LINEAR QUAD- AND OCT-TREES: THEIR USE IN GENERATING SIMPLE ALGORITHMS FOR IMAGE PROCESSING

Irene Gargantini* and Zale Tabakman
Department of Computer Science
The University of Western Ontario
LONDON, CANADA N6A 5B9

ABSTRACT

A linear quad-tree (oct-tree) is an efficient data structure used to represent all relevant properties of a quad-tree (oct-tree). It basically consists of a sorted array of quaternary (octal) integers, where each integer corresponds to a black node in the quad-tree (oct-tree) representation, and each digit corresponds to a quadrant subdivision. Linear quad-trees (oct-trees) are extremely efficient structures from the point of view of space requirements, saving from 75% (80%) up to 98% (99%) of the memory locations required by the regular quad-trees (oct-trees). In this paper we briefly mention all the algorithms so-far implemented at Western using linear quad-trees (oct-trees), we display their worst-case complexities and provide the reader with references to the papers in which a detailed description of the algorithms and the corresponding running-time estimates are given.

KEYWORDS: quadtrees, octtrees, linear quadtrees, linear octtrees, data structures, image processing, computer graphics

1. Introduction

The new data structure called 'linear quad-tree', recently developed at Western [2,3,4,5, 12], can be successfully used to generate simple and fast algorithms for image processing. This paper contains:

(i) a short introduction to linear quad-trees, their properties and space requirements;

(ii) a listing of the algorithms designed to perform operations on regions represented by linear quad-trees;

(iii) the extension of linear quad-trees to the three-dimensional case.

When compared with regular quad-trees [1,6,7,10, 11] a 'linear quad-tree' presents the following advantages:

(a) the information contained in the six fields of a quad-tree node is packed into one single field;

(b) only black nodes need to be stored;

(c) the encoding used for each node incorporates adjacency relations in the four principal directions as well as the path from the root to the node in the corresponding (unbuilt) quad-tree;

(d) operations like encoding, decoding, contour finding, rotation, superposition and detection of connectivity have lower worst-case time complexities than the corresponding algorithms using regular quadtrees.

* This work was partially supported by the Canadian Government through the Natural Sciences and Engineering Research Council, grant A7136.

2. Representation of a Simply Connected Region by Means of a Linear Quadtree

A simply connected region is usually given as a set of unit-picture elements (called 'pixels') in a $2^n \times 2^n$ - array arrangement, where n denotes the resolution parameter or degree of refinement of the screen. Fig. 1 gives an instance of a region which is simply connected while Fig. 2 gives an example of a region which is not.

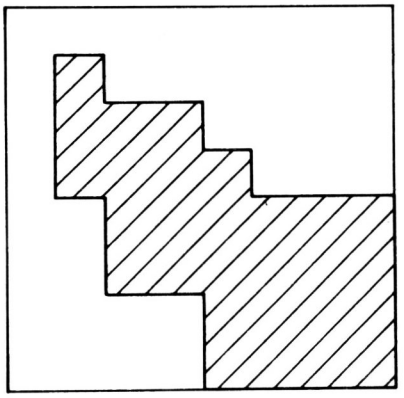

Fig. 1. A simply connected region.

A quad-tree is a data structure used to represent some of the properties of a given region in terms of 'large' groups of pixels instead than in terms of each single pixel as in the array form. A 'linear quad-tree' is a structure aiming at representing the same properties of a quad-tree in a more compact form. This idea can be at best illustrated by an example. Consider the region shown in Fig. 3 with n = 3 and the following encoding scheme: 0 for the North-West-quadrant, 1 for the North-East-quadrant, 2 for the South-West-quadrant and 3 for the South-East-quadrant. Use now a weighted quarternary code (with digits 0,1,2,3 to the base 4) to describe the successive quadrant subdivisions, starting from the largest. In the example of Fig. 3, the region is described by the following (sorted) sequence:

003, 021, 023, 030, 031, 032, 033, 122, 210, 211, 212, 213, 300, 301, 302, 303, 310, 311, 312, 313, 320, 321, 322, 323, 330, 331, 332, 333.

Condensation [2,3,12] is applied by introducing a special 'marker X', which must be encoded with an integer >3. We refer to this 'mixed encoding' as the 'mixed-quarternary'

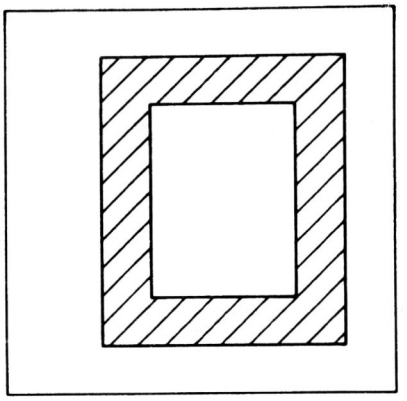

Fig. 2. The region shown is NOT simply connected.

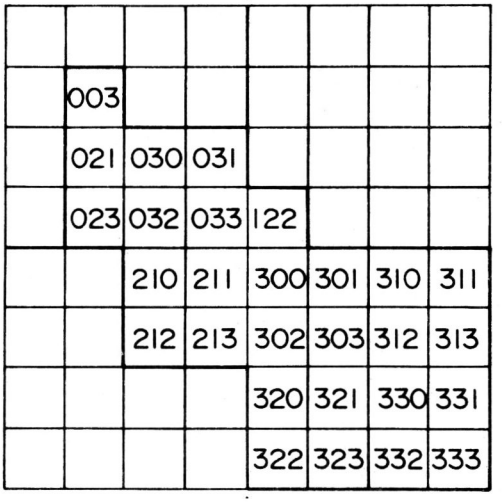

Fig. 3. Labelling pixels in the quarternary codes.

representation. The sequence becomes

033, 021, 023, 03X, 21X, 3XX.

The reader familiar with regular quad-trees can see that only black nodes have been stored and that the encoding of each node (from left to right) completely describes the path from the root to the node. Linear quad-trees have been shown to require, in the worst case, at most 25% of the memory locations needed by regular quad-trees and only 2% in the most favorable cases [3].

3. The Algorithms

Before describing briefly the algorithms, let us introduce some notation. Let n, as indicated before, represent the resolution parameter of the screen. Let I and J denote the row and column indices (respectively) of a pixel in the $2^n \times 2^n$ - array. Let NP denote the number of black pixels, N the number of black nodes, and L the perimeter of the region, defined as the number of unit-pixels sides forming the boundary of the region. When more than one region is involved, subscript 1 refers to the first region, subscript 2 to the second region, and so forth. The relations

$$N \leq NP < 2^{2n},$$

valid for all binary images, are often used to derive estimates of running times under the form

$$\log_2 N \leq \log_2 NP < 2n.$$

ALGORITHM ENCODING maps a pixel given in the (I,J)-form (I,J = 0,1,..., $2^n - 1$) into its quarternary code. ALGORITHM DECODING maps a pixel or a node into the (I,J)-form. Both algorithms perform their jobs in time proportional to n and can be applied to all pixels simultaneously. Both mappings, therefore, can be performed in parallel fashion (see, for instance, [2,5]).

ALGORITHM CONDENSE groups together pixels or nodes which belong to the same subquadrant. This algorithm can be executed in linear time [12].

ALGORITHM ADJACENCY (north to south, south to north, west to east and east to west) finds the node adjacent to a given one in a prescribed direction. This procedure evaluates quadrant transitions (if any) starting from the rightmost digit of a given node. This algorithm operates in time proportional to n [2,5].

ALGORITHM SEARCH establishes whether or not a quarternary code represents a black pixel: such a procedure is needed because we store only black nodes. Since the sequence of sorted encoded nodes can be stored in an array, searching can be done in logarithmic time with respect to the total number of black nodes [12].

ALGORITHM ROTATION-90 is a trivial procedure, based on the natural way we adopt to represent quadrants. A rotation of $\pm 90°$ (or a multiple thereof) can be accomplished by a parallel one-to-one digit conversion of black nodes according to quadrants transition [4,5].

ALGORITHM CONTOUR (incorporating ADJACENCY and SEARCH) accepts as input a linear quad-tree and a given node, and produces, as output, the corresponding chain code [12]. CONTOUR includes an initialization procedure which takes care of the pixels on the boundary of the screen and a termination criterion. Its time complexity is bounded by (n*L), where L is the perimeter of the region, as previously introduced.

ALGORITHM SUPERPOSITION distinguishes among the various cases occurring when the two given regions have:

(i) same pixel size and same screen center;

(ii) different pixel size but same screen center;

(iii) different screen center but same pixel size.

In cases (i) and (ii) the corresponding procedures are based on merging two sorted lists, and can, therefore, be carried out in linear time [2,5]; case (iii) requires a suitable translation of the center as explained in [5]. Intersection can be designed in a similar fashion.

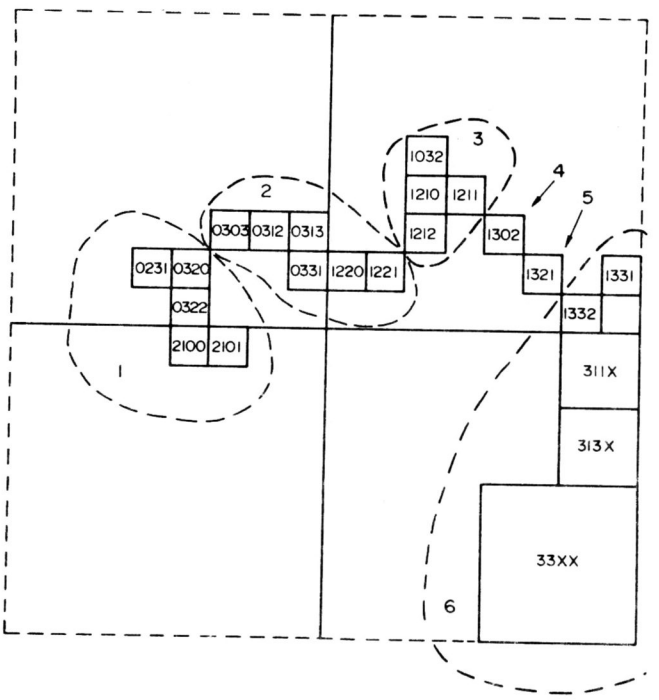

Fig. 4. Detecting connected regions

ALGORITHM CONNECTIVITY is designed to identify k ≥ 1 simply connected regions [3,12]. CONNECTIVITY accepts as input a linear quad-tree and outputs k linear quad-trees, each representing a simply connected region. For instance, in the case of the region shown in Fig. 4, six linear quad-trees would be identified, corresponding to the regions delimited by dotted lines or represented by single pixels. The algorithm can be executed in time proportional to (n*N), where N is the number of black nodes of the input linear quad-tree.

The worst-case complexities for the above algorithms are given in Table 1.

Operation	Worst-case time complexity	
	pixel	region
Encoding	O(n)	O(n*NP)
Decoding	O(n)	O(n*N)
Condense		O(NP)
Adjacency	O(n)	
Search		O(n)
Rotation	O(n)	O(n*N)
Contour		O(n*L)
Connectivity		O(n*N)
Superposition cases (i), (ii)	two regions	$O(N_1 + N_2)$
Superposition case (iii)		$O(n*(NP_1 + NP_2))$

Table 1. Time complexities

4. Linear Oct-trees

A much greater saving in terms of space is shown to occur when we extend our new structure to three dimensions. We can reduce the ten or eleven fields required by regular oct-trees as described in [8] to only one. We represent each voxel in a weighted octal system (with digits 0,1,2,..., 7 to the base 8). Condensation is still represented by marker X which now must be encoded with an integer >8.

Algorithms for encoding voxel into their octal representation, decoding nodes, finding adjacent nodes, searching for a black voxel, rotating a three-dimensional object and superposing two objects have been designed [4] according to a scheme very similar to the one adopted for the two-dimensional case. In three dimensions another interesting problem arises, consisting of projecting an object onto the principal planes. With linear oct-trees this problem can be easily solved as shown in [4]. Some experimental programs designed to test the above algorithms can be found in [9].

5. Concluding Remarks

Both linear quad- and oct-trees are very efficient structures from the point of view of space requirements. The algorithms for image manipulation also compare favourably, on the average, with those formulated for regular quad- and oct-trees. One of the main characteristics of quad-trees which we did not discuss in this paper is the dynamic capability a quadtree has, i.e., its ability to expand through its terminal nodes ONLY when n, resolution parameter, is incremented. A simulation of this capability is discussed in [2] where we show how all white nodes can be generated DIRECTLY from a linear quad-tree.

Other interesting topics to be investigated are: the formulation of an efficient algorithm to determine the boundary of a three-dimensional object and the reconstruction of a linear quad-tree (or oct-tree) from the chain code. We hope to address these questions in a forthcoming paper.

A software package for linear quad-trees has been developed as part of the fourth-year thesis of the second author [12] and is available on request. At the present time it is running on the DEC system-10 and is written in the PASCAL language. Future plans include the development of its optimized version for the VAX 11/780 system of the Department of Computer Science.

References

1. Dyer, C. R., Rosenfeld, A., and Samet, H. Region representations: boundary codes from quadtrees. Comm. ACM 23 (1980), 171-179.

2. Gargantini, I. An efficient way to represent properties of quadtrees. Conditionally accepted by Comm. ACM.

3. Gargantini, I. Detection of connectivity for regions represented by linear quadtrees. To appear in Comput. Math. Applics.

4. Gargantini, I. Linear oct-trees for fast processing of three-dimensional objects. To appear in Comptr. Graph. and Image Processing.

5. Gargantini, I. Translation, rotation and superposition of linear quad-trees.

Submitted to the Inter. Journal of Man-Machine Studies on Jan. 26, 1982.

6. Hunter, G. M. and Steiglitz, K. Operations on images using quadtrees. IEEE Trans. on Pattern Analysis and Machine Intell. 1 (1979), 145-153.

7. Hunter, G. M. and Steiglitz, K. Linear transformations of pictures represented by quadtrees. Comptr. Graphics and Image Processing 10 (1979), 289-296.

8. Jackins, C. L. and Tanimoto, S. L. Oct-trees and their use in representing three-dimensional objects. Comptr. Graphics and Image Processing 14 (1980), 249-270.

9. Lam, G. Linear oct-trees for image processing. CS490y Thesis (1982), Computer Science Department, The University of Western Ontario, London, Canada N6A 5B9.

10. Samet, H. An algorithm for converting rasters to quadtrees. IEEE Trans. on Pattern Analysis and Machine Intell. 3 (1981), 93-95.

11. Samet, H. Region representation: quad-trees from boundary codes. Comm. ACM 23 (March 1980), 163-170.

12. Tabakman, Z. A software package for linear quad-trees. CS490y Thesis (1982), Computer Science Department, The University of Western Ontario, London, Canada N6A 5B9.