USING QUADTREES TO REPRESENT SPATIAL DATA

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

ABSTRACT

Use of the quadtree data structure in representing spatial data is reviewed. The focus is on its properties that make it appropriate for applications in image processing. A number of operations in which the quadtree finds use are discussed.

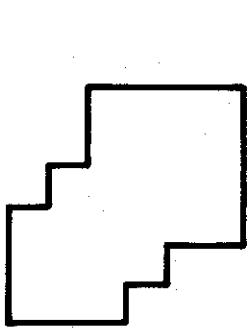
1. INTRODUCTION

The quadtree is a term used to describe a class of hierarchical data structures whose common property is that they are based on the principle of regular decomposition. Such data structures are becoming increasingly important as representations in the fields of image processing, computer graphics, and geographic information systems [1]. The numerous variants of quadtrees can be differentiated on the basis of the type of data that they are used to represent, and on the principle guiding the decomposition process. Presently, quadtrees are used for point data, regions, curves, and volumes. The decomposition may be into equal-sized parts (termed a regular decomposition), or it may be governed by the input. In this chapter we focus on quadtree representations of two-dimensional binary regions and to a minor extent on point and curvilinear data. The chapters by Freeman and Rosenfeld discuss the related octree and pyramid representations respectively.

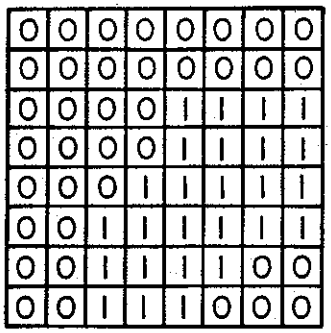
In order to illustrate the quadtree data structure, consider the region shown in Figure 1a which is represented by a 2**3 by 2**3 binary array in Figure 1b. 1's correspond to picture elements (termed pixels) which are in the region and 0's correspond to picture elements that are outside the region. Quadtrees represent regions by successively subdividing their array representation into four equal-size quadrants. When the array does not consist entirely of 1's or 0's

*The support of the Engineering Topographic Laboratories (under Contract DAAK-70-31-C0059) is gratefully acknowledged, as is the help of Janet Salzman in preparing this paper.

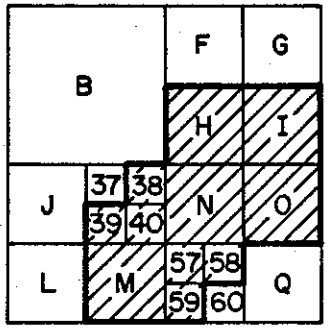
NATO ASI Series, Vol. F18
Computer Architectures for Spatially Distributed Data
Edited by H. Freeman and G. G. Pieroni



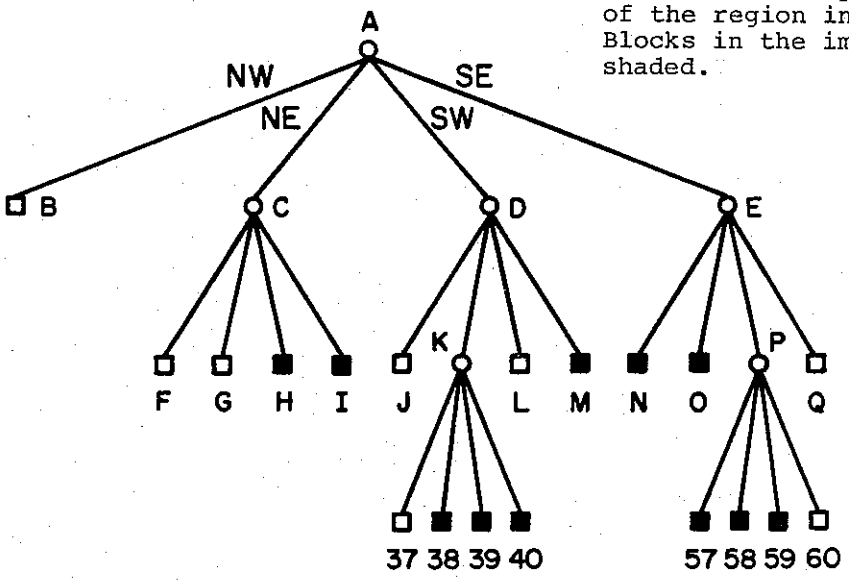
(a) Region



(b) Binary array



(c) Block decomposition of the region in (a). Blocks in the image are shaded.



(d) Quadtree representation of the blocks in (c).

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

(i.e., the region does not cover the entire array), we subdivide it further into quadrants, subquadrants, ... until we obtain blocks (possibly single pixels) that consist of 1's or 0's; i.e., they are entirely contained in the region or entirely disjoint from it. For example, the resulting blocks for the binary array of Figure 1b are shown in Figure 1c. This process is represented by a tree of out degree 4 (i.e., each non-leaf node has four sons) in which the root node corresponds to the entire array. The four sons of the root node represent the quadrants (labeled in order NW, NE, SW, SE), and the leaf nodes correspond to those blocks for which no further subdivision is necessary. Leaf nodes are said to be BLACK or WHITE depending on

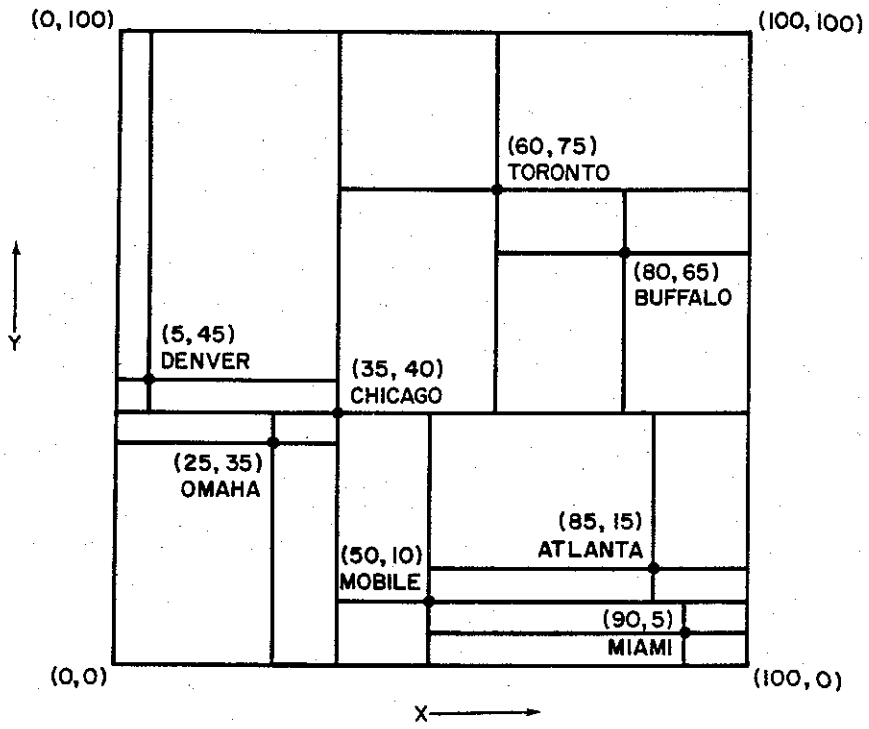
whether their corresponding blocks are entirely within or outside of the region respectively. All non-leaf nodes are said to be GRAY. The quadtree representation for Figure 1c is shown in Figure 1d.

As described above, the region quadtree 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 [2,3] who used the term Q-tree whereas Hunter [4] was the first to use the term quadtree in such a context. A similar partition of space into rectangular quadrants, also termed a quadtree, is due to Finkel and Bentley [5]. It is an adaptation of the binary search tree [6] to two dimensions (and can be easily extended to an arbitrary number of dimensions). It is primarily of use for representing multidimensional point data and we refer to it as a point quadtree. As an example, consider the tree in Figure 2 which is built for the sequence Chicago, Mobile, Toronto, Buffalo, Denver, Omaha, Atlanta, and Miami. Note that its shape is highly dependent on the order in which the points are added to the tree. For an improvement on the point quadtree see the k-d tree of Bentley [7]. The survey of Bentley and Friedman [8] describes related data structures.

The principle of recursive decomposition has been frequently used. Warnock [9] implemented a hidden surface elimination algorithm using a recursive decomposition of the picture area. It is repeatedly subdivided into successively smaller rectangles while searching for areas sufficiently simple to be output. Other early uses include robotics [10], space planning in an architectural context [11], and edge detection [12]. Related developments in the image processing domain include the recognition cone [13], the preprocessing cone [14], and the pyramid [15].

2. MAXIMAL BLOCK REPRESENTATIONS

A number of region representations are characterized as being a collection of maximal blocks that are contained in a given region. The simplest such representation is the run length where the blocks are 1 by m rectangles [16]. A more general representation treats the region as a union of maximal square blocks (or blocks of any desired shape) that it contains. The region is determined by specifying the centers and radii of these blocks. This representation is called the medial axis transformation (MAT) [17,18]. The quadtree is a variant on the maximal block representation where the blocks are disjoint,



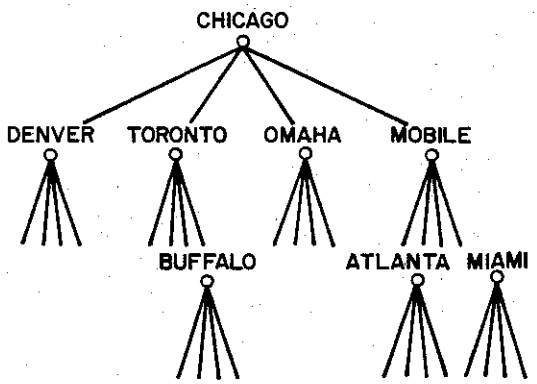


Figure 2. A point quadtree and the records it represents.

square, and have standard sizes (i.e., sides of lengths that are powers of two) and positions.

It should be clear that the quadtree is not a unique image representation. Representations based on triangular and hexagonal tessellations are also appropriate. In general, a planar decomposition should be an infinitely repetitive pattern and also should be infinitely decomposable into increasingly finer patterns [19]. The latter requirement is not satisfied by the hexagonal tessellation since a hexagon cannot be decomposed into smaller hexagons although hexagon-based systems do exist [20]. The choice between square and triangle quadtrees depends on the grid (i.e., the result of a sampling process). Our discussion is limited to the square quadtree.

3. NEIGHBOR FINDING TECHNIQUES

Most of the operations that we wish to perform on quadtrees are implemented as tree traversals. The difference between them is in the nature of the computation that is performed at the node. Often, these computations involve the examination of some nodes that are adjacent to the node being processed (i.e., the blocks corresponding to the nodes are touching along a common side). We call such nodes, corresponding to blocks of greater than or equal size, neighbors (the neighbor may be GRAY). In order for the operations to be performed in the most general manner, we must be able to locate neighbors in a way that is independent of both position (i.e., the coordinates) and size of the node. We also do not want to use any additional links to adjacent nodes. In other words, we only use the structure of the tree and no pointers in excess of the four links from a node to its four sons and one link to its father for a non-root node. This is in contrast with the methods of Klinger and Rhodes [21] which make use of size and position information, and those of Hunter and Steiglitz [4, 22,23] which locate neighbors through the use of explicit links (termed ropes and nets).

It is quite easy to locate adjacent neighbors in the horizontal or vertical directions. The basic idea is to ascend the tree until a common ancestor is located, and then descend back down the tree in search of the neighboring node. For example, suppose we wish to find the western neighbor of node N in Figure 1. The nearest common ancestor is the first ancestor node which is reached via its NE or SE son (i.e., the first ancestor node of which N is not a western descendant). Next, we retrace the path used to locate the common ancestor, except

that we make mirror image moves about an axis formed by the common boundary between the nodes. In the case of a western neighbor, the mirror images of NW and SW are NE and SE respectively. Therefore, the western neighbor of node N in Figure 1 is node K. It is located by ascending the tree until the nearest common ancestor, A, has been located. This requires going through a NW link to reach node E, and a SE link to reach node A. Node K is subsequently located by backtracking along the previous path with the appropriate mirror image moves (i.e., following a SW link to reach node D, and a NE link to reach node K).

It should be clear that neighbors need not be of the same size. If the neighbor is larger, then only part of the path from the common ancestor is retraced. Note that similar techniques can be used to locate diagonal neighbors (i.e., nodes corresponding to blocks that touch a given node's block at a corner). For example, node 57 in Figure 1 is the SE neighbor of node 40. For more details see [24].

4. CONVERSION

The quadtree is a useful representation for binary images because its hierarchical nature facilitates the performance of a large number of operations. Nevertheless, images are traditionally represented using binary arrays, rasters (i.e., run lengths), chain codes (i.e., borders), or polygons (vectors). Some of these representations are chosen due to hardware reasons (e.g., run lengths are particularly useful for raster-like devices such as television). Thus we need techniques to efficiently switch between these various representations.

The most common image representation is the binary array. There are a number of ways of constructing a quadtree from a binary array. The simplest approach is one that converts the array to a complete quadtree (i.e., for a 2**n by 2**n image, a tree of height n with one node per pixel). The resulting quadtree is subsequently reduced in size by repeatedly attempting to merge groups of four pixels or four blocks of a uniform color that are appropriately aligned. This approach is simple but is extremely wasteful of storage since many nodes may be needlessly created. In fact, it is not inconceivable to exhaust available memory when an algorithm employing this approach is used while the resulting tree fits in the available memory.

We can avoid the needless creation of nodes by visiting the elements of the binary array in the order defined by the labels on the

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

Figure 3. Binary array representation of the region in Figure 1a.

array in Figure 3 which corresponds to the image of Figure 1. Using such a method we never create a leaf node until it is known to be maximal. Equivalently, we never need to merge four leaves of the same color and change the color of their parent from GRAY to BLACK or WHITE as is appropriate. For example, we note that since pixels 25,26,27, and 28 are all BLACK, no quadtree nodes were created for them - i.e., node H corresponds to the part of the image spanned by them. This algorithm is shown in [25] to have an execution time proportional to the number of pixels in the image.

When a raster representation is used, we have to scan the array in a row by row manner as we build the quadtree. Such an algorithm, having an execution time proportional to the number of pixels in the image, is described in [26]. The reverse process is also useful since output is usually done on a raster device. The most obvious method is to generate an array corresponding to the quadtree. However, this method may require more memory than is available and we do not consider it further. In [27] a number of quadtree to raster algorithms are described. All of the algorithms traverse the quadtree by rows and visit each quadtree node once for each row that intersects it. These algorithms have execution times that only depend on the number of blocks in the image (irrespective of their color) and not on their particular configuration.

Another very common representation used in cartographic applications is the chain code (also known as a boundary code). It can be specified, relative to a given starting point, as a sequence of unit vectors (i.e., one pixel long) in the principal directions. We can represent the directions by numbers, e.g., let i, an integer quantity ranging from 0 to 3, represent a unit vector having a direction of 90*i degrees. For example, the chain code for the boundary of the region in Figure 1, moving clockwise starting from the

left of the uppermost border points, is

$$0^4 3^4 2^2 3^1 2^1 1^1 2^3 1^3 0^1 1^1 0^1 1^2.$$

An algorithm for the conversion of quadtrees to chain codes is given in [29] and the algorithm for the reverse process of converting chain codes to quadtrees is given in [30].

Use of the chain code corresponds to approximating a polygon by unit vectors. It is also common to represent polygonal data by a set of vertices, or even a point and a sequence of vectors consisting of (magnitude, direction) pairs. Hunter and Steiglitz [4,22,23] address the problem of representing simple polygons (i.e., polygons with non-intersecting edges) using quadtrees. A polygon is represented by a three-color variant of the quadtree. In essence, there are three types of nodes - interior, boundary, and exterior. A node is said to be of type boundary if an edge of the polygon passes through it. Boundary nodes are not subject to merging. Interior and exterior nodes correspond to areas within, and outside of, respectively, the polygon and can be merged to yield larger nodes. Figure 4 illustrates a sample polygon and its quadtree corresponding to the definition of [22]. The disadvantage of such a representation for polygonal lines is that a width is associated with them whereas in a purely technical sense these lines have a width of zero. Algorithms for building a quadtree from a polygon are presented in [4,22].

5. SET OPERATIONS

Perhaps the most useful application of the quadtree is the performance of set operations such as union (i.e., overlay) and intersection

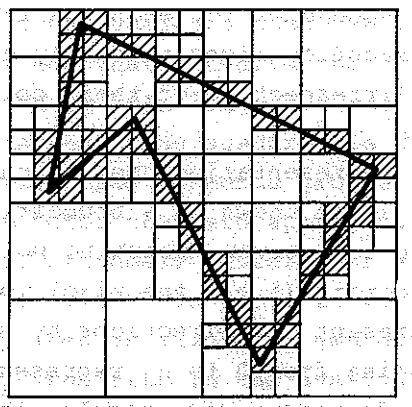


Figure 4. Hunter and Steiglitz's quadtree representation of a polygon.

...eral images. This is described in greater detail in [4,22,31]. ...mple, to obtain the quadtree corresponding to the union of S ... we merely traverse the two trees in parallel while constructing ...sulting tree, say U. If either of the two nodes is BLACK, ...the corresponding node in U is BLACK. If one node is WHITE, say ...en the corresponding node in U is set to the other node, i.e., ... both nodes are GRAY, then U is set to GRAY and the algorithm ...plied recursively to the sons of S and T. However, when both ...are GRAY, once the sons have been processed, we must check if ...ger is to take place since all four sons could be BLACK. Com- ... the intersection of two quadtrees is analogous to computing ...ion with the roles of BLACK and WHITE interchanged.

TRANSFORMATIONS

The impetus for the development of the quadtree concept was a ... to provide an efficient data structure for computer graphics. ...k [9] used recursive decomposition as the basis for the hidden ...ce elimination algorithm. Hunter's Ph.D. thesis [4] was a sig- ...ant extension of the quadtree concept from both a theoretical ...practical standpoint. Hunter's goal was to provide a framework ...performing computer animation efficiently. In order to do this, ...ability is necessary to perform a number of basic transforma- ... Scaling by a power of two is trivial when using quadtrees ... it corresponds to a reduction in resolution. Rotation by mul- ...s of 90 degrees is equally simple - i.e., a recursive rotation ...ons at each level of the tree. The transformation of one quadtree ... another by applying a linear operator is also feasible [22]. The linear transformation algorithm, and the scaling and rotation ...tions have a common shortcoming. With the exception of scaling ...ranslations by a power of two and transformations involving ro- ...ions in multiples of 90 degrees, the results are approximations. ...ight lines are not necessarily transformed into straight lines. ... shortcoming is often mistakenly attributed to the quadtree rep- ...ntation where in fact it is a direct result of the underlying ...lization process. It should be clear that it manifests itself ...matter what underlying representation is used when doing raster ...phics. For a quadtree-based representation that is free of such ...roblem see the PM quadtree [32].

Quadtrees have also been used for image processing operations ...ch involve gray-scale images rather than binary images. Some

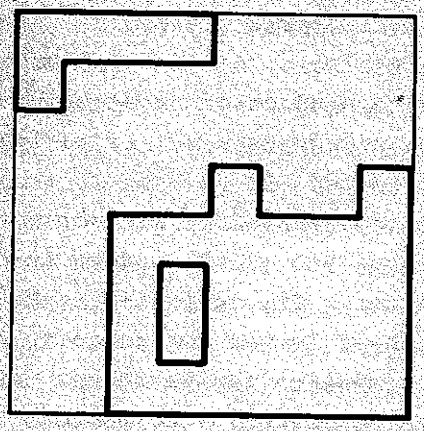
examples include image segmentation [33], edge enhancement [34], image smoothing [35], and threshold selection [36].

7. GEOMETRIC PROPERTIES

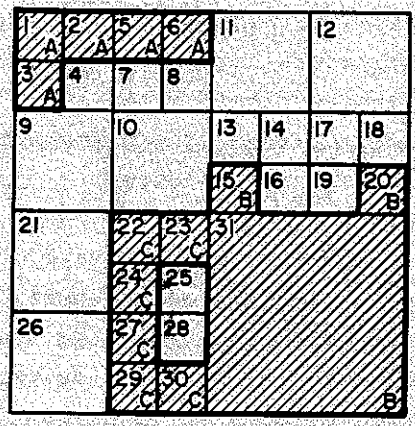
Areas and moments for images represented by quadtrees are easy to compute. To find the area we only need to traverse the quadtree in postorder and accumulate the sizes of the BLACK blocks. For a BLACK block at level k, the contribution to the area is 2**(2*k). Moments can be computed with equal ease - i.e., we simply sum the moments of the BLACK blocks. The position of each BLACK block is easy to ascertain because we know the path that was taken to reach the block when we start processing at the root of the tree. With knowledge of the area and the first moments, we can compute the coordinates of the centroid and now central moments relative to the centroid can be obtained [31].

One of the basic operations in any image processing system is connected component labeling. In graph-theoretical terms, it is analogous to finding the connected components of a graph. For example, the image of Figure 5 has two components. Given a binary array representation, the traditional method of performing this operation is to scan the image row by row from left to right and assign the same label to adjacent BLACK pixels that are found to the right and in the downward direction. During this process pairs of equivalences may be generated and thus two more steps are needed. The first merges the equivalences and the second updates the labels associated with the various pixels to reflect the merger of the equivalences.

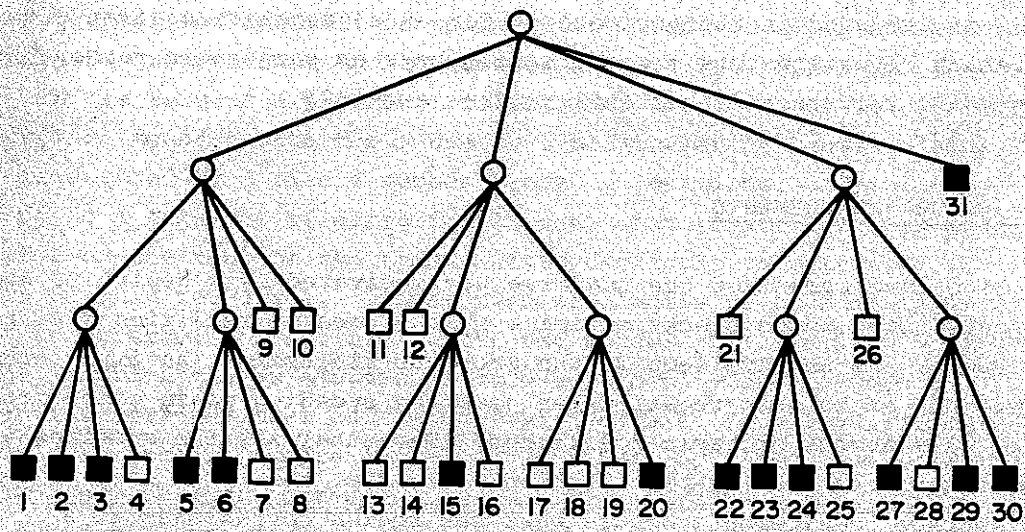
When an image is represented by a quadtree, we perform an analogous three-step process [37]. The first step is a postorder tree traversal where for each BLACK node that is encountered, say A, we find all adjacent BLACK nodes on the southern and eastern sides of A and assign them the same label as A. Adjacency exploration is done using the neighbor finding techniques described in [24]. At times, the adjacent node may already have been assigned a label in which case we note the equivalence. The second step merges all the equivalent pairs that were generated during step one. The third step performs another traversal of the quadtree and updates the labels on the nodes to reflect the equivalences generated by the first two steps of the algorithm.



(a) Image



(b) Block decomposition of the image in (a).



(c) Quadtree representation of the blocks in (b).

Figure 5. An image, its maximal blocks, and the corresponding quadtree. Blocks in the image are shaded, background blocks are blank.

The execution time for labeling the connected components can be obtained by examining the three steps of the algorithm. Let B be the number of BLACK nodes in the quadtree. Steps 1 and 3 are of O(B) while step 2, the mergei of equivalence classes, is known to be of O(B*log B) [38] and thus the algorithm is of O(B*log B). This is a very important result because it is dependent only on the number of blocks in the image and not on their size. In contrast, the analogous algorithm for the binary array has an execution time that is proportional to the number of pixels and hence to the size of the

blocks. Thus we see that the hierarchical structure of the quadtree data structure not only saves space but also saves time. A somewhat analogous result is shown in [39] as a byproduct of an algorithm for the computation of the Euler number (i.e., genus) [40] of an image represented by a quadtree.

Perimeter computation of an image represented by a quadtree [41] can be done in a manner analogous to step one of the connected component labeling process described earlier. The only difference is that when labeling connected components we looked for adjacent BLACK neighbors whereas for the purpose of computing the perimeter we must look for adjacent WHITE neighbors. In other words, we perform a post-order tree traversal and for each BLACK node that is encountered, we explore its four adjacent sides looking for WHITE neighbors. For each WHITE neighbor that is found, the length of the corresponding shared side is included in the perimeter. For an alternative perimeter computation algorithm that transmits neighbors as parameters rather than having to rely on neighbor exploration, see [42].

8. SPACE REQUIREMENTS

The development of the quadtree was motivated by a desire to aggregate homogeneous blocks of space in the hope of realizing savings in space. As we have seen in the previous discussion, an important byproduct of this aggregation has been the speeding up in execution time of a number of basic operations. Nevertheless, the quadtree is not always the ideal representation. Clearly, the worst case in terms of storage requirements occurs when the region corresponds to a checkerboard pattern. The amount of space required is a function of the resolution (i.e., the number of levels in the tree). Hunter [4] has proved that the quadtree grows linearly in the number of nodes as the resolution is doubled whereas when using a binary array representation, each doubling of the resolution leads to a quadrupling of the number of pixels.

The space required by a quadtree is very sensitive to its orientation. Dyer [43] has shown that the amount of space necessary when arbitrarily placing a square of size 2**m by 2**m at any position in a 2**n by 2**n image is o(p+n) when p is the perimeter (in pixel widths) of the block. Clearly, shifting the image within the space in which it is embedded can reduce the total number of nodes. Grosky and Jain [44] have shown that for a region such that d is the maximum of its

horizontal and vertical extent (measured in pixel widths) and 2**(n-1)< d<2**n, then the optimal grid resolution is either n or n+1. In other words, embedding the region in an area larger than 2**(n+1) by 2**(n+1) and shifting it around will not lead to fewer nodes being required. This result is used by Li, Grosky, and Jain [45] to obtain an algorithm which finds the optimal configuration of the quadtree in the sense of requiring a minimum number of nodes. The shift sensitivity of the quadtree data structure can be reduced, at times, by using the Quadtree Medial Axis Transform (QMAT) [46] which is based on a partition of space into square blocks, possibly non-disjoint, of side lengths which are sums of powers of two rather than disjoint square blocks of side lengths that are powers of two as is the case for the quadtree.

The fact that the quadtree data structure requires pointers leads to a considerable amount of overhead. Recently, there has been an increasing amount of interest in pointer-less quadtree representations. They can be grouped into two categories. The first represents the image in the form of a preorder traversal of the nodes of its quadtree [47]. The second treats the image as a collection of leaves. 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 tree. It is difficult to attribute the origin of this technique. It was used as a means of organizing quadtrees on external storage by Klinger and Rhodes [21]. A base 5 variant of it which has an additional code as a don't care is used by Gargantini [48] and Abel and Smith [49] (see also [50,51,52,53,54]) to yield an encoding where each leaf in a 2**n by 2**n image is n digits long. A leaf corresponding to a 2**k by 2**k block (k<n) will have n-k don't care digits.

9. BOUNDARY REPRESENTATIONS

The region quadtree is an approach to region representation that is based on describing its interior. There also exist representations that specify borders of regions. One of the most common representations is the chain code [28]. Other popular representations include polygons in the form of vectors [55]. Recently, there has also been a considerable amount of interest in hierarchical representations. These are primarily based on rectangular approximations to the data [56,57, 58]. In particular, Burton [57] uses upright rectangles, Ballard [56] uses rectangular strips of arbitrary orientation, and Peucker [58] uses sets of bands. There also exist methods that are based on a

regular decomposition in two dimensions as reported by Hunter and Steiglitz [22], Shneier [59], Martin [60], and Samet and Webber [32,61].

The edge quadtree of Shneier [59] is an example of a quadtree-based boundary representation. It is an attempt to store linear feature information (e.g., curves) for an image (binary and gray-scale) in a manner analogous to that used for storing region information. A region containing a linear feature or part thereof is subdivided into four squares repeatedly until a square is obtained that contains a single curve that can be approximated by a single straight line. Each leaf node contains the following information about the edge passing through it: magnitude (i.e., 1 in the case of a binary image or the intensity in case it is a gray-scale image), direction, intercept, and a directional error term (i.e., the error induced by approximating the curve by a straight line using a measure such as least squares). If an edge terminates within a node, then a special flag is set and the intercept denotes the point at which the edge terminates. Applying this process leads to quadtrees in which long edges are represented by large leaves or a sequence of large leaves. However, small leaves are required in the vicinity of corners or intersecting edges. Of course, many leaves will contain no edge information at all. Note that the edge quadtree is pixel-based and thus the accuracy of the resulting approximation is constrained, in part, by the resolution of the data being represented. For a representation that does not suffer from this problem, see the PM quadtree [32]. As an example of the decomposition that is imposed by the edge quadtree, consider Figure 6 which is the edge quadtree corresponding to the polygon of Figure 4 when represented on a 2**4 by 2**4 grid. What is desired is a regular decomposition

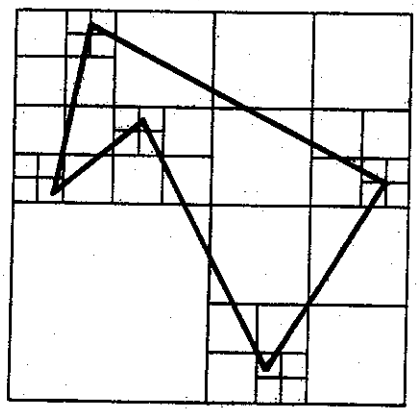


Figure 6. The edge quadtree corresponding to the polygon of Figure 4.

strip tree or variant thereof.

10. CONCLUDING REMARKS

In this chapter, we have attempted to review the use of the quadtree data structure for representing spatial data. We have seen that the quadtree is a representation that can be applied in many traditional image processing operations. Its value is not merely in the saving of space but more importantly in the speeding up of the execution times of these operations. As time passes, alternative representations to the pointer-based quadtree will undoubtedly be developed (e.g., [62]). However, conceptually speaking, the principle of recursive decomposition, of which the quadtree is an embodiment, will continue to be of utility.

REFERENCES

1. A. Rosenfeld, H. Samet, C. Shaffer, and R. E. Webber, Application of hierarchical data structures to geographical information systems, Computer Science TR-1197, University of Maryland, College Park, MD, June 1982.

2. A. Klinger, Patterns and search statistics, in Optimizing Methods in Statistics, J. S. Rustagi, Ed., Academic Press, New York, 1971.

3. A. Klinger and C. R. Dyer, Experiments in picture representations using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

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

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

6. D. E. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms, Second Edition, Addison-Wesley, Reading, MA, 1975.

7. J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18, September 1975, 509-517.

8. J. L. Bentley and J. H. Friedman, Data structures for range searching, ACM Computing Surveys 11, December 1979, 397-409.

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

10. N. J. Nilsson, A mobile automaton: an application of artificial intelligence techniques, Proceedings of the First International Joint Conference on Artificial Intelligence, Washington, DC, 1969, 509-520.

11. C. M. Eastman, Representations for space planning, Communications of the ACM 13, April 1970, 242-250.

12. M. D. Kelly, Edge detection in pictures by computer using planning Machine Intelligence 6, 1971, 397-409.

13. L. Uhr, Layered "recognition cone" networks that preprocess, classify, and describe, IEEE Transactions on Computers 21, 1972, 758-768.

14. E. M. Riseman and M. A. Arbib, Computational techniques in the visual segmentation of static scenes, Computer Graphics and Image Processing 6, 1977, 221-276.

15. S. Tanimoto and T. Pavlidis, A hierarchical data structure for picture processing, Computer Graphics and Image Processing 4, 1975, 104-119.

16. D. Rutovitz, Data structures for operations on digital images, in Pictorial Pattern Recognition, G. C. Cheng et al.,Eds., Thompson Book Co., Washington, DC, 1968, 105-133.

17. H. Blum, A transformation for extracting new descriptors of shape, in Models for the Perception of Speech and Visual Form, W. Wathen-Dunn, Ed., M.I.T. Press, Cambridge, MA, 1967, 362-380.

18. A. Rosenfeld and J. L. Pfaltz, Sequential operations in digital image processing, Journal of the ACM 13, October 1966, 471-494.

19. N. Ahuja, On approaches to polygonal decomposition for hierarchical image representation, to appear in Computer Vision,Graphics and Image Processing, 1983 (see also Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, 1981, 75-80).

20. L. Gibson and D. Lucas, Vectorization of raster images using hierarchical methods, Computer Graphics and Image Processing 20, 1982, 82-89.

21. A. Klinger and M. L. Rhodes, Organization and access of image data by areas, IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 1979, 50-60.

22. G. M. Hunter and K. Steiglitz, Operations on images using quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 1, 1979, 145-153.

23. G. M. Hunter and K. Steiglitz, Linear transformation of pictures represented by quadtrees, Computer Graphics and Image Processing 10, 1979, 289-296.

24. H. Samet, Neighbor finding techniques for images represented by quadtrees, Computer Graphics and Image Processing 18, 1982, 37-57.

25. H. Samet, Region representation: quadtrees from binary arrays, Computer Graphics and Image Processing 18, 1980, 88-93.

26. H. Samet, An algorithm for converting rasters to quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 3, 1981, 487-501.

27. H. Samet, Algorithms for the conversion of quadtrees to rasters, to appear in Computer Vision, Graphics, and Image Processing, 1933 (also University of Maryland Computer Science TR-979).

28. H. Freeman, Computer processing of line-drawing images, ACM Computing Surveys 6, March 1974, 57-97.

29. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, Communications of the ACM 23, March 1980, 171-179.

30. H. Samet, Region representation: quadtrees from boundary codes, Communications of the ACM 23, March 1980, 163-170.

31. M. Shneier, Calculations of geometric properties using quadtrees, Computer Graphics and Image Processing 16, 1981, 296-302.

32. H. Samet and R. E. Webber, Using quadtrees to represent polygonal maps, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC, 1983, 127-132.

33. S. Ranade, A. Rosenfeld, and J. M. S. Prewitt, Use of quadtrees for image segmentation, Computer Science TR-878, University of Maryland, College Park, MD, February 1980.

34. S. Ranade, Use of quadtrees for edge enhancement, IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, 370-373.

35. S. Ranade and M. Shneier, Using quadtrees to smooth images, IEEE Transactions on Systems, Man, and Cybernetics 11, 1981, 373-376.

36. A. Y. Wu, T. H. Hong, and A. Rosenfeld, Threshold selection using quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 1982, 90-94.

37. H. Samet, Connected component labeling using quadtrees, Journal of the ACM 28, July 1981, 487-501.

38. R. E. Tarjan, On the efficiency of a good but not linear set union algorithm, Technical Report 72-148, Computer Science Department, Cornell University, Ithaca, New York, November 1972.

39. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, Communications of the ACM 23, March 1980, 171-179.

40. M. Minsky and S. Papert, Perceptrons: An Introduction to Computional Geometry, MIT Press, Cambridge, MA, 1969.

41. H. Samet, Computing perimeters of images represented by quadtrees, IEEE Transactions on Pattern Analysis and Machine Intelligence 3, 1981, 683-687.

42. C. Jackins and S. L. Tanimoto, Quad-trees, oct-trees, and k-trees - a generalized approach to recursive decomposition of Euclidean space, Department of Computer Science Technical Report 82-02-02, University of Washington, Seattle, 1982.

43. C. R. Dyer, The space efficiency of quadtrees, Computer Graphics and Image Processing 19, 1982, 335-348.

44. W. I. Grosky and R. Jain, Optimal quadtrees for image segments, IEEE Transactions on Pattern Analysis and Machine Intelligence 5, 1983, 77-83.

45. M. Li, W. I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, Computer Graphics and Image Processing 20, 1982, 72-81.

46. H. Samet, A quadtree medial axis transform, Communications of the ACM, 26, November 1983, 680-693.

47. E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression, IEEE Transactions on Pattern Analysis and Machine Intelligence 2, 1980, 27-35.

48. I. Gargantini, An effective way to represent quadtrees, Communications of the ACM 25, December 1982, 905-910.

49. D. J. Abel and J. L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, to appear in Computer Vision, Graphics and Image Processing, 1983.

50. G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, IBM Canada, 1966.

51. B. G. Cook, The structural and algorithmic basis of a geographic data base, in Proceedings of the First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, G. Dutton, Ed., Harvard Papers on Geographic Information Systems, 1978.

52. W. Weber, Three types of map data structures, their ANDs and NOTs, and a possible OR, in Proceedings of the First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, G. Dutton, Ed., Harvard Papers on Geographic Information Systems, 1978.

53. J. R. Woodwark, The explicit quadtree as a structure for computer graphics, Computer Journal 25, 1982, 235-238.

54. M. A. Oliver and N. E. Wiseman, Operations on quadtree-encoded images, Computer Journal 26, 1983, 83-91.

55. G. Nagy and S. Wagle, Geographic data processing, ACM Computing Surveys 11, June 1979, 139-181.

56. D. H. Ballard, Strip trees: A hierarchical representation for curves, Communications of the ACM 24, May 1981, 310-321 (see also corrigendum, Communications of the ACM 25, March 1982, 213).

57. W. Burton, Representation of many-sided polygons and polygonal lines for rapid processing, Communications of the ACM 20, March 1977, 166-171.

58. T. Peucker, A theory of the cartographic line, International Yearbook of Cartography, 1976.

59. M. Shneier, Two hierarchical linear feature representations: edge pyramids and edge quadtrees, Computer Graphics and Image Processing 17, 1981, 211-224.

60. J. J. Martin, Organization of geographical data with quad trees and least square approximation, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, 1982, 458-463.

61. H. Samet and R. E. Webber, Line quadtrees: a hierarchical data structure for encoding boundaries, Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, 1982, 90-92 (also University of Maryland Computer Science TR-1162).

62. M. Tamminen, Encoding pixel trees, Laboratory of Information Processing Science, Helsinki University of Technology, Espoo, Finland, 1983.