U

Correspondence

Storing Feature Descriptions as 2-D Trees

THOMAS C. HENDERSON AND ERNST TRIENDL

Abstract—**Many methods have been proposed which produce low-level features from digital images, e.g., the raw primal sketch or intrinsic images. However, in some cases the features occur sparsely in the image, and a more efficient storage scheme can be used than a registered array of feature images. Edges constitute one of the most useful sorts of information for scene analysis. Even though edge responses usually occur sparsely throughout an image, the output from an edge detector in most image analysis systems is itself an image of the same dimensions (but possibly multichannel) as the original intensity image. Appreciable savings in space and time can be achieved if the full edge descriptions (orientation, radius, and likelihood information) are stored as a 2-D tree. This is a binary tree which uses the (x, y) locations of the pixels as keys and splits the data at the median along the key with greatest spread (i.e., this is a k-d tree for $k = 2$).**

I. Introduction

Feature extraction plays a prominent role in image analysis, and has always been important for the analysis of remotely sensed data. Features range from the intrinsic characteristics found in images (edges, reflectance, depth, etc.) to physical characteristics of a surface (temperature, smoothness, compressibility). Features are often used to characterize objects, and as time efficiency is crucial, features are usually chosen so as to provide an adequate description which is obtained cheaply and reliably. Feature extraction will be viewed as a distinct step performed on the raw sensor data, but obviously a "smart" sensor might provide such features directly.

In the context of digital image analysis, various schemes have been proposed for organizing properties or features recovered from 2-D images. For example, Marr proposed the primal sketch [7], Barrow and Tenenbaum investigated the intrinsic images of a scene [1], [2], and in a more limited context, Pavlidis has described the region adjacency graph [8].

Marr proposed the computation of a primitive but rich description of the gray level changes present in an image as the first important step for early visual information processing. Such a description is called the primal sketch. In this approach, the vision problem begins with a gray level intensity array. The primal sketch consists of a set of assertions, expressed in terms of a vocabulary of symbols and identifiers that are powerful enough to capture all of the important information in an intensity array, such as edge position and orientation.

Barrow and Tenenbaum suggested that an appropriate representation of early visual processing is a description of the scene in terms of intrinsic characteristics, such as range orientation, reflectance and incident illumination, of the surface element visible at each point in the image. They envision a set of cooperative processes operating on a registered set of intrinsic characteristic images.

Of all the low-level representations, the one that is the closest precursor to what we propose below is the region adjacency graph.

Manuscript received January 30, 1985; revised October 15, 1985.

T. C. Henderson is with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112.

E. Triendl is with DFVLR-Oberpfaffenhofen, 8031 Wessling, West Germany.

IEEE Log Number 8407045.

Each node in this graph corresponds to a region in the image, and an arc between two nodes indicates that they are neighbors in the image. The region adjacency graph has been used in various forms in many picture segmentation schemes.

However, most feature detectors produce very sparse results in terms of the class of images under study. The remainder of this paper discusses the use of the 2-D tree as a more efficient representation of such features. The following discussion addresses the problem of edge information, but it applies equally well to many other types of features. The major idea is that images may not be the most efficient data structure for feature storage and processing.

Various data compaction schemes have been proposed for images, one of the most important being the quad-tree [9]. A quad-tree is a successive subdivision of an image into quadrants, where a nonterminal node represents a nonuniform quadrant and leaf nodes represent a uniform quadrant at some level. This data structure is inappropriate for feature encoding since edges usually occur in thin strips throughout an image; moreover, quad-trees are most useful in conjunction with binary images.

II. k-d Trees

The k-d tree is a generalized form of the simple binary tree used to achieve order(nlogn) sorting and searching. Therefore, a k-d tree is a binary tree in which each node represents a subset of the vectors in a set of vectors and a partitioning of that subset [4]. The root of the tree represents the whole set of vectors (in our application, the vectors are (x, y) locations in an image where a feature was detected). Each nonterminal node has two successor nodes which represent the two subsets classified by the partition. The terminal nodes represent mutually exclusive small collections of the vectors in the set. These data vectors collectively form a partition of the set and are known as buckets. See Bentley [3] for the original definition of k-d trees. The version used here minimizes the expected number of vectors examined during the search for nearest neighbors. This is achieved by appropriately choosing both the discriminator key element and the partition value for each subset, and the number of vectors in each bucket.

Since information provided to a binary choice is maximal when the two alternatives are equally probable, it is equally likely that a vector will be placed on either side of the partition. Hence, irrespective of which key (i.e., which element of the vector) is selected as the discriminator, the median of the marginal distribution of key values serves well as the partition.

The search algorithm can stop searching the subset on the side of the partition opposite the query vector if the partition boundary does not intersect the ball centered at the query vector with radius equal to the dissimilarity to the mth closest vector so far encountered. Consequently, the partition will intersect least the ball for that key which showed the greatest range in values before partitioning.

With these considerations in mind, the optimized k-d tree algorithm chooses at every nonterminal node the key with the largest range in values as the discriminator, and the median of the discriminator key values as the partition. (Upon analysis of performance, the terminal buckets should each contain one record in order to minimize the number of vectors examined.) The average case complexity required to build a k-d tree is of the order(nlogn), and the m nearest neighbors for a vector query can be found in order(logn) operations.

As an example, consider the set of points shown in Fig. 1. The sequence of partitions is shown in Fig. 2(a), where the first parti-

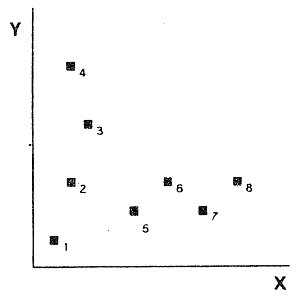

Fig. 1. A set of points in the plane.

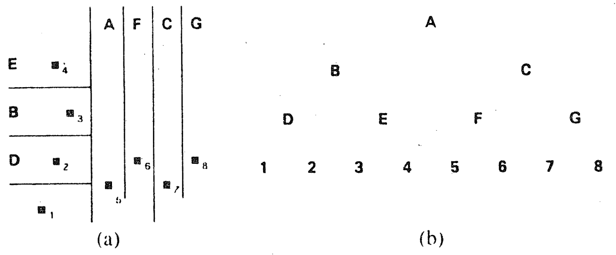

Fig. 2. (a) The partitions of the k-d tree. (b) The k-d tree.

tion is labeled A, the second B, etc.; the k-d tree (where $k = 2$) is shown in Fig. 2(b).

III. Edge Organization in Landsat Data

Edge information can serve as the basis for many purposes, including segmentation, shape analysis, and image registration. The edge-guided registration of Landsat images with other types of imagery and drawings motivates the approach described here. In order to achieve a subpixel registration accuracy, full edge descriptions of multichannel Landsat images are computed (see Henderson *et al.* [5], [6]). Edges are described at each pixel as a triple: (orientation, radius, edge likelihood).

The orientation and radius locate the edge with respect to the center of the pixel that serves as the origin, and the horizontal and vertical image axes serve as the x-axis and y-axis, respectively. To obtain values for the radius and orientation of the edge, we use a small window w centered at the pixel in question (usually a 3×3 or a 5×5). The gradient is used to get initial estimates for the radius and orientation. Using these estimates, intensities can be generated for the appearance of the edge in a similarly sized window s. These two windows, w and s, are then compared. An optimization technique is used to determine the best values for the radius and orientation parameters such that the synthesized edge appearance most closely matches the actual image window. See Triendl [10] for a description of this edge detector and Triendl and Henderson [11] for its application to texture images.

Thus, the output of the edge detector is a three-channel image with the same number of pixels in each channel as in the original; the three channels give the orientation, radius, and edge likelihood of the most likely edge at a given pixel. For a four-channel Landsat image, then, the output is a twelve-channel edge description image. Note that many image processing systems provide only the edge likelihood as the output image.

IV. Storage Comparisons

Let us consider now the advantages of storing the edge description as a 2-D tree. Suppose that the input image is a k-channel m by n image; let $p = mn$, and suppose there are ap significant edge responses (i.e., above some likelihood threshold), where a is in [0, 1], and finally, suppose that each edge description consists of e elements. Then in the standard case, the number of storage elements required is ekp. If the 2-D tree scheme is used, then there

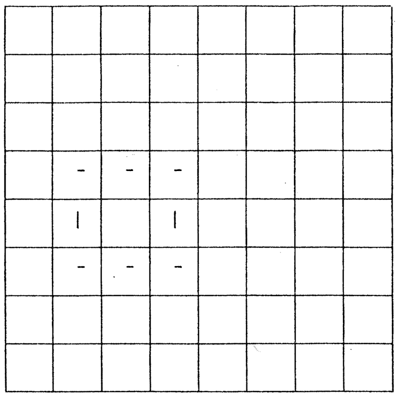

Fig. 3. An edge image.

are $(e + 2)apk$ storage elements required for each leaf of the tree, as the (x, y) location must be stored in addition to the e elements of edge description; also, there is a $(4apk)/b$ storage element overhead for the nonterminals in the tree, where b is the bucket size or number of records per leaf and assuming four storage elements per nonterminal. Thus, the 2-D tree is more economical whenever

$$a < e/\{(4/b) + e + 2\}.$$

For single-channel input image and edge descriptions, the breakeven point is around 14-percent edge density when the bucket size is 1; below this percentage of edge responses needing to be stored, the 2-D tree is more economical, while above it, the standard image representation requires less space. When the bucket size is 4, the breakeven point is at 25-percent edge density. In our application, i.e., edge data associated with a given control point, this efficiency can only become more important as the library of control points is increased.

For example, consider the image in Fig. 3. Then, we have that

$$k = 1, \quad p = 64, \quad a = 0.125, \quad e = 3, \quad \text{and} \quad b = 1.$$

Therefore, the storage required as an image is 192 elements, whereas the 2-D tree requires only 72 elements. If there were four channels of data, then the required storage would go up by four for each case; that is, an image would require 768 elements, while the 2-D tree would require only 288 elements.

V. Processing Comparisons

Typical operations on edge descriptions include channel grouping and spatial grouping. Channel grouping means that the edge responses across all the channels are averaged together in some way. In other words, for every pixel, the edge descriptions at that location from each channel are combined. Spatial grouping means that in each channel, the edge description at each pixel is modified by taking into account the edge descriptions at the neighboring pixels. This tends to produce smoother continuation of edges on boundaries.

A comparison of the cost of these grouping operations reveals that the 2-D tree representation offers time advantages, too. Consider first the time required to perform channel grouping.

In the usual case, i.e., multichannel edge description, the number of memory accesses required for channel grouping is kp since at each pixel the edge likelihood channel must be checked for each of the k output edge descriptions. However, if the edge descriptions are organized as a 2-D tree using the (x, y) location in the image as the two keys, then the tree will contain multiple entries with the same (x, y) keys if there are edge responses at the same pixel in more than one channel. Multiple responses can be stored as a linked list associated with a single entry for the (x, y) location. This means that channel grouping can be accomplished by simple tree traversal and the number of memory accesses is $2apk + (4apk)/b$ for terminals and nonterminals, respectively.

Thus, whenever

$$a < b/\{2(2 + b)\}$$

then the 2-D tree representation is more efficient. For bucket size of 1, this is around 17 percent of the edge responses, and at $b = 4$, the breakeven point is 33 percent.

Spatial grouping of edge responses requires computing a weighted average of edge responses occurring in some predefined neighborhood of a given pixel. In the standard array image representation, these neighbors can be found in constant time; thus, if spatial grouping is performed on the m neighborhood of a pixel, then m memory accesses must be performed for every pixel in the image, i.e., mp memory accesses. For the 2-D tree representation, the entire tree must be traversed, and for each leaf record, all neighbors within the prescribed distance must be found. This gives a number of memory accesses that is proportional to

$$\log (2ap)2ap + (4ap)/b.$$

Although no direct comparison is possible, there are two major reasons why the 2-D tree representation will be more efficient than the standard one. First, the 2-D tree only performs spatial grouping at pixels where an edge response occurred and then only accesses those locations in the prescribed neighborhood where edge responses occurred. Second, given the sizes of the representations, it is much more likely that the complete 2-D tree will fit in core memory, whereas with the complete array representation, there will be significant disk I/O overhead.

VI. Conclusion

The 2-D tree representation of feature images has been shown to be more efficient under certain conditions than the standard image array representation. In practice, we have found the 2-D tree to offer significant advantages. Obviously, the 2-D tree representation does not offer an immediate visual representation. However, in the case of edge descriptions, this is not a significant disadvantage in that an edge visualization image is computed from the multichannel edge description image anyway. Thus, this intermediate step applies equally well to the 2-D tree representation.

References

[1] H. G. Barrow and J. M. Tenenbaum, "Computational vision," in *Proc. IEEE*, vol. 69, no. 5, pp. 572-595, May 1981.

[2] —, "Recovering intrinsic scene characteristics from images," SRI International, Tech. Rep. 157, Apr. 1978.

[3] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *CACM*, vol. 18, no. 9, pp. 509-517, Sept. 1975.

[4] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Soft.*, vol. 3, no. 3, pp. 209-226, Sept. 1977.

[5] T. Henderson, E. Triendl, and R. Winter, "Edge- and shape-based geometric registration," *IEEE Trans. Geosci. Remote Sensing*, vol. GE-23, no. 3, pp. 334-342, May 1985.

[6] —, "Edge-based image registration," in *Proc. 2nd Scandinavian Conf. Image Analysis*, pp. 106-111, June 1981.

[7] D. Marr, "Early processing of visual information," *Phil. Trans. Roy. Soc. London*, vol. B. no. 275, pp. 483-524, 1976.

[8] T. Pavilidis, *Structural Pattern Recognition*. New York: Springer-Verlag, 1977.

[9] H. Samet, "Region representation: Quadtrees from boundary codes," *CACM*, vol. 23, no. 3, pp. 163-170, Mar. 1980.

[10] E. Triendl, "How to get the edge into the map," in *Proc. Int. Joint Conf. Pattern Recognition*, pp. 946-950, 1978.

[11] E. Triendl and T. Henderson, "A model for texture edges," in *Proc. Int. Conf. Pattern Recognition*, Dec. 1980.