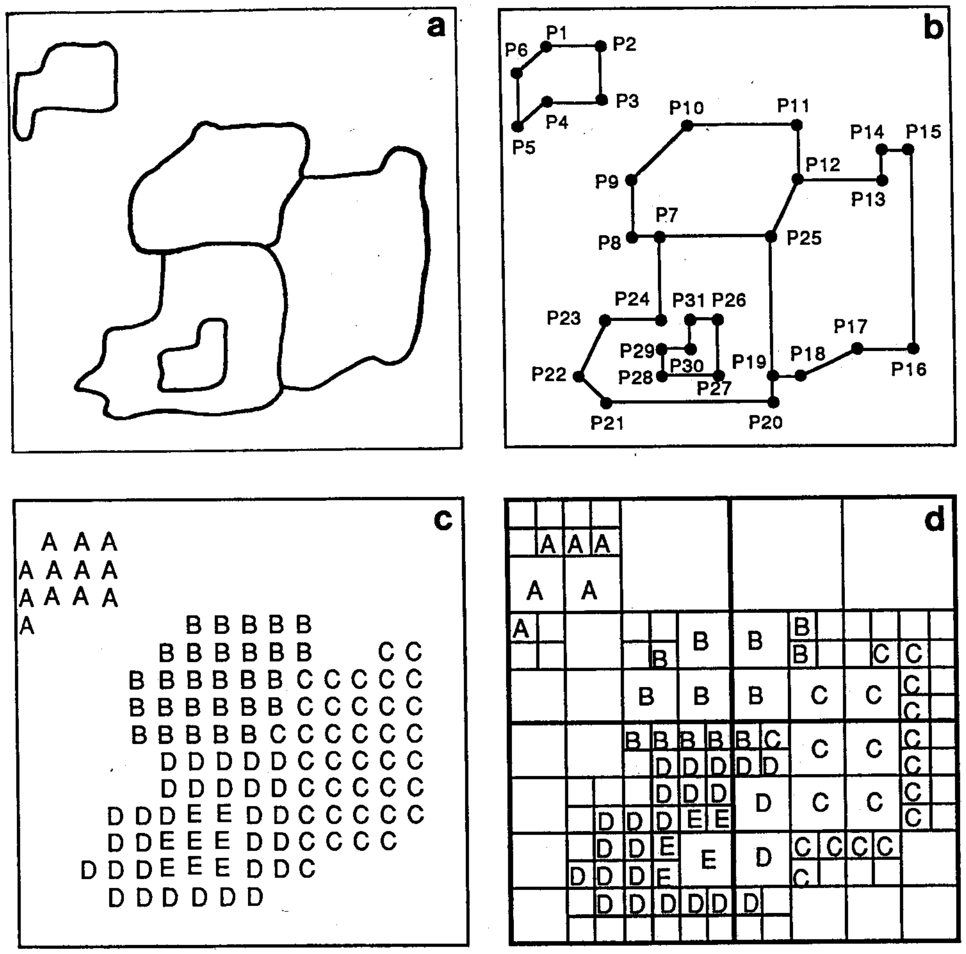

Fig. 2: A polygonal map and its vector, array and quadtree representations.

edges of the polygons are stored (plus some incidental amount of information to identify the polygon), we see that the storage requirements for the vector representation are related to the complexity of the polygon boundaries. In the worst case, this will be proportional to the length of the polygon boundaries (i.e., the perimeter of the polygons). As the database becomes more complicated (i.e., more polygons, or more complex polygons), the storage requirements will grow. When the resolution of the database is increased (requiring a possible breakup of long vectors at the coarse resolution into several vectors at the finer resolution) the storage requirements of the database will increase by a ratio proportional to the "complexity" of the image. A recently developed branch of research, called fractal geometry (MANDELBROT 1983) investigates this phenomenon. In the worst case, we can expect that the increase in storage requirements for the vector representation of a polygonal map will be by a factor of two when the image resolution is doubled.

A significant problem encountered in the use of vector representations is the inherent lack of organization of the data. Storing a chain of vectors forming the polygon boundary gives no indication of the spatial relationship between the individual vectors (other than their connectivity), and between polygons. At best, a conceptual relationship may explicitly be stored, such as adjacency and topography information. Operations such as point-in-polygon and polygonal overlay (intersections between two sets of polygons, as illustrated in Figure 3) are thus very inefficient to perform, and difficult to program. Point-in-polygon algorithms are fairly well understood, but polygonal overlay algorithms for vector representations are still a research topic. Some attempts have been made to combine vectors with some form of raster-based spatial ordering (e.g., the vaster of PEUQUET 1983), but no extensive work has appeared for such representations.

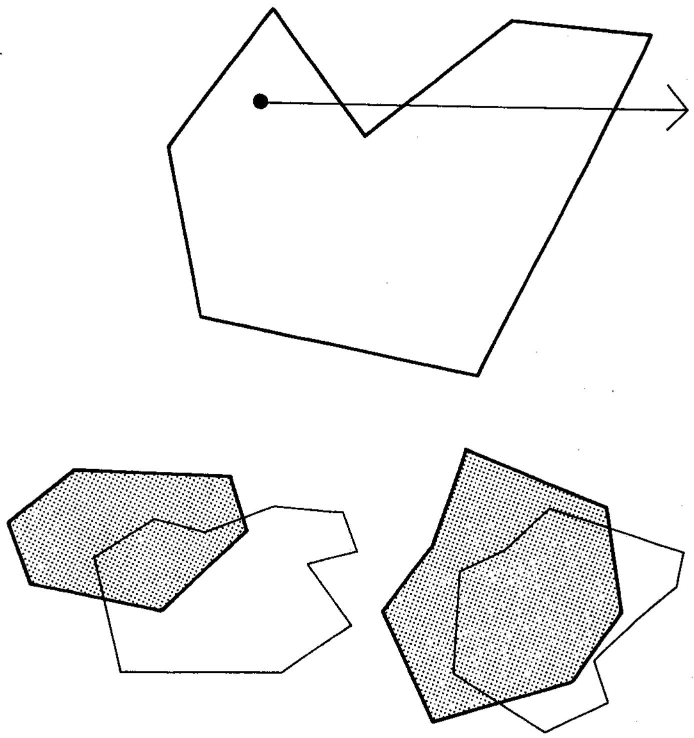
Fig. 3: Examples of the point-in-polygon and polygonal overlay operations.

Figure 2c illustrates the use of an array representation for a region map. The array has straight-forward storage requirements, based solely on the area of the region represented, and not on its complexity. For a given resolution, a map of R rows and C columns, with pixels of size P bytes requires RxCxP bytes of storage. Thus, as image resolution doubles in both the horizontal and vertical directions, storage requirements increase in the array by a factor of 4.

Functions such as area, and polygonal overlay are computed very simply in the array, requiring a single examination of every pixel. Many other cartographic algorithms have been developed. Those algorithms (such as area computation) which can be performed on vector representations on O(Number of vectors) typically require in the array O(Number of pixels). For such algorithms, the vector representation should have the advantage over the array. However, the importance of polygonal overlay and point-in-polygon operations seems to outweigh the importance of those operations where vectors perform efficiently. Original interest in the array was due to its compatibility with raster output devices (such as line printers). Since many modern graphic display devices can display arbitrary vectors as primitive objects, this particular advantage for arrays has been reduced, at least for display purposes.

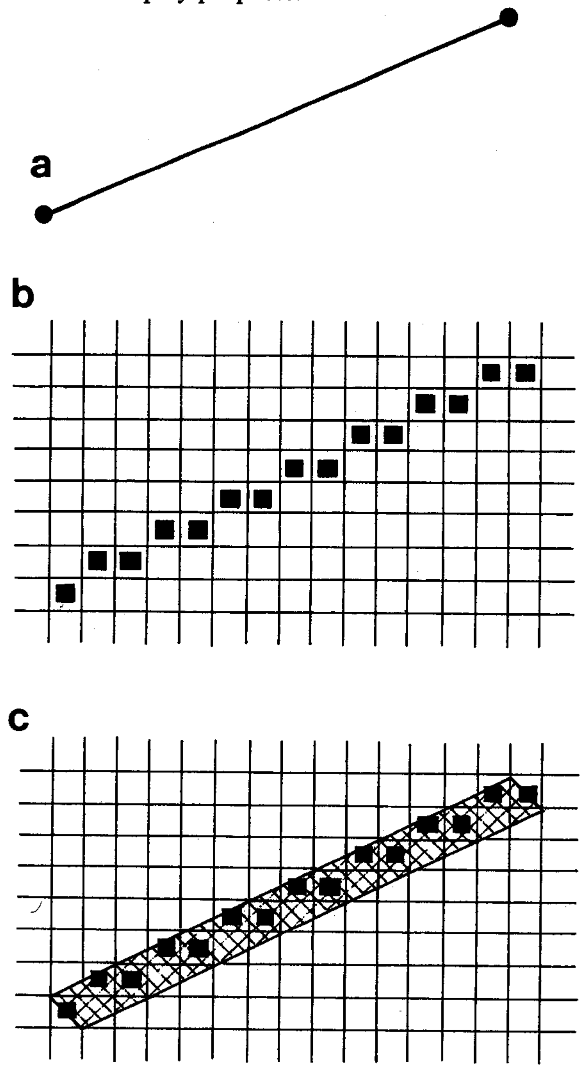

Fig. 4: The true accuracy for a vector whose endpoints' resolution is the same as the pixel resolution used by an array.

When comparing the digitization of a vector in Figure 4b with the original vector in Figure 4a, the digitization shows a "staircase" effect. It is possible to be misled by the precision of the vector endpoints as drawn into the belief that vector representations are inherently more accurate than area representations, where this staircase is not as obvious (note that there is some blockiness, however, in the polygonal representation of Figure 2b as compared to Figure 2a). In particular, a "loss of accuracy" appears when converting from a vector representation to an array representation. A shown by Figure 4c, this is entirely an artifact of false precision ascribed to the vector endpoints and line width in the illustration. If the vector endpoints have the same precision as an array pixel, then the precision of the vector is actually as indicated in Figure 4c. The vector representation has no greater accuracy than the array when both are stored with the same precision.

Like the vector representation, the region quadtree's storage requirements for a given map are related to the complexy of the map. HUNTER (1978) has demonstrated that the number of nodes in the quadtree representation of an image is related to the length of the perimeter of the polygons in that image. The fact that the quadtree's complexity is related to the perimeter is significant. It means that as the resolution is increased by a factor of two, the quadtree will require approximately twice as much storage (as would the vector representation), whereas the array's storage requirements will grow by a factor of four. Section 4.1 below provides empirical evidence to support this claim.

When comparing the quadtree and array representations for a map, the complexity of that map is clearly the key factor in determining which will be more efficient. Since a given quadtree node must store some form of position and size information, it takes up more space than a given array pixel in the case where both the quadtree node and the array pixel use the same amount of storage to represent the pixel value. Second, we can expect more time to be required to process a given quadtree node than a given array pixel. This is because a quadtree node requires more time to locate, more time to decode, and has a more complicated relationship with neighboring nodes.

Since a given quadtree node takes more space and processing time than an array pixel, it is interesting to determine the trade-off point in terms of pixels/node above which the quadtree is more efficient than the array. Clearly, if every pixel of the image has the same value, and thus the image can be represented by a single quadtree node, any reasonable function will run much faster on the quadtree for all but the smallest of images. Conversely, if every pixel has a different value, and a complete quadtree is required for its representation, then the quadtree is inappropriate. The key question for the practicality of a geographic information system based on quadtrees is where this trade-off point falls with respect to the complexity of the "typical" image to be represented in the system. This depends in part on what operations the system will perform, since different operations have different trade-off points.

3. Point and Linear Feature Representation

A general purpose geographic information system will not process only region data. Point and linear feature data plays an important role as well, each data type associated with its own operations. In this section we will briefly consider the ability of vectors, arrays, and quadtrees to store these data types.

The vector is naturally very efficient for storing linear feature data, since linear features are most commonly approximated by chains of line segments. When the linear features are not meant to form polygons, the difficult operations of point-in-polygon and polygonal overlay need not be performed. However, as with polygonal data, there is no spatial organization to the collection of vectors. Operations such as finding all linear features in a region is nearly as difficult to perform with vector representations as computing polygonal overlays. Points can be represented as degenerate line segments (i.e., simply store a list of points). Once again, there is no spatial organization.

The arrays is a poor representation for point and linear feature data. Point data can be stored in an array by associating each point with a particular pixel. This leads to much wasted space with most pixels being empty. The array does, however, maintain the spatial relationship between the data points. Efficient array representations for linear feature data are hard to produce. A simple representation would make BLACK those array pixels which contain a line segment. However, this indicates only a weak relationship among the set of pixels making up a line segment (i.e., adjacent BLACK pixels may or may not be part of the same segment). The most difficult problem is to distinguish between several line segments which intersect a given pixel. At best, the array stores an approximation of the line data in a very inefficient manner.

Several methods have been proposed to use the quadtree as a spatial organization for a collection of points or line segments (SAMET et al. 1984, SAMET & WEBBER 1985, NELSON & SAMET 1986). Basically, a list of line segments is maintained, and a quadtree decomposition storing pointers into the segment list is used for the spatial organization. Such a structure allows for efficient manipulation of the line data without any loss of accuracy. The cost of such a spatial organization is related to the size of the segment list. The PR quadtree (ORENSTEIN 1982, SAMET 1984b) is a similar representation designed to store point data.

4. Empirical Comparisons

In this section we present empirical results on the storage requirements and operational efficiency of the vector, array and quadtree representations. Section 4.1 discusses relative storage requirements for the three representations. Section 4.2 compares computational costs for selected operations on the array and quadtree data structures.

4.1 Storage Comparisons

The test data consisted of two sets. The first set is a series of images each of the same approximate size (three images of 400 x 450 pixels and five images of 512 x 512 pixels) but with varying complexity. These images included the floodplain, landuse, and topography maps which make up our standard test set (Figures 5a to 5c). Three images derived from these were also used; the center region from the floodplain map and the ACC and VV landuse classes from the landuse map. Finally, the pebble and stone images were also included (Figures 6a and 6b).

The pebble and stone images are quite complicated, as can be seen from the number of nodes required for their quadtree representations (see Table 1). The landuse class, topography, and floodplain maps are considerably simpler, and finally, the center region, ACC class, and VV class maps are simpler yet.

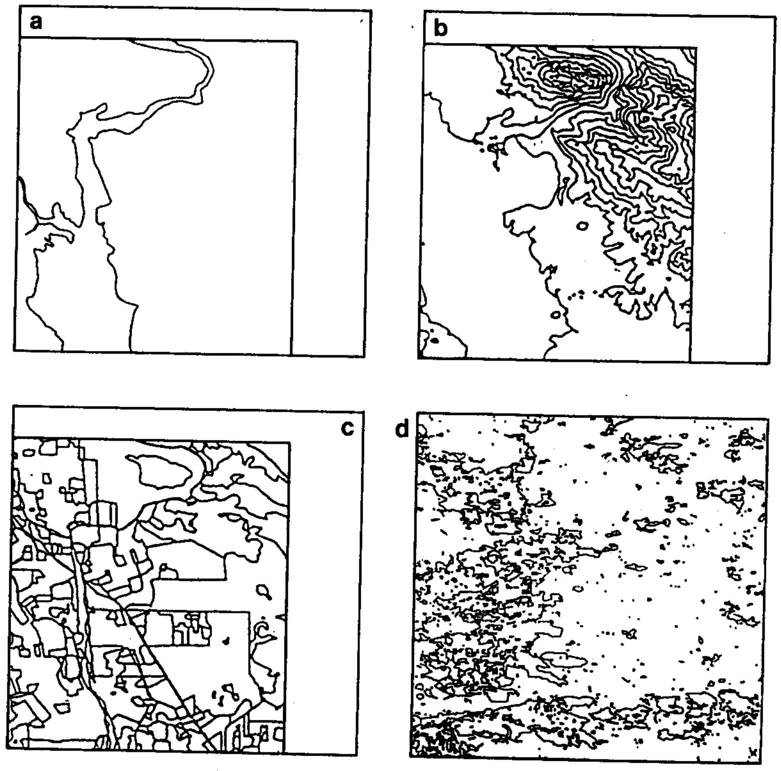

Fig. 5: Four example maps.
a) The Floodplain map.
b) The Topography map.
c) The Landuse map.
d) DMA data.

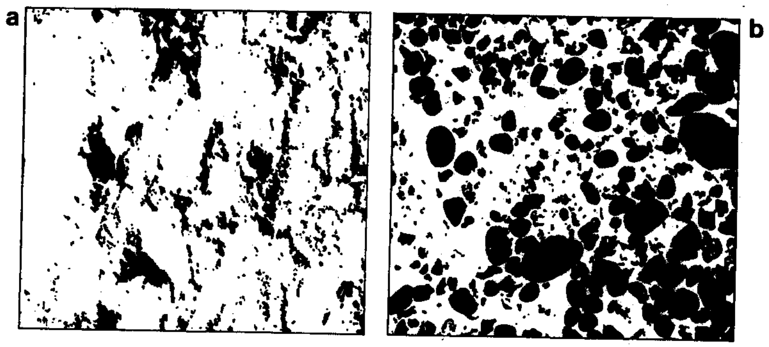

Fig. 6: Two example images.

The second test set is derived from data supplied by the Defense Mapping Agency. This test data does not seem to correspond to any known geographical location or objects, but is presumed to represent "reasonable" data. The original data was in vector format, to a resolution of 36,000 by 36,000 units. For this experiment, the data was rescaled to a variety of digitizations, and converted to our array representation. The vector to array conversion process removed those line segments whose vertices both fell at the same location. This gives the effect, as the resolution is increased, of having small features reappear to fill empty space.
This mirrors the natural practice of filling in a given map with smaller scale features as the resolution increases. The image, at a resolution of 500 x 500 pixels, is shown in Figure 5d. As can be seen in Table 1, the image is extremely complex, nearly as complex as the pebble image. The data has been digitized at seven resolutions, from 63 x 63 pixels up to 4000 x 4000 pixels. They are referred to as P63, P125, etc., where the number indicates the image size.

For the purpose of empirical experimentation, a storage implementation for each of the three data structures was selected. Arrays are stored as disk files containing a contiguous block of pixels in row major order. Differing applications have widely differing storage requirements for pixel elements, ranging anywhere from one bit/pixel for binary images to 199 bits/pixel used by the DMA Prototype Tactical Terrain Analysis Database. Throughout this test, the array pixels were stored using 32 bits; the quadtree nodes also store 32 bit value fields. The quadtree implementation used is the QUILT system (SAMET et al. 1984, SHAFFER et al. 1985), which utilizes disk-based files containing a linear quadtree (GARGANTINI 1982). The linear quadtree recasts the leaf nodes as a sorted list; this list is organized by a B-tree (COMER 1979). No vector-based geographic software was available, and time did not permit an implementation to be developed specifically for this project. However, a raster to vector (polygonal approximation) algorithm was available to allow computation of vector storage requirements. For storage calculations, each vertex point is assumed to require 4 bytes of storage (for the x and y values), and each polygon was arbitrarily assumed to require an additional 8 bytes.

Table 1. Storage requirements

	Vector			Array		Quadtree		
Image	Points	Polys	Bytes	Pixels	Bytes	Nodes	Bytes	Pixels/ Node
Flood	792	3	3,192	180,000	720,000	5,266	47,104	49.7
Land	4,890	212	21,256	180,000	720,000	28,447	249,856	9.2
Top	5,086	94	21,095	180,000	720,000	24,859	217,088	10.5
Stone	–	–	–	262,144	1,048,576	31,969	279,552	8.1
Pebble	–	–	–	262,144	1,048,576	44,950	392,192	5.8
ACC	405	1	1,628	262,144	1,048,576	4,687	41,984	55.9
Center	13	1	60	262,144	1,048,576	3,280	29,696	79.9
VV	241	19	1,116	262,144	1,048,576	103	1,024	2545.0
P63	5,177	606	25,556	3,969	15,876	1,162	13,312	3.5
P125	9,183	814	43,244	15,625	62,500	4,351	43,008	3.7
P250	16,253	923	72,396	62,500	250,000	14,770	138,240	4.4
P500	22,340	947	96,936	250,000	1,000,000	42,931	387,072	6.1
P1000	26,255	961	112,708	1,000,000	4,000,000	106,432	948,224	9.8
P2000	27,521	962	117,780	4,000,000	16,000,000	235,615	2,089,984	17.8
P4000	27,865	962	119,156	16,000,000	64,000,000	501,127	4,350,976	33.4

As expected, the vector representation was the most efficient of the data structures for most test images (surprisingly, both the quadtree and the array were more efficient for the image P63). For all test images, the quadtree requires less space than the array. One of the most important observations to be noted here is the effect of increased image resolution. As can be seen in the progression from P63 to P4000, when the image doubles in resolution, the space requirements for the array increase by a factor of four. For the quadtree, beyond the smaller images, the increase approaches a factor of two. Thus, as pictures of a given complexity get larger, the space efficiency of the quadtree with respect to the array becomes more dramatic. The relative space efficiency of the vector representation is also dramatic.

4.2 Computation Cost Comparisons

In order to compare the efficiency of the quadtree representation to a standard pixel array, we have implemented a set of six functions using both the array and the quadtree as the underlying data structure for image representations. The functions which we tested were area, perimeter, point in polygon (i.e., random access), windowing, WITHIN, and single polygon value changing. These six functions were then executed on the test images, with the results presented in Tables 2 through 5 below. All times in these tables are measured in CPU seconds. The tests were run on a SUN Microsystems 2/120 workstation using a 200 megabyte Eagle disk drive. The algorithms for the six functions are briefly described below.

The area function for both the array and the quadtree simply visits each pixel or node. If the value is not WHITE, then the size of data element is added to the area counter. The array is processed in raster-scan order; the quadtree in a pre-order traversal. The test results are shown in Table 2. Other operations that can be computed in the same way include bounding rectangles, area of a subset of polygons whose membership can be deter-

Table 2. Execution times for area, perimeter, and access

	Area		Perimeter		Access		
Image	Array Time	Quadtree Time	Array Time	Quadtree Time	Array Time	Quadtree Time	Pixels/ Node
Flood	5.4	1.0	8.5	3.0	9.3	5.3	49.7
Land	5.5	5.2	8.6	14.3	9.6	11.2	9.2
Top	5.4	4.4	8.7	12.5	9.7	9.7	10.5
Stone	7.3	4.9	12.1	16.0	10.0	10.9	8.1
Pebble	7.3	7.0	12.2	22.5	10.4	14.0	5.8
Center	7.4	0.8	12.2	2.6	10.1	4.0	55.9
ACC	7.4	0.6	12.0	2.8	10.7	3.3	79.9
VV	7.3	0.2	12.1	0.2	10.9	1.7	2545.0
P63	0.2	0.3	0.3	0.7	4.1	2.5	3.5
P125	0.6	0.8	0.9	2.4	4.4	6.5	3.7
P250	2.2	2.3	3.3	7.4	5.4	9.4	4.4
P500	7.3	6.8	11.7	21.2	9.9	12.3	6.1
P1000	26.7	18.2	44.5	58.5	11.0	19.9	9.8
P2000	100.2	37.5	171.5	117.7	12.1	20.9	17.8
P4000	377.5	80.0	666.0	252.3	13.0	24.4	33.4

mined by node value, the size of a population where the value of a node indicates density of the population, and centroid of a polygon. The trade-off point (beyond which the quadtree is more efficient than the array) for the area operation seems to be at about 5 pixels/node. These results indicate that for most applications, quadtrees are more efficient than arrays for simple node visit functions.

The perimeter function for the array processes each row in conjunction with the previous row. For each pixel, the value of the neighbor to its left is compared to the value of the current pixel. If they differ, the perimeter count is incremented. The neighbor in the previous row is likewise examined. For the quadtree, the nodes are visited in a preorder traversal. An active border table is maintained which describes the border of that portion of the tree visited so far (SAMET & TAMMINEN 1985). Each node is compared against its western and northern neighbors as represented in the table; where they differ, the length of the edge is added to the perimeter counter. This perimeter algorithm is similar to that used when labeling connected components; it is the standard quadtree algorithm that we use for any function calculable by comparing nodes to previously processed neighbors. As can be seen from analyzing Table 2, the trade-off point seems to be at about 12-15 pixels/node. Thus, for the 512 x 512 pixel sized images in our test set, the more complicated images are more efficient with arrays, the less complicated with quadtrees. For larger images, the quadtree has a significant advantage.

The third function compares the ability of the two representations to compute the value of a given point (i.e., random access). 2000 random points are generated for each image, with both the array and the corresponding quadtree searching for the same 2000 points. For each point the corresponding node or pixel is retrieved. While the array is more efficient than the quadtree, this test shows mainly that both the quadtree and the array are quite efficient then performing random access. Either is likely to be much more efficient than a vector based representation at determining the value of a given point. Moreover, the size of the image has only a small effect on the execution time.

The fourth function tested takes a window from an image; i.e., it extracts a rectangular subsection. We first compared the array and the quadtree for the special case in which the windows are not rotated with respect to the image x and y axes. The first window was half of the width of the image, with origin at (0,0); the second was half of the width of the image with origin at (1,1); the third and fourth were each a quarter of the width of the image with origins at (0,0) and (1,1) respectively. This allows for two tests each for the best position for quadtree windowing (origin at (0,0)) and then the worst position (origin at (1,1)). Windowing for the array where there is no rotation of the window is extremely efficient; those rows within the window are read in, and the appropriate subportions are then re-written. Windowing for quadtrees is much more complicated; the algorithm is presented in SAMET et al. (1985). The windowing function is ideally suited to the array. On the other hand, it is unsuited to the quadtree since the window border may not mesh with the quadtree nodes, thus causing much splitting and remerging of nodes. From Table 3, we see that the array is superior for all but the most homogeneous images (although such cases do frequently occur, particularly for smaller windows).

The next test compared the ability of the array and the quadtree to extract rotated windows. This function is calculated in the quadtree in a manner almost identical to that used for unrotated windowing. However, the array algorithm needed exteme modification. Our algorithm makes several passes over the input array. During each pass, as

Table 3. **Empirical results for windowing**

Image	Origin (0,0)				Origin (1,1)			
	One half		One quarter		One half		One quarter	
	Array Time	Quadtree Time	Array Time	Quadtree Time	Array Time	Quadtree Time	Array Time	Quadtree Time
Flood	4.1	7.4	2.7	1.3	4.0	13.1	2.4	2.8
Land	4.1	34.8	2.4	6.9	4.0	48.6	2.5	10.6
Top	4.0	8.3	2.5	2.0	4.1	15.3	2.3	10.6
Stone	4.3	20.7	2.7	4.0	4.6	31.1	2.5	6.4
Pebble	4.2	30.6	2.5	8.4	4.4	43.8	2.4	11.9
Center	4.4	7.1	2.3	1.3	4.4	13.1	2.6	2.8
ACC	4.3	6.9	2.5	0.9	4.4	13.2	2.5	1.9
VV	4.2	0.3	2.5	0.3	4.5	0.8	2.6	0.3
P63	0.4	1.8	0.2	0.7	0.4	1.8	0.3	0.7
P125	0.8	5.0	0.4	1.6	0.8	6.0	0.4	1.9
P250	2.1	16.1	0.9	5.0	2.1	20.2	0.9	6.0
P500	4.3	51.2	2.5	13.7	4.2	20.2	0.9	6.0
P1000	13.0	155.1	5.4	34.5	12.0	200.8	5.2	47.0
P2000	43.4	284.5	17.2	93.0	42.8	528.0	17.5	136.2
P4000	153.6	807.2	57.1	225.4	156.9	1260.2	56.4	351.3

many output rows as can be stored in memory are computed. At the end of each pass, these rows are output; the next several rows are then processed in turn. For test purposes, the output row buffer was as large as the memory buffer allocated to the quadtree memory management system (12K bytes). Naturally, the actual number of input image rows read during each pass is dependent on the rotation angle and size of the output image window. Only those input rows actually covering pixels of the output window need processed. Thus, the best rotation angle is 0 degrees, while the worst is 90 or 270 degrees. For our tests, we took two windows with origin at (WIDTH/2, WIDTH/4) where WIDTH is the width of the image. Each window was rotated by 45 degree with respect to the input image axes. For rotated windows, the window's origin does not significantly affect the quadtree's efficiency, so only one window of each size was tested. The results of this test are shown in Table 4. Timing results for the 4000 x 4000 pixel image are unavailable since this algorithm was tested at a latter time than the others. Sufficient disk space was no longer available to store the array! The quadtree performed better than the array in nearly all test cases. The trade-off point for the quadtree is at about 5 pixels/node. Note that scale change and projection transformation algorithms are quite similar to that used for rotated windowing.

The fifth function compared is termed the WITHIN function. This function expands the non-WHITE polygons of a given image by a specified number of pixels. For the array, this function was computed by use of a standard two-pass chessboard distance transform in the WHITE pixels, with the second pass setting each pixel originally WHITE to BLACK if its distance transform value is less than the radius value. For each test image, a radius of 1/64th of the image width was used. The quadtree computes this function by calculating the distance transform for all WHITE pixels, and converting them to BLACK as appropriate. Our WITHIN algorithm is an example of a case where the array representation needs only to examine neighbor values for each pixel, while the

Table 4. Empirical results for windows rotated 45 degrees

Image	One half		One quarter	
	Array Time	Quadtree Time	Array Time	Quadtree Time
Flood	47.3	8.8	13.1	1.8
Land	47.8	44.6	13.1	9.9
Top	48.7	39.2	13.6	14.5
Stone	76.9	45.6	20.4	9.7
Pebble	58.5	51.9	15.7	14.0
Center	61.0	6.8	15.9	1.8
ACC	58.9	16.3	15.8	2.9
VV	59.7	2.5	15.6	1.5
P63	0.6	1.6	0.2	0.8
P125	1.7	4.1	0.7	1.5
P250	8.9	13.4	2.6	3.4
P500	59.9	43.0	16.0	10.9
P1000	651.0	128.0	164.4	31.5
P2000	13765.7	344.3	3434.4	81.8
P4000	–	–	–	–

quadtree must also do splitting and merging. Thus, the array is superior for images with less than about 40 pixels/node at the 512 x 512 image size, as indicated in Table 5. Since the quadtree algorithm seems to be of cost O(n^2) where n is the number of WHITE nodes, larger images (with an absolute increase in the number of WHITE nodes) does not lead to a relative improvement in the quadtree over the array. This is the only function that we examined which has this property. It should be noted, however, that in practice the WITHIN function would likely be executed on images such as the ACC or center maps rather than the landuse map. This is so because the user will normally wish to derive those features 'within' a given distance of specified features - i.e., a polygon or landuse class. Such images are, of course, less complex.

The final function tested changes the color of a particular polygon within an image. For each test image, the function was performed with input point (WIDTH/2, WIDTH/2) for an image of width WIDTH. The function was then computed with input point (WIDTH/4, WIDTH/4). Both the array and the quadtree utilize an algorithm similar to the standard two-pass connected components algorithm. The first pass for both the quadtree and the array is nearly identical to the perimeter function, with the exception that each node or pixel must then be rewritten with the value indicating the equivalence class of that node or pixel. The second pass visits each node or pixel, restoring its proper color value, and outputs the image. The size of the polygon being changed has some impact on the amount of time required. Quadtrees are particularly efficient when the polygon is small. In general, from the Table 6 we find that the trade-off point is about 10 or 11 pixels/node, making the change function slightly better for quadtrees for a "typical" image. This function is also a necessary step in creating a map containing a subset of polygons from an input map.

Table 5. Empirical results for the WITHIN function

Image	Array Time	Quadtree Time	Pixels/ Node
Flood	31.3	20.2	49.7
Land	32.3	105.0	9.2
Top	33.2	105.9	10.5
Stone	97.0	472.0	8.1
Pebble	78.6	454.9	5.8
Center	100.8	71.1	55.9
ACC	107.1	60.4	79.0
VV	109.2	3.1	2545.0
P63	2.2	10.1	3.5
P125	7.2	44.2	3.7
P250	23.5	140.6	4.4
P500	86.5	499.6	6.1
P1000	345.7	1524.2	9.8
P2000	1354.9	5197.3	17.8
P4000	4494.5	18092.2	33.4

Table 6. Empirical results for the change function

Image	Change at (1/2, 1/2)		Change at (1/4, 1/4)		Pixels/ Node
	Array Time	Quadtree Time	Array Time	Quadtree Time	
Flood	30.0	6.0	23.2	9.2	49.7
Land	22.8	23.0	20.7	19.5	9.2
Top	20.9	20.4	25.8	20.3	10.5
Stone	48.8	67.2	48.4	65.2	8.1
Pebble	37.5	87.8	41.7	83.0	5.8
Center	48.6	9.3	30.1	8.6	55.9
ACC	50.8	0.5	50.9	0.5	2545.0
P63	1.4	2.6	1.3	2.6	3.5
P125	4.1	9.0	2.7	8.7	3.7
P250	12.9	28.1	8.6	28.0	4.4
P500	45.1	78.3	45.1	78.0	6.1
P1000	165.8	187.1	166.2	186.2	9.8
P2000	636.9	408.3	636.7	404.1	17.8
P4000	2455.6	854.5	2462.5	848.3	33.4

5. Conclusions

We have compared the vector, array, and quadtree representations as to their usefulness in a geographic information system. We have found that the vector representation is the most efficient representation in terms of storage, but very inefficient for execution of many important functions. This result is, of course, well known. What this paper provides is a discussion of the relative merits of the quadtree data structure with respect to the better known vector and array representations. Our experiments show that the quadtree is more efficient in its storage requirements than the array, with an even greater advantage to the quadtree as the image resolution increases.

The quadtree can perfom some, but not all, of the common functions efficiently in terms of execution time as compared to the array. The current state of the art in quadtree database implementations allows the quadtree to compute simple tree traversal functions (such as area and rotated window computation) more quickly than the comparable algorithms in the array for most images of interest. Scale and projection change functions would likely utilize algorithms nearly identical to our rotation algorithm. The perimeter computation and polygon value changing functions are computed by our implementations in about the same time for both the quadtree and the array if we assume that the "typical" image requires about 10 pixels/node (as do the landuse and topography maps). Both representations are quite efficient at computing point-in-polygon values, unlike a vector based representation. The array has an advantage for the WITHIN function, and a tremendous advantage when performing the special case of un-rotated windowing.

When selecting a data structure, one must examine the desired application to determine which of two representations is more appropriate. For geographic systems, the quadtree seems generally to perform at least as well as the array for our particular implementations. There are two major factors which argue in favor of the quadtree at this stage. First, our implementation is only a prototype. It should be possible to achieve greater efficiency through further optimizations. Array implementations for many functions are quite simple. While this is an advantage for developing programs for short term use, for production systems the developers can spend the time necessary to produce greater efficiency. We feel that the particular quadtree implementation tested has a potential for greater efficiency, while our array implementation does not.

The second major advantage of the quadtree is the fact that it is more suitable for storing point and line data than the array. Quadtree representations can use the same underlying representation and disk accessing/memory management code for all three types of data. While a (storage inefficient) array-based point representation could be developed, it would be particularly hard to develop a line representation based on arrays. One of the major difficulties is the fact that an unbounded number of line segments can pass through a given array pixel.

Finally, the quadtree becomes more efficient with respect to the array as the image size increases, Thus, for databases with larger map objects, the quadtree may prove to be significantly more efficient than the array.

Acknowledgements: I would like to thank KIKUO FUJIMURA for his help in writing the array comparison algorithms and DAVID OSKARD for writing the DMA/SLF data conversion routines.

6. References

COMER, D. (1979): The Ubiquitous B-tree. - ACM Computing Surveys, **11**, 2: 121-137; New York.

DYER, C.R., ROSENFELD, A. & SAMET, H. (1980): Region representation: boundary codes from quadtrees. - Comm. ACM, **25**, 12: 171-179; New York.

GARGANTINI, I. (1982): An effective way to represent quadtrees. - Comm. ACM, **25**, 12: 905-910; New York.

HUNTER, G.M. (1978): Efficient computation and data structures for graphics. - Ph.D. Diss., Dept. of Electrical Engineering and Computer Science, Princeton Univ.; Princeton.

MANDELBROT, B. (1983): The Fractal Geometry of Nature. - New York (Freeman).

NELSON, R.C. & SAMET, H. (1986): A consistent hierarchical representation for vector data. - Computer Graphics, **20**, 4: 197-206; New York.

ORENSTEIN, J.A. (1982): Multidimensional tries used for associative searching. - Information Processing Letters, **14**, 4: 150-157; Washington.

PEUQUET, D.J. (1979): Raster processing: an alternative approach to automated cartographic data handling. - The American Cartographer, **6**, 2: 129-139; Washington.

- (1983): A hybrid structure for the storage and manipulation of very large spatial data sets. - Computer Vision, Graphics and Image Processing, **24**, 1: 14-27; New York.

SAMET, H. (1980): Region representation: Quadtrees from boundary codes. - Comm. ACM, **25**, 12: 163-170; New York.

- (1981): An algorithm for converting rasters to quadtrees. - IEEE Transactions on Pattern Analysis and Machine Intelligence, **3**, 1: 93-95; New York.

- (1984a): Algorithms for the conversion of quadtrees to rasters. - Computer Vision, Graphics and Image Processing, **26**, 1: 1-16; New York.

- (1984b): The quadtree and related hierarchical data structures. - ACM Computing Surveys, **16**, 2: 187-260; New York.

-, ROSENFELD, A., Shaffer, C.A. & WEBBER, R.E. (1984): A geographic information system using quadtrees. - Pattern Recognition, **17**, 6: 647-656; New York.

- & TAMMINEN, M. (1985): Computing geometric properties of images represented by linear quadtrees. - IEEE Transactions on Pattern Analysis and Machine Intelligence, **7**, 2: 229-240; New York.

- & WEBBER, R.E. (1985): Storing a collection of polygons using quadtrees. - ACM Transactions on Graphics, **4**, 3: 182-222; New York.

-, ROSENFELD, A., SHAFFER, C.A., NELSON, R.C., HUANG, Y.G. & FUJIMURA, K. (1985): Applications of hierarchical data structures to geographical information systems phase IV. - Computer Science, **TR-1578**, Univ. Maryland; College Park.

SHAFFER, C.A. (1986): Application of alternative quadtree representation. - Ph.D. Diss., **TR-1672**, Computer Science Department, Univ. Maryland; College Park.

-, SAMET, H., WEBBER, R.E., NELSON, R.C. & HUANG, Y.G. (1985): An implementation for a geographic information system based on quadtrees. - Tutorial Lecture Notes, **25**, SIGGRAPH '85: 23-86; San Francisco.

- & SAMET, H. (1987): Optimal quadtree construction algorithms. - Computer Vision, Graphics and Image Processing **31**, 3: 402-419; New York.

SHNEIER, M. (1981): Calculations of geometric properties using quadtrees. - Computer Vision, Graphics and Image Processing, **16**, 3: 296-302; New York.