QUADTREE STRUCTURES FOR REGION PROCESSING

Hanan Samet
Azriel Rosenfeld

Department of Computer Science
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

Research into the use of the quadtree data structure for image processing applications is described. A quadtree represents an image array by a tree of out degree 4 which is constructed by recursively subdividing the array into blocks of constant value. This representation is particularly useful when applied to binary arrays representing regions (i.e., 1's are region points). Algorithms are informally discussed for conversion between this and other representations, and for measuring geometric properties of regions represented in this manner. Results of execution time analyses of these algorithms are also given.

1. INTRODUCTION

Region representation is an important issue in image processing, computer graphics and cartography. There are numerous representations currently in use. In this paper we focus our attention on the quadtree [1,6-11] representation. We discuss its relationship to more traditional representations and present informal descriptions of algorithms for converting between quadtrees and these representations. We also show how geometric properties of regions represented by quadtrees can be measured.

In our discussion we assume that a region is a subset of a 2^n by 2^n array which is viewed as being composed of unit-square pixels. The most common region representations used in image processing are the binary array and the run length representation [14]. The binary array represents region pixels by 1's and non-region pixels by 0's. The run length representation represents each row of the binary array as a sequence of runs of 1's alternating with runs of 0's.

Boundaries of regions are often specified as a sequence of unit vectors in the principal directions. This representation is termed a chain code [5]. For example, letting i represent $90° * i$ (i=0,1,2,3), we have the following sequence as the chain code for the region in Figure 1a:

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

Note that this is a clockwise code which starts at the leftmost of the uppermost border points. Chain codes yield a compact representation; however, they are somewhat inconvenient for performing operations such as set union and intersection.

Regions can also be represented by a collection of maximal blocks that are contained in the given region. One such trivial representation is the run length where the blocks are 1 by m rectangles. A more general representation treats the region as a union of maximal blocks (of 1's) of a given shape. The medial axis transform (MAT) [2,12] is the set of points serving as centers of these blocks and their corresponding radii.

The quadtree is a variant on the maximal block representation in which the blocks have standard sizes and positions (i.e., powers of two). It is an approach to region representation which is based on the successive subdivision of an image array into quadrants. If the array does not consist entirely of 1's or entirely of 0's, then we subdivide it into quadrants, subquadrants,... until we obtain blocks (possibly single pixels) that consist of 1's or of 0's, i.e., they are entirely contained in the region or entirely disjoint from it. This process is represented by a tree of out degree 4 in which the root node represents 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 of the array 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". Since the array was assumed to be 2^n by 2^n, the tree height is at most n. As an example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree.

2. PRELIMINARIES

In the quadtree representation, by virtue of its tree-like nature, most operations are carried out by techniques which traverse the tree. In fact, many of the operations that we describe can be characterized as having two basic steps. The first step either traverses the quadtree in a specified order or constructs a quadtree. The second step performs a computation at each node which often makes use of its neighboring nodes, i.e. nodes representing image blocks that are adjacent to the given node's block. Frequently these two steps are performed in parallel.

In general, we prefer to avoid having to use position (i.e., coordinates) and size information when making relative transitions (i.e., locating neighboring nodes) in the quadtree since they involve computation (rather than simply chasing links) and are clumsy when adjacent blocks are of different sizes (e.g., when a neighboring block is larger). Also, we do not assume that there are links from a node to its neighbors, because we do not want to use links in excess of four links from a non-leaf node to its sons and the link from a non-root node to its father. Thus all of our operations are implemented by algorithms that make use of the existing structure of the tree. This is in contrast with the methods of Klinger and Rhodes [11] which make use of size and position information, and those of Hunter and Steiglitz [6-8] which locate neighbors through the use of explicit links (termed nets and ropes).

Locating neighbors in a given direction is quite straightforward. Given a node corresponding to a specific block in the image, its neighbor in a particular direction (horizontal or vertical) is determined by locating a common ancestor. For example, if we want to find a eastern neighbor, the common ancestor is the first ancestor node which is reached via its NW or SW son. Next, we retrace the path from the common ancestor, but making mirror image moves about the appropriate axis, e.g., to find an eastern or western neighbor, the mirror images of NE and SE are NW and SW, respectively. For example, the eastern neighbor of node 32 in Figure 1c is node 33. It is located by ascending the tree until the common ancestor, H, is found. This requires going through a SE link to reach L and a NW link to reach H. Node 33 is now reached by backtracking along the previous path with the appropriate mirror image moves (i.e., going through a NE link to reach M and a SW link to reach 33).

In general, adjacent neighbors need not be of the same size. If they are larger, then only a part of the path to the common ancestor is retraced. If they are smaller, then the retraced path ends at a "gray" node of equal size. Note that similar techniques can be used to locate diagonal neighbors (i.e., nodes corresponding to blocks that touch the given node's block at a corner). For example, node 20 in Figure 1c is the NW neighbor of node 22. For more details, see [21].

3. CONVERSION

3.1 Quadtrees and Arrays

The definition of a quadtree leads naturally to a "top down" quadtree construction process. This may lead to excessive computation because the process of examining whether a quadrant contains all 1's or all 0's may cause certain parts of the region to be examined repeatedly by virtue of being composed of a mixture of 1's and 0's. Alternatively, a "bottom-up" method may be employed which scans the picture 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 sequence in which the pixels are examined. As maximal blocks of 0's or 1's are discovered, corresponding leaf nodes are added along with the necessary ancestor nodes. This is done in such a way that leaf nodes are never created until they are known to be maximal. Thus there is never a need to merge four leaves of the same color and change the color of their common parent from gray to white or black as is appropriate. See [19] for the details of such a algorithm whose execution time is proportional to the number of pixels in the image.

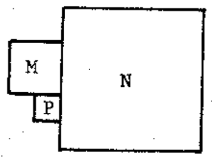
If it is necessary to scan the picture row by row (e.g., when the input is a run length coding) the quadtree construction process is somewhat more complex. We scan the picture a row at a time. For odd-numbered rows, nodes corresponding to the pixel or run values are added to the tree, one node per pixel. For even-numbered rows, nodes are added for the pixels and attempts are made to discover maximal blocks of 0's or 1's whose size depends on the row number (e.g., when processing the fourth row, maximal blocks of maximum size 4-by-4 can be discovered). In such a case merging is said to take place. See [18] for the details of an algorithm that constructs a quadtree from a row by row scan such that at any instant of time a valid quadtree exists. This algorithm has an execution time that is proportional to the number of pixels in the image.

Similarly, for a given quadtree we can output the corresponding binary picture by traversing the tree in such a way that for each row the appropriate blocks are visited and a row of 0's or 1's is output. In essence, we visit each quadtree node once for each row that intersects it (i.e., a node corresponding to a block of size 2^K by 2^K is visited 2^K times). For the details see [20] where an algorithm is described whose execution time depends only on the number of blocks of each size that comprise the image - not on their particular configuration.

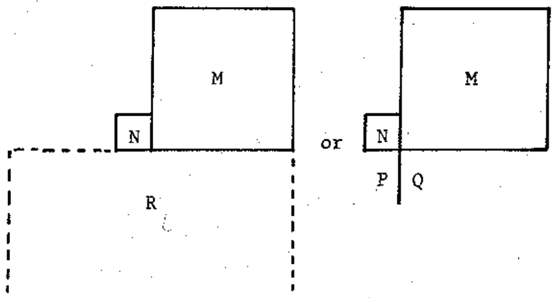
3.2 Quadtrees and Borders

In order to determine, for a given leaf node M of a quadtree, whether the corresponding block is on the border, we must visit the leaf nodes that correspond to 4-adjacent blocks and check whether they are black or white. For example, to find M's right hand neighbor we use the neighbor finding techniques outlined in Section 2. If the neighbor is a leaf node, then its block is at least as large as that of M and so it is M's sole neighbor to the right. Otherwise, the neighbor is the root of a subtree whose leftmost leaf nodes correspond to M's right-hand neighbors. These nodes are found by traversing that subtree.

Let M,N be black and white leaf nodes whose associated blocks are 4-adjacent. Thus the pair M,N defines a common border segment of length 2^K (2^K is the minimum of the side lengths of M and N) which ends at a corner of the smaller of the two blocks (they may both end at a common point). In order to produce a boundary code representation for a region in the image we must determine the next segment along the border whose previous segment lay between M and N. This is achieved by locating the other leaf P whose block touches the end of the segment between M and N:



If the M,N segment ends at a corner of both M and N, then we must find the other leaf R or leaves P,Q whose blocks touch that corner:



Again, this can be accomplished by using neighbor finding techniques as outlined in Section 2.

For the non-common corner case, the next border segment is the common border defined by M and P if P is white, or the common border defined 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 [13] for traversing region borders. This process is repeated until we re-encounter the block pair M,N. At this point the entire border has been traversed. The successive border segments constitute a 4-direction chain code, broken up into segments whose lengths are sums of powers of two. The time required for this process is on the order of the number of border nodes times the tree height. For more details see [4].

Using the methods described in the last two paragraphs, we can traverse the quadtree, find all borders, and generate their codes. During this process, we mark each border as we follow it, so that it will not be followed again from a different starting point. Note that the marking process is complicated by the fact that a node's block may be on many different borders.

In order to generate a quadtree from a set of 4-direction chain codes we use a two-step process. First, we trace the boundary in a clockwise direction and construct a quadtree whose black leaf nodes are of a size equal to the unit code length. All the black nodes correspond to blocks on the interior side of the boundary. All remaining nodes are left uncolored. Second, all uncolored nodes are set to black or white as appropriate. This is achieved by traversing the tree, and for each uncolored leaf node, examining its neighbors. The node is colored black unless any of its neighbors is white or is black with a border along the shared boundary. At any stage, merging occurs if the four rows of a non-leaf node are leaves having the same color. The details of the algorithm are given in [15]. The time required is proportional to the product of the perimeter (i.e., the 4-direction chain code length) and the tree height.

3.3 Quadtrees of Derived Sets

Let S be the set of 1's in a given binary array, and let $\bar{S}$ be the complement of S. The quadtree of $\bar{S}$ 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 and if the two nodes are gray, then their subtrees are traversed. 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, we copy the subtree of T at the corresponding 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 [23].

3.4 Skeletons and Medial Axis Transforms

The medial axis of a region is a subset of its points each of which has a distance from the complement of the region (using a suitably defined distance metric) which is a local maximum. The medial axis transform (MAT) consists of the set of medial axis or "skeleton" points and their associated distance values. The quadtree representation may be rendered even more compact by the use of a skeleton-like representation. Recall that a quadtree is a set of disjoint maximal square blocks having sides whose lengths are powers of 2. We define a quadtree skeleton to be a set of maximal square blocks having sides whose lengths are sums of powers of two. The maximum value (i.e., "chessboard") distance metric [13] is the most appropriate for an image represented by a quadtree. See [21] for the details of its computation for a quadtree; see also [24] for a different quadtree distance transform. A quadtree medial axis transform (QMAT) is a quadtree whose black nodes correspond to members of the quadtree skeleton while all remaining leaf nodes are white. See [22] for the details of a quadtree to QMAT conversion algorithm whose execution time is on the order of the number of nodes in the tree.

4. PROPERTY MEASUREMENT

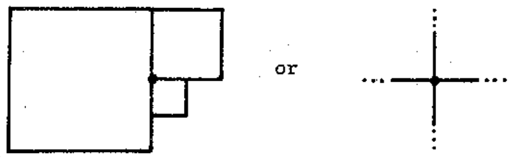
4.1 Connected Component Labeling

Traditionally, connected component labeling is achieved by scanning a binary array row by row from left to right and labeling adjacencies that are discovered to the right and downward. During this process equivalences will be generated. A subsequent pass merges these equivalences and updates the labels of the affected pixels. In the case of the quadtree representation we also scan the image in a sequential manner. However, the sequence's order is dictated by the tree structure - i.e., we traverse the tree in postorder. Whenever, a black leaf node is encountered all black nodes that are adjacent to its south and east sides are also visited and are labeled accordingly. Again, equivalences generated during this traversal are subsequently merged and a tree traversal is used to update the labels. The interesting result is that the algorithm's execution time is proportional to the number of blocks in the image and does not depend on their size. In contrast, for the binary array representation the execution time is proportional to the number of pixels. An analogous result is described in the next section. See [17] for the details of an algorithm that labels connected components in time on the order of the number of nodes in the tree plus the product of $B \cdot \log B$ where B is the number of black leaf nodes.

4.2 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 $\begin{matrix}1\\1\end{matrix}$'s, and F the number of $\begin{matrix}11\\11\end{matrix}$'s in the array; it is well known [13] that $g=V-E+F$. This result can be generalized to the case where the array is represented by a quadtree [3]. 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.



Then $g=V-E+F$. These adjacencies can be found (see Section 3.2) by traversing the tree; the time required is on the order of the number of nodes in the tree.

4.3 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 coordinates 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 the origin. The time required for any of these computations is proportional to the number of nodes in the tree. Further details on moment computation from quadtrees can be found in [23].

4.4 Perimeter

An obvious way of obtaining the perimeter of a region represented by a quadtree is to simply traverse its border and sum the number of steps. However, there is no need to traverse the border segments in order. Instead, we use a method which traverses the tree in postorder and for each black leaf node examines the colors of its neighbors on its four sides. For each white neighbor the length of the corresponding border segment is included in the perimeter. See [16] for the details of such an algorithm which has execution time proportional to the number of nodes in the tree.

5. CONCLUDING REMARKS

We have briefly sketched algorithms for accomplishing traditional region processing operations by use of the quadtree representation. Many of the methods used on the pixel level carry over to the quadtree domain (e.g., connected component labeling, genus, etc.). Because of its compactness, the quadtree permits faster execution of these operations. Often the quadtree algorithms require time proportional to the number of blocks in the image, independent of their size.

Quadtrees constitute an interesting alternative to the standard methods of digitally representing regions. Their chief disadvantage is that they are non shift-invariant; two regions differing only by a translation may have quite different quadtrees (but see [22]). Thus shape matching from quadtrees is not straightforward. Nevertheless, in other respects 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 requiring excessive storage for parts of the image where detail is missing.

References

1. N. Alexandridis and A. Klinger, Picture decomposition, tree data-structures, and identifying directional symmetries as node combinations, Computer Graphics Image Processing 8, 1978, 43-77.

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

3. C. R. Dyer, Computing the Euler number of an image from its quadtree, Computer Science TR-769, University of Maryland, College Park, MD, May 1979, to appear in CGIP.

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

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

6. 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.

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

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

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

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

11. A. Klinger and M. L. Rhodes, Organization and access of image data by areas. IEETPAMI-1, 1979, 50-60.

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

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

14. 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.

15. H. Samet, Region representation: quadtrees from boundary codes, Computer Science TR-741, University of Maryland, College Park, MD, March 1979, to appear in CACM.

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

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

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

19. H. Samet, Region representation: quadtrees from binary arrays, Computer Science TR-767, University of Maryland, College Park, MD, May 1979, to appear in CGIP.

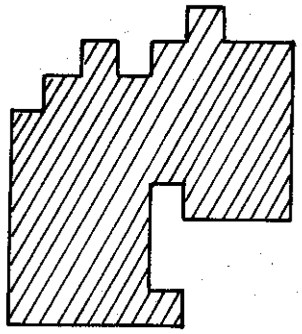
20. H. Samet, Region representation: quadtree-to-raster conversion, Computer Science TR-768, University of Maryland, College Park, MD, June 1979.

21. H. Samet, A distance transform for images represented by quadtrees, Computer Science TR-780, University of Maryland, College Park, MD, July 1979.

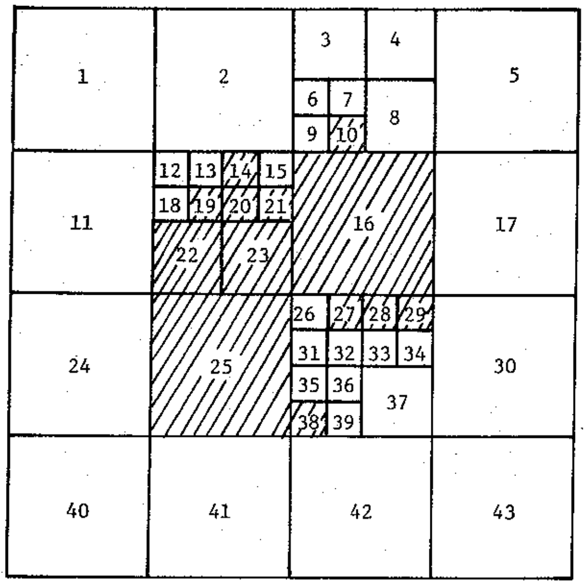
22. H. Samet, A quadtree medial axis transform, Computer Science TR-803, University of Maryland, College Park, MD, August 1979.

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

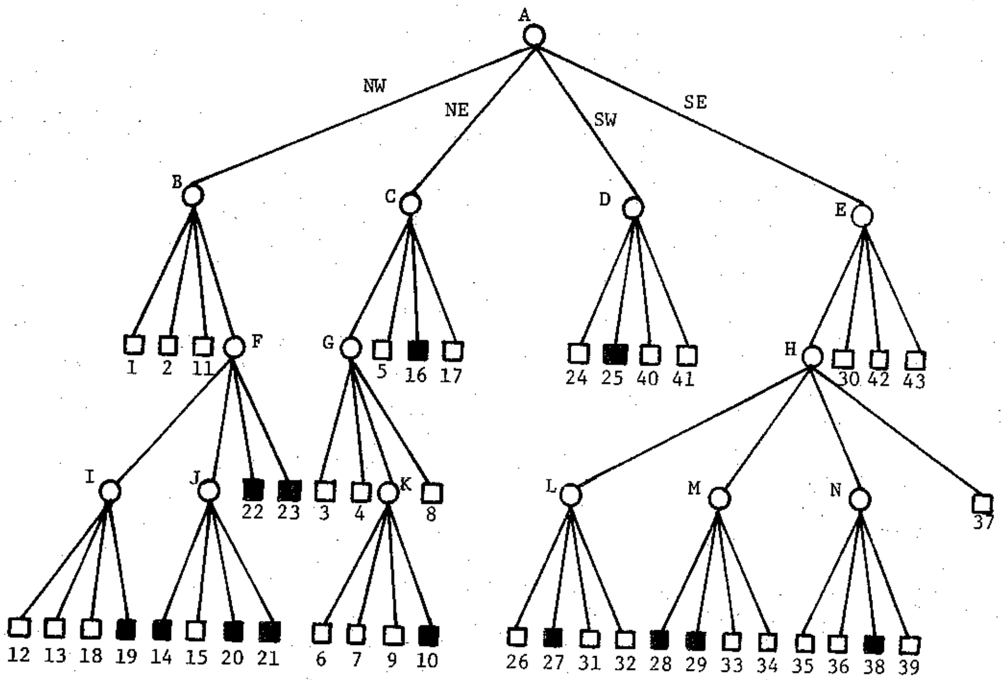
24. M. Shneier, A path-length distance transform for quadtrees, Computer Science TR-794, University of Maryland, College Park, MD, July, 1979.



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.