TREE STRUCTURES FOR REGION REPRESENTATION

Azriel Rosenfeld and Hanan Samet
Computer Vision Laboratory
University of Maryland
College Park, MD 20742

1. Introduction

Region representation plays a key role in image and scene analysis, computer cartography, and computer graphics. There are a variety of approaches to representing regions, based on their boundaries or their "skeletons"; some of these are reviewed in the following paragraphs. Recently, a tree representation has been proposed which offers a number of advantages; it is also described below.

We assume in what follows that a region is an arbitrary subset of a 2^n-by-2^n array, which we regard as being made up of unit-square "pixels". Any boundary of such a region can thus be specified, relative to a given starting point, as a sequence of unit vectors in the principal directions. We can represent the directions by numbers, e.g., let i represent $90i°$ $(i=0,1,2,3)$. For example, the direction sequence for the boundary of the region in Figure 1a, moving clockwise starting from the left-most of the upper-most border points, is

$$0\ 3\ 0^2\ 3^5\ 2^3\ 1\ 2\ 3^3\ 0\ 3\ 2^5\ 1^6\ 0\ 1\ 0\ 1\ 0\ 3\ 0\ 1\ 0\ 1.$$

This type of boundary representation is called a chain code. Generalized chain codes, involving more than four directions, can also be used. Chain codes provide a very compact region representation, and make it easy to detect features of the region boundary, such as sharp turns ("corners") or concavities. On the other

hand, it is harder to determine properties such as elongatedness from a chain code, and it is also difficult to perform operations such as union and intersection on regions represented by chain codes. A general introduction to chain codes and their uses can be found in [1].

Another class of region representations involves various types of maximal "blocks" that are contained in a given region. For example, we can represent a region R as a linked list of the runs (of pixels) in which R meets the successive rows of the array [2]. Here each "block" is a 1-by-m rectangle, where m is the run length; the runs are the largest such blocks that R contains, and R is determined by specifying the initial points (or centers) and lengths of the runs. Alternatively, we can represent R by the set of maximal square blocks (or blocks of any other desired shape) that it contains; here R is determined by specifying the centers and radii of these blocks. This representation is called the medial axis transformation, or MAT [3]. It is somewhat less compact than chain code [4], but it has advantages with respect to performing union and intersection operations or detecting properties such as elongatedness (in terms of the smallness of the radii relative to the number of centers).

There has been recent interest in an approach to region representation based on successive subdivision of the array into quadrants. If the region does not cover the entire array, we subdivide the array, and repeat this process for each quadrant, each subquadrant,... as long as necessary, until we obtain blocks (possibly single pixels) that are entirely contained in the region or entirely disjoint from it. The resulting blocks for the region of Figure 1a are shown in Figure 1b. This process can be represented by a tree of degree 4 (for brevity: a quadtree) in which the entire array is the root node, the four sons of a node are its quadrants, and the leaf nodes correspond to those blocks for which no further subdivision is necessary.* The quadtree representation for Figure 1b is shown in Figure 1c. Note that here again we are representing the region as a union of maximal blocks, but this time the blocks must have standard sizes and positions (powers

*The quadtree region representation described here should not be confused with the quadtree representation of two-dimensional point data introduced by Finkel and Bentley [5].

of 2). Since the array was assumed to be 2^n-by-2^n, the tree height is at most n. This method of region representation was proposed by Klinger [6-8]; it has also been studied extensively by Hunter and Steiglitz [9-11]. It is relatively compact, and is also well suited to operations such as union and intersection, and to detecting various region properties.

This paper informally describes a collection of algorithms for converting between quadtree and other representations, and for measuring geometric properties of regions represented by quadtrees. Detailed descriptions of the algorithms can be found in a series of papers by Samet et al. [12-20].

2. Conversion

2.1 Quadtrees and arrays

In Section 1 we described a method of constructing the quadtree corresponding to a given region by recursively subdividing the picture into blocks which are quadrants, subquadrants,... . We assume that the region is represented by a binary array, with region points having value 1 and nonregion points, value 0. If a block consists entirely of 1's or 0's, it corresponds to a "black" or "white" leaf node in the tree; otherwise, it corresponds to a "gray" nonleaf node, which has four sons corresponding to its four quadrants. If we do this in a "top-down" fashion, i.e., first examine the entire picture, then its quadrants, then their quadrants, etc. as needed, it may require excessive computational effort, since parts of the picture that contain finely divided mixtures of 0's and 1's will be examined repeatedly.

As an alternative, we can build the quadtree "bottom-up" by scanning the picture in a suitable order, e.g., in the sequence

```
 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...
```

where the numbers indicate the order in which the points are examined. As we discover maximal blocks of 0's or

of 1's, we add leaf nodes to the tree, together with their needed ancestor gray nodes. This can be done in such a way that leaf nodes are never created until they are known to be maximal, so that it is never necessary to merge four leaves of the same color and change their common parent node from gray to black or white. For the details of this algorithm see [17].

Bottom-up quadtree construction becomes somewhat more complicated if we want to scan the picture row by row. Here we add leaf nodes to the tree as we discover maximal 1-by-1 or 2-by-2 blocks of 0's or 1's; if four leaves with a common father all have the same color, they are merged. The details can be found in [16].

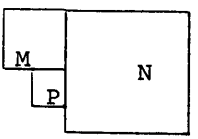
Given a quadtree, we can construct the corresponding binary picture by scanning the tree and, for each leaf, creating a block of 0's or 1's of the appropriate size in the appropriate position. A more complicated process can be used if we want to create the picture row by row. Here we must visit each quadtree node once for each row that intersects it (i.e., a node corresponding to a 2^k by 2^k block is visited 2^k times) and, for each leaf, output a run of 0's or 1's of the appropriate length (2^k) in the appropriate position. For the details, see [18].

2.2 Quadtrees and borders

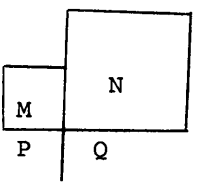
In order to determine, for a given leaf node M of a quadtree, whether the corresponding block is on a border, we must visit the leaf nodes that correspond to 4-adjacent blocks and check whether they are black or white. To find the nodes corresponding to, e.g., right-hand neighbor blocks, we move upward from M in the tree until we reach some ancestor node from its northwest or southwest son. (If we reach the root node before this happens, M is on the east edge of the picture and has no right-hand neighbor blocks.) As soon as this occurs, we go back down the tree making the mirror images of the moves made on the way up--i.e., the first move down is to the northeast or southeast son, and the following moves are to northwest or southwest sons. If a leaf node is reached by the time we come to the end of this move sequence, its block is at least as large as M's block, and so is M's sole right-hand neighbor. Otherwise, the nonleaf node reached at the

end of the sequence is the root of a subtree whose leftmost leaf nodes correspond to M's right-hand neighbors, and we can find these nodes by traversing that subtree.

Let M,N be black and white leaf nodes whose blocks are 4-adjacent. Thus the pair M,N defines a common border segment of length 2^k (the smaller of the side lengths of M and N) which ends at a corner of M or of N (or both). To determine the next segment along this border, we must find the other leaf P whose block touches the end of this segment:



If the segment ends at a corner of both M and N, we must find the other two leaves P,Q whose blocks meet at that corner:



This can be done by an ascending and descending procedure similar to that described in the preceding paragraph; see [12] for the details. The next border segment is then the common border defined by M and P if P is white, or by N and P if P is black. (In the common corner case, the pair of blocks defining the next border segment is determined exactly as in the standard "crack following" algorithm for traversing region borders.) This process is repeated until we come to M,N again, at which stage the entire border has been traversed. The successive border segments constitute a 4-direction chain code, broken up into pieces whose lengths are powers of 2. The time required for this process is on the order of the number of border nodes times the tree height.

Using the methods described in the last two paragraphs, we can traverse the quadtree, find all borders, and generate their codes. We should mark each border as

we follow it, so that we will not follow it again from another starting point; note that the marking process is complicated by the fact that a node's block may be on many different borders.

To generate a quadtree from a set of 4-direction chain codes, we first traverse each code and create pairs of leaf nodes having the given border segments, together with the necessary nonleaf nodes. We then generate leaf (and nonleaf) nodes corresponding to the interior blocks. At any stage, if four leaves with a common father all have the same color, they are merged. The details of this algorithm will not be given here; see [13]. The time required is on the order of the perimeter (=total 4-direction chain code length) times tree height.

2.3 Quadtrees of derived sets

Let S be the set of 1's in a given binary array, and let $\overline{S}$ be the complement of S. The quadtree of $\overline{S}$ is the same as that of S, with black leaf nodes changed to white and vice versa. To get the quadtree of $S \cup T$ from those of S and T, we traverse the two trees simultaneously. Where they agree, the new tree is the same. If S has a gray (=nonleaf) node where T has a black node, the new tree gets a black node; if T has a white node there, we copy the subtree of S at that gray node into the new tree; if S has a white node and T a black node, the new tree gets a black node. The algorithm for $S \cap T$ is exactly analogous, with the roles of black and white reversed. The time required for these algorithms is proportional to the number of nodes in the smaller of the two trees [20].

3. Property Measurement

3.1 Connected component labeling

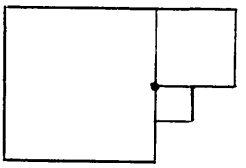
Given a binary array represented by a quadtree, we can label its components by traversing the tree in a standard order, say, NW, NE, SW, SE. Whenever we come to a black leaf node, we visit the leaf nodes whose blocks adjoin M's block on its south and east sides (and at its southeast corner, if we are labeling 8-components); see Section 2.2 on how to find these nodes. If we find unlabeled black leaf nodes, we give them the same label

as M; if we find black leaf nodes that already have labels, we note that their labels are equivalent. When the traversal is complete, we sort out the equivalences, retraverse the tree, and give the black leaf nodes their final labels. The time required is on the order of the number of nodes in the tree times the tree height. Alternatively, it is on the order of Q log Q, where Q is the number of leaf nodes in the tree. For the details of this algorithm, see [15].

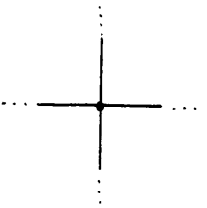
3.2 Component counting and genus computation

Once the connected components have been labeled, it is trivial to count them, since their number is the same as the number of inequivalent labels. We will next describe a method of determining the number of components minus the number of holes by counting certain types of local patterns in the array; this number g is known as the genus or Euler number of the array.

Let V be the number of 1's, E the number of 11's and $\begin{smallmatrix}1\\1\end{smallmatrix}$'s, and F the number of $\begin{smallmatrix}11\\11\end{smallmatrix}$'s in the array; it is well known [21] that the g=V-E+F. This result can be generalized to the case where the array is represented by a quadtree [19]. In fact, let V be the number of black leaf nodes; E the number of pairs of such nodes whose blocks are horizontally or vertically adjacent; and F the number of triples or quadruples of such nodes whose blocks meet at and surround a common point, e.g.,



or



Then g=V-E+F. These adjacencies can be found (see Section 2.2) by traversing the tree; the time required is on the order of the number of nodes in the tree times the tree height, or of Q log Q, as before.

3.3 Area and moments

The area of a region represented by a quadtree can be obtained by summing the areas of the black leaf nodes, i.e., counting 4^h for each such node that represents

a 2^h by 2^h block. Similarly, the first x and y moments of the region relative to a given origin can be computed by summing the first moments of these blocks; note that we know the position (and size) of each block from the position of its leaf in the tree. Knowing the area and the first moments gives us the coordinates of the centroid, and we can then compute central moments relative to the centroid as origin. The time required for any of these computations is essentially proportional to the number of nodes in the tree. Further details on moment computation from quadtrees can be found in [20].

3.4 Perimeter

The perimeter of a region represented by a quadtree can be obtained by traversing all the borders of the region (Section 2.2) and summing the numbers of steps in the resulting codes. Alternatively, it can be computed by traversing the tree, and for each leaf node, checking the colors of the nodes whose blocks are adjacent to its block on two sides, say, bottom and right, to determine which of these adjacencies contributes to the total perimeter. The time required for this is proportional to the number of leaf nodes; see [14] for the details.

4. Concluding Remarks

Quadtrees constitute an interesting alternative to the standard methods of digitally representing regions. Their chief disadvantage is that they are not shift-invariant; two regions differing only by a translation may have quite different quadtrees. Thus shape matching from quadtrees is not straightforward. In other respects, however, they have many potential advantages. They provide a compact and easily constructed representation from which standard region properties can be efficiently computed. In effect, they are "variable-resolution arrays" in which detail is represented only when it is available, without leading to excessive storage requirements for parts where detail is lacking. Hopefully, this paper has drawn attention to the advantages of quadtrees as data structures for possible use in digital cartographic information systems.

References

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

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

3. H. Blum, A transformation for extracting new descriptors of shape, in W. Wathen-Dunn, ed., Models for the Perception of Speech and Visual Form, MIT Press, Cambridge, MA, 1967, 362-380.

4. J. L. Pfaltz and A. Rosenfeld, Computer representation of planar regions by their skeletons, Comm. ACM 10, 1967, 119-122, 125.

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

6. A. Klinger, Data structures and pattern recognition, Proc. 1st Intl. Joint Conf. on Pattern Recognition, 1973, 497-498.

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

8. N. Alexandridis and A. Klinger, Picture decomposition, tree data-structures, and identifying directional symmetries as node combinations, ibid. 8, 1978, 43-77.

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

10. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, IEEETPAMI-1, 1979, 145-153.

11. G. M. Hunter and K. Steiglitz, Linear transformations of pictures represented by quad trees, Computer Graphics Image Processing 10, 1979, 289-296.

12. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, Computer Science Center TR-732, University of Maryland, College Park, MD, February 1979.

13. H. Samet, Region representation: quadtrees from boundary codes, Computer Science Center TR-741, University of Maryland, College Park, MD, March 1979.

14. H. Samet, Computing perimeters of images represented by quadtrees, Computer Science Center TR-755, University of Maryland, College Park, MD, April 1979.

15. H. Samet, Connected component labeling using quadtrees, Computer Science Center TR-756, University of Maryland, College Park, MD, April 1979.

16. H. Samet, Region representation: raster-to-quadtree conversion, Computer Science Center TR-766, University of Maryland, College Park, MD, May 1979.

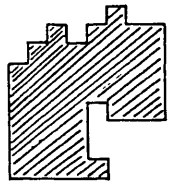
17. H. Samet, Region representation: quadtrees from binary arrays, Computer Science Center TR-767, University of Maryland, College Park, MD, May 1979.

18. H. Samet, Region representation: quadtree-to-raster conversion, Computer Science Center TR-768, University of Maryland, College Park, MD, June 1979.

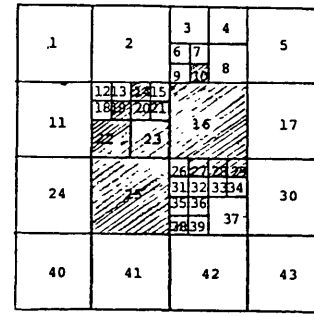
19. C. R. Dyer, Computing the Euler number of an image from its quadtree, Computer Science Center TR-769, University of Maryland, College Park, MD, May 1979.

20. M. Shneier, Linear time calculations of geometric properties using quadtrees, Computer Science Center TR-770, University of Maryland, College Park, MD, May 1979.

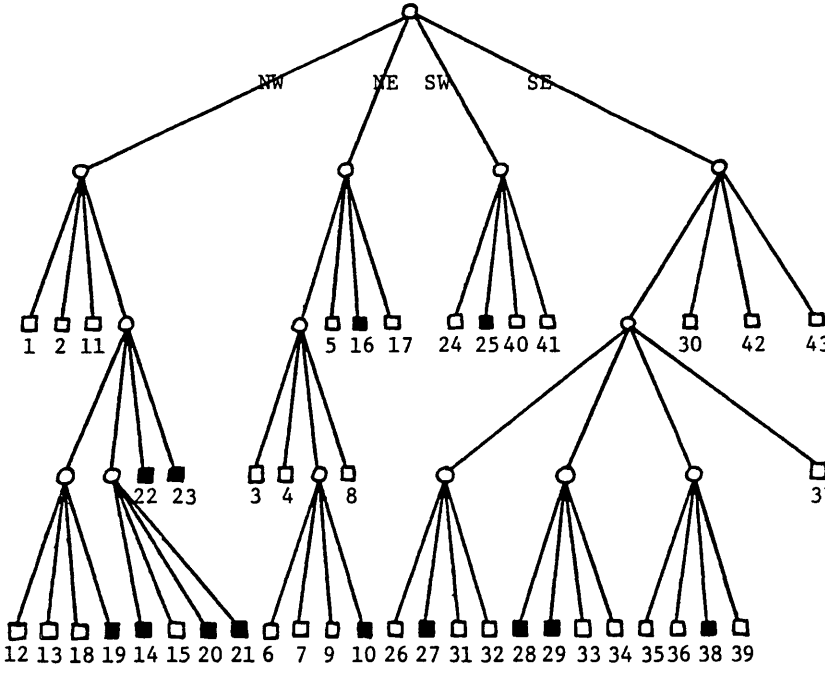
21. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.



a. Region.



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



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

Figure 1. A region, its maximal blocks, and the corresponding quadtree. Blocks in the region are shaded, background blocks are blank.