15. A Tutorial on Quadtree Research

H. Samet

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

Region representation is an important issue in image processing, cartography, and computer graphics. A wide number of representations is currently in use. Recently, there has been much interest in a hierarchical data structure termed the quadtree. It is compact and depending on the nature of the region saves space as well as time and also facilitates operations such as search. In this section we give a brief overview of the quadtree data structure and related research results.

15.1 Introduction

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 [15.1]. The binary array represents region pixels by 1s and nonregion pixels by 0s. The run length representation represents each row of the binary array as a sequence of runs of 1s alternating with runs of 0s.

Boundaries of regions are often specified as a sequence of unit vectors in the principal directions. This representation is termed a chain code [15.2]. 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 Fig.15.1a:

$$0^1 3^2 5^2 3^1 2 3^1 3^3 0^1 3^2 5^1 1^6 0101030101 \ .$$

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. For an alternative boundary representation, see the strip trees of BALLARD [15.3].

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 1s) of a given shape. The medial axis transform (MAT) [15.4,5] is the set of points serving as centers of these blocks and their corresponding radii.

The quadtree is a 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 1s or entirely of 0s, then we subdivide it into quadrants, subquadrants,...

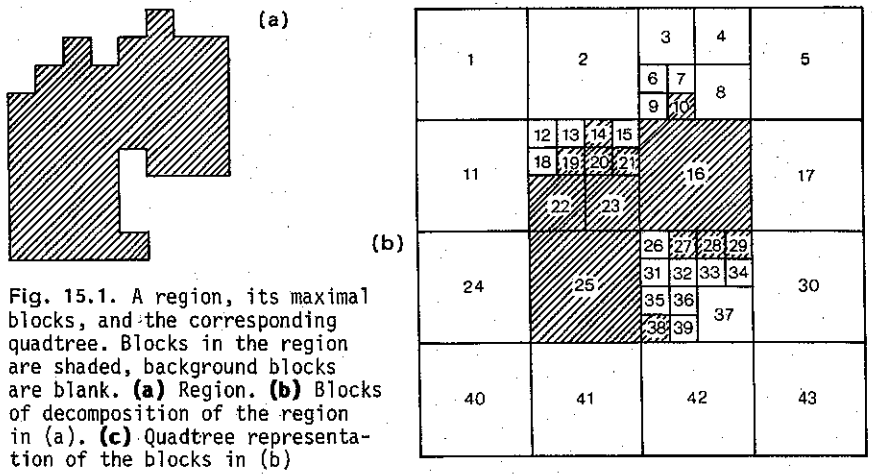
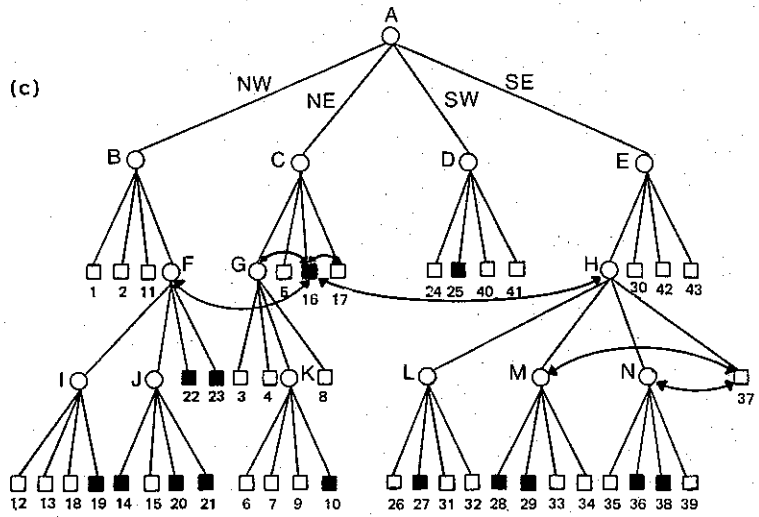


Fig. 15.1. A region, its maximal blocks, and the corresponding quadtree. Blocks in the region are shaded, background blocks are blank. **(a)** Region. **(b)** Blocks of decomposition of the region in (a). **(c)** Quadtree representation of the blocks in (b)



until we obtain blocks (possibly single pixels) that consist of 1s or of 0s, 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 (i.e., each nonleaf node has four sons) 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 nonleaf 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, Fig.15.1b is a block decomposition of the region in Fig.15.1a while Fig.15.1c is the corresponding quadtree. Each quadtree node is implemented, storage-wise, as a record with six fields. Five fields contain pointers to the four sons and the father of a node. The sixth field con-

tains type information such as color, etc. Note that the quadtree representation discussed here should not be confused with the quadtree representation of two-dimensional point space data introduced by FINKEL and BENTLEY [15.6] and also discussed in [15.7,8] and improved upon in [15.9].

The quadtree method of region representation is based on a regular decomposition. It has been employed in the domains of computer graphics, scene analysis, architectural design [15.10], and pattern recognition. In particular, WARNOCK's [15.11-13] algorithm for hidden surface elimination is based on such a principle--i.e., it successively subdivides the picture into smaller and smaller squares in the process of searching for areas to be displayed. Application of the quadtree to image representation was proposed by KLINGER [15.14] and further elaborated upon in [15.15-20]. It is relatively compact [15.15] and is well suited to operations such as union and intersection [15.21-23] and detecting various region properties [15.15,21,22,24]. HUNTER's Ph.D. thesis [15.21,22,24], in the domain of computer graphics, develops a variety of algorithms (including linear transformations) for the manipulation of a quadtree region representation. In [15.25-27] variations of the quadtree are applied in three dimensions to represent solid objects and in [15.28] to more dimensions.

There has been much work recently on the interchangeability between the quadtree and other traditional methods of region representation. Algorithms have been developed for converting a binary array to a quadtree [15.29], run lengths to a quadtree [15.30] and a quadtree to run lengths [15.31], as well as boundary codes to a quadtree [15.32] and a quadtree to boundary codes [15.33]. Work has also been done on computing geometric properties such as connected component labeling [15.34], perimeter [15.35], Euler number [15.36], areas and moments [15.23], as well as a distance transform [15.37,38]. In addition, the quadtree has been used in image processing applications such as shape approximation [15.39], edge enhancement [15.40], image segmentation [15.41], threshold selection [15.42], and smoothing [15.43].

15.2 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. For examples, see [15.30-38]. Frequently, these two steps are performed in parallel.

In general, it is preferable 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). Similarly, 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 nonleaf node to its sons and the link from a nonroot node to its father. Such techniques, described in [15.44], are used in [15.30-38] and result in algorithms that only make use of the existing structure of the tree. This is in contrast with the methods of KLINGER and RHODES [15.19] which make use of size and position information, and those of HUNTER and STEIGLITZ [15.21,22,24] 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 an 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 Fig.15.1c 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. Thus a "neighbor" is correctly defined as the smallest adjacent leaf whose corresponding block is of greater than or equal size. If no such node exists, then a gray node of equal size is returned. 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 Fig.15.1c is the NW neighbor of node 22. For more details, see [15.44].

In contrast with our neighbor-finding methods is the use of explicit links from a node to its adjacent neighbors in the horizontal and vertical directions reported in [15.21,22,24]. This is achieved through the use of adjacency trees, "ropes," and "nets." An adjacency tree exists whenever a leaf node, say X, has a GRAY neighbor, say Y, of equal size. In such a case, the adjacency tree of X is a binary tree rooted at Y whose nodes consist of all sons of Y (BLACK, WHITE, and GRAY) that are adjacent to X. For example, for node 16 in Fig.15.1, the western neighbor is GRAY node F with an adjacency tree as shown in Fig.15.2. A rope is a link between adjacent nodes of equal size at least one of which is a leaf node. For example, in Fig.15.1, there exists a rope between node 16 and nodes G, 17, H, and F. Similarly, there exists a rope between node 37 and nodes M and N; however, there does not exist a rope between node L and nodes M and N.

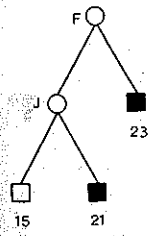


Fig. 15.2. Adjacency tree for the western neighbor of node 16 in Fig. 15.1.

The algorithm for finding a neighbor using a roped quadtree is quite simple. We want a neighbor, say Y, on a given side, say D, of a block, say X. If there is a rope from X on side D, then it leads to the desired neighbor. If no such rope exists, then the desired neighbor must be larger. In such a case, we ascend the tree until encountering a node having a rope on side D that leads to the desired neighbor. In effect, we have ascended the adjacency tree of Y. For example, to find the eastern neighbor of node 21

in Fig.15.1, we ascend through node J to node F, which has a rope along its eastern side leading to node 16.

At times it is not convenient to ascend nodes searching for ropes. A data structure named a net is used [15.21,22,24] to obviate this step by linking all leaf nodes to their neighbors regardless of their size. Thus in the previous example, there would be a direct link between nodes 21 and 16 along the eastern side of node 21. The advantage of ropes and nets is that the number of links that must be traversed is reduced. However, the disadvantage is that the storage requirements are considerably increased since many additional links are necessary. In contrast, our methods are implemented by algorithms that make use of the existing structure of the tree--i.e., four links from a nonleaf node to its sons, and a link from a nonroot node to its father.

15.3 Conversion

15.3.1 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 1s or all 0s may cause certain parts of the region to be examined repeatedly by virtue of being composed of a mixture of 1s and 0s. 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 0s or 1s 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 [15.29] for the details of such an 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 for the pixels and attempts are made to discover maximal blocks of 0s or 1s 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 [15.30] 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 0s or 1s 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 [15.31] 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.

15.3.2 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 in Fig.15.3, we use the neighbor-finding techniques outlined in Sect.15.2.2. 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.

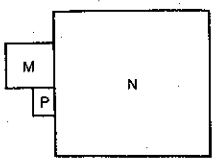


Fig. 15.3. Sample pair of blocks illustrating border following

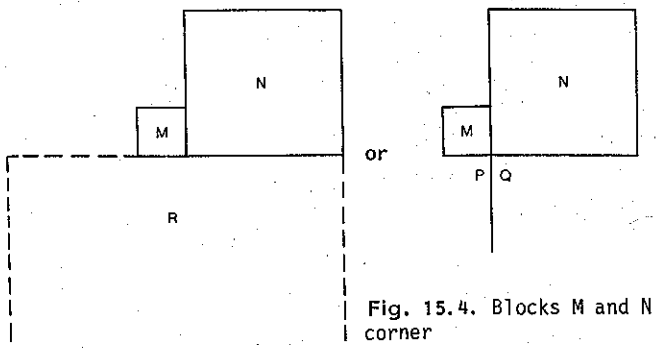


Fig. 15.4. Blocks M and N ending at a common corner

Let M,N in Fig.15.3 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 as in Fig.15.4). 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 (see Fig.15.4). Again, this can be accomplished by using neighbor-finding techniques as outlined in Sect.15.2.

For the noncommon 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 defin-

ing the next border segment is determined exactly as in the standard "crack following" algorithm [15.45] 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 [15.33].

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 sons of a nonleaf node are leaves having the same color. The details of the algorithm are given in [15.32]. The time required is proportional to the product of the perimeter (i.e., the 4-direction chain code length) and the tree height.

15.3.3 Quadtrees of Derived Sets

Let S be the set of 1s in a given binary array, and let $\overline{S}$ be the complement of S. The quadtree of the complement of S is the same as that of S, with black leaf nodes changed to white and vice versa. To get the quadtree of the union of S and 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 the intersection of S and 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 [15.23].

15.3.4 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 [15.45] is the most appropriate for an image represented by a quadtree. See [15.37] for the details of its computation

for a quadtree; see also [15.38] 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. The QMAT has several important properties. First, it results in a partition of the image into a set of possibly nondisjoint squares having sides whose lengths are sums of powers of two rather than, as is the case with quadtrees, a set of disjoint squares having sides of lengths which are powers of two. Second, the QMAT is more compact than the quadtree and has a decreased shift sensitivity. See [15.46] 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.

15.4 Property Measurement

15.4.1 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 pixels. An analogous result is described in the next section. See [15.34] 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 B log B, where B is the number of black leaf nodes.

15.4.2 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 1s, E the number of horizontally adjacent pairs of 1s (i.e., 11) and vertically adjacent pairs of 1s, and F the number of two-by-two arrays of 1s in the array; it is well known [15.45] that g=V-E+F.

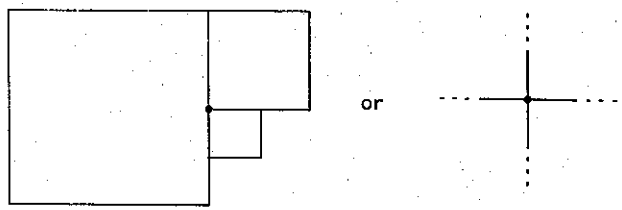


Fig. 15.5. Possible configurations of blocks that meet at and surround a common point

This result can be generalized to the case where the array is represented by a quadtree [15.36]. 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 (see Fig.15.5). Then g=V-E+F. These adjacencies can be found (see Sect.15.3.2) by traversing the tree; the time required is on the order of the number of nodes in the tree.

15.4.3 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 [5.23].

15.4.4 Perimeter

An obvious way of obtaining the perimeter of a region represented by a quadtree is simply to 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 [15.35] for the details of such an algorithm which has execution time proportional to the number of nodes in the tree. An even better formulation is reported in [15.47] which generalizes the concept of perimeter to n dimensions.

15.5 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.

The quadtree data structure requires storage for the various links. However, use of neighbor-finding techniques rather than ropes à la HUNTER [15.21, 22,24] is a compromise. In fact, experimental results show that the extra storage cost of ropes is not justified by the resulting minor decrease in execution time. This is because the average number of links traversed by neighbor finding methods is 3.5 in contrast with 1.5 for ropes. Nevertheless, there is a possibility that the quadtree may not be efficient spacewise. For example, a checkerboard-like region does not lead to economy of space. The space efficiency of the quadtree is analyzed in [15.48]. Some savings can be obtained by normalizing the quadtree [15.49,50] as is also possible by constructing a forest of quadtrees [15.51] to avoid large regions of WHITE. Storage can also be saved by using a locational code for all BLACK blocks [15.52]. Gray level quadtrees using a sequence of array codes to economize on storage are reported in [15.53].

The quadtree is especially useful for point-in-polygon operations as well as for query operations involving image overlays and set operations. Its hierarchical nature enables one to use image approximations. In particular, a breadth-first transmission of an image yields a successively finer image yet enabling the user to have a partial image. Thus the quadtree could be used in browsing through a large image database.

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 (but see [15.46]). 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. Their variable-resolution property is superior to trees based on a hexagonal decomposition [15.54] in that a square can be repeatedly decomposed into smaller squares (as can be done for triangles as well [15.55]), whereas once the smallest hexagon has been chosen it cannot be further decomposed into smaller hexagons. Note that the variance of resolution only applies to the area. For an application of the quadtree concept to borders, as well as area, see the line quadtree of [15.56].

References

15.1 D. Rutovitz: "Data structures for operations on digital images", in Pictorial Pattern Recognition, ed. by G. C. Cheng et al. (Thompson Book Co., Washington, DC, 1968), pp. 105-133

15.2 H. Freeman: Computer processing of line-drawing images, Computing Surveys 6, 57-97 (1974)

15.3 D. H. Ballard: Strip trees: a hierarchical representation for curves, Comm. ACM 24, 310-321 (1981)

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

15.5 J. L. Pfaltz, A. Rosenfeld: Computer representation of planar regions by their skeletons, Comm. ACM 10, 119-122 (1967)

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

15.7 H. Samet: Deletion in two-dimensional quad trees, Comm. ACM 23, 703-710 (1980)

15.8 D. T. Lee, C. K. Wong: Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, Acta Informatica 9, 23-29 (1977)

15.9 J. L. Bentley: Multidimensional binary search trees used for associative searching, Comm. ACM 18, 509-517 (1975)

15.10 C. M. Eastman: Representations for space planning, Comm. ACM 13, 242-250 (1970)

15.11 J. E. Warnock: "A Hidden Surface Algorithm for Computer Generated Half Tone Pictures", Computer Science Department TR 4-15, University of Utah (1969)

15.12 I. E. Sutherland, R. F. Sproull, R. A. Schumacker: A characterization of ten hidden-surface algorithms, Computing Surveys 6, 1-55 (1974)

15.13 W. M. Newman, R. F. Sproull: Principles of Interactive Computer Graphics, 2nd ed. (McGraw-Hill, New York, 1971)

15.14 A. Klinger: "Patterns and search statistics", in Optimizing Methods in Statistics, ed. by J. S. Rustagi (Academic Press, New York, 1972) pp. 303-339
15.15 A. Klinger, C. R. Dyer: Experiments in picture representation using regular decomposition, Computer Graphics Image Processing 5, 68-105 (1975)
15.16 S. L. Tanimoto, T. Pavlidis: A hierarchical data structure for image processing, Computer Graphics Image Processing 4, 104-119 (1976)
15.17 S. L. Tanimoto: Pictorial feature distortion in a pyramid, Computer Graphics Image Processing 5, 333-352 (1976)
15.18 E. M. Riseman, M. A. Arbib: Computational techniques in the visual segmentation of static scenes, Computer Graphics Image Processing 6, 221-276 (1976)
15.19 A. Klinger, M. L. Rhodes: Organization and access of image data by areas, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-1, 50-60 (1979)
15.20 N. Alexandridis, A. Klinger: Picture decomposition, tree data-structures, and identifying directional symmetries as node combinations, Computer Graphics Image Processing 8, 43-77 (1978)
15.21 G. M. Hunter: "Efficient Computation and Data Structures for Graphics", Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University (1978)
15.22 G. M. Hunter, K. Steiglitz: Operations on images using quad trees, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-1, 145-153 (1979)
15.23 M. Shneier: Calculations of geometric properties using quadtrees, Computer Graphics Image Processing 16, 296-302 (1981)
15.24 G. M. Hunter, K. Steiglitz: Linear transformation of pictures represented by quad trees, Computer Graphics Image Processing 10, 289-296 (1979)
15.25 D. R. Reddy, S. Rubin: "Representation of Three-Dimensional Objects", Department of Computer Science Technical Report 78-113, Carnegie-Mellon University (1978)
15.26 C. L. Jackins, S. L. Tanimoto: Oct-trees and their use in representing three-dimensional objects, Computer Graphics Image Processing 14, 249-270 (1980)
15.27 D. J. R. Meagher: "Octree Encoding: a New Technique for the Representation, Manipulation, and Display of Arbitrary 3-D Objects by Computer", TR 80-111, Rensselaer Polytechnic Institute (1980)
15.28 S. N. Srihari, M. Yau: "A Hierarchical Data Structure for Multidimensional Digital Images", Department of Computer Science Technical Report 185, State University of New York at Buffalo (1981)
15.29 H. Samet: Region representation: quadtrees from binary arrays, Computer Graphics Image Processing 13, 88-93 (1980)
15.30 H. Samet: An algorithm for converting rasters to quadtrees, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-3, 93-95 (1981)
15.31 H. Samet: "Algorithms for the Conversion of Quadtrees to Rasters", Computer Graphics Image Processing, to appear
15.32 H. Samet: Region representation: quadtrees from boundary codes, Comm. ACM 23, 163-170 (1980)
15.33 C. R. Dyer, A. Rosenfeld, H. Samet: Region representation: boundary codes from quadtrees, Comm. ACM 23, 171-179 (1980)
15.34 H. Samet: Connected component labeling using quadtrees, J. ACM 28, 487-501 (1981)
15.35 H. Samet: Computing perimeters of images represented by quadtrees, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-3, 683-687 (1981)
15.36 C. R. Dyer: Computing the Euler number of an image from its quadtree, Computer Graphics Image Processing 13, 270-276 (1980)

15.37 H. Samet: Distance transform for images represented by quadtrees, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-4, 298-303 (1982)
15.38 M. Shneier: Path-length distances for quadtrees, Information Sciences 23, 49-67 (1981)
15.39 S. Ranade, A. Rosenfeld, H. Samet: Shape approximation using quadtrees, Pattern Recognition 15, 31-40 (1982)
15.40 S. Ranade: Use of quadtrees for edge enhancement, IEEE Trans. Systems, Man, Cybernetics SMC-11, 370-373 (1981)
15.41 S. Ranade, A. Rosenfeld, J. M. S. Prewitt: "Use of Quadtrees for Image Segmentation", Computer Science TR-878, University of Maryland (1980)
15.42 A. Y. Wu, T. H. Hong, A. Rosenfeld: Threshold selection using quadtrees, IEEE Trans. Pattern Analysis Machine Intelligence PAMI-4, 90-94 (1982)
15.43 S. Ranade, M. Shneier: Using quadtrees to smooth images, IEEE Trans. Systems, Man, Cybernetics SMC-11, 373-376 (1981)
15.44 H. Samet: Neighbor finding techniques for images represented by quadtrees, Computer Graphics Image Processing 18, 37-57 (1982)
15.45 A. Rosenfeld, A. C. Kak: Digital Picture Processing (Academic Press, New York, 1976)
15.46 H. Samet: A quadtree medial axis transform, Comm. ACM, to appear.
15.47 C. Jackins, S. L. Tanimoto: "Quad-trees, Oct-trees, and K-trees: a Generalized Approach to Recursive Decomposition of Euclidean Space", IEEE Trans. Pattern Analysis Machine Intelligence, to appear.
15.48 C. R. Dyer: The space efficiency of quadtrees, Computer Graphics Image Processing 19, 335-348 (1982)
15.49 W. I. Grosky, R. Jain: "Optimal Quadtrees for Image Segments", IEEE Trans. Pattern Analysis Machine Intelligence PAMI-5, 1983, 77-83
15.50 M. Li, W. I. Grosky, R. Jain: Normalized quadtrees with respect to translations, Computer Graphics Image Processing 20, 72-81 (1982)
15.51 L. Jones, S. S. Iyengar: "Representation of regions as a forest of quadtrees", in Proc. Pattern Recognition and Image Processing Conf., Dallas, TX, 1981, pp. 57-59
15.52 I. Gargantini: An effective way to represent quadrees, Comm. ACM 25, 905-910 (1982)
15.53 E. Kawaguchi, T. Endo, J. Matsunaga: "DF-Expression Viewed from Digital Picture Processing", Department of Information Systems, Kyushu University (1982)
15.54 L. Gibson, D. Lucas: "Spatial data processing using generalized balanced ternary", in Proc. Pattern Recognition and Image Processing Conf., Los Vegas, NV, 1982, pp. 566-571
15.55 N. Ahuja: "Approaches to recursive image decomposition", in Proc. Pattern Recognition and Image Processing Conf., Dallas, TX, 1981, pp. 75-80
15.56 H. Samet, R. E. Webber: "On Encoding Boundaries with Quadtrees", Computer Science TR-1162, University of Maryland (1982)