A RECURSIVE CHAIN CODE TO QUADTREE CONVERTING METHOD WITH A LOOKUP TABLE

Zen Chen and I-Pin Chen
Department of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C
E-mail address: zchen@csie.nctu.edu.tw

ABSTRACT

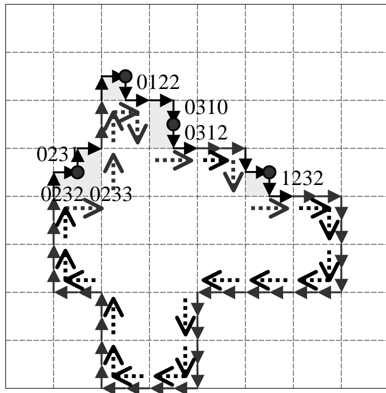
We present a simple recursive method for converting a chain code into a quadtree representation. We generate the quadtree black nodes recursively from the finest resolution level to the coarsest resolution level. Meanwhile, at each resolution level a new object border is unveiled after the removal of the black nodes. The chain code elements for this new object border can then be easily generated. Thus, the generation of the quadtree black nodes at one level and the generation of the chain code elements of the new object border both constitute a basic cycle of the conversion process. We also show the generations can be done with the aid of a table lookup. Finally, our method is shown to be better than the well-known Samet's method in terms of the total numbers of major operations including node allocations, node color filling, and node pointer linking.

1. INTRODUCTION

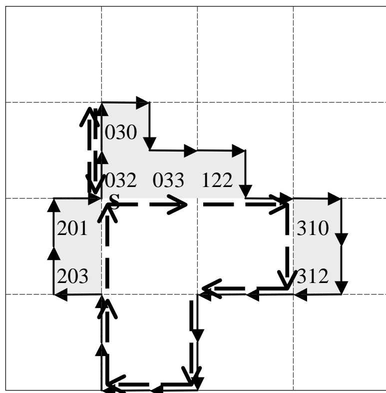
To present the chain code to quadtree conversion, let us assume that the region (i. e., the grid cells) on the right-hand side of a code element is inside the object. Also, assume that the chain code is not self-intersecting. The conversion is basically a color-filling operation, that is, coloring black on those nodes inside the object. There are four popular existing methods : Samet [1], Mark and Abel [2], Webber [3], and Lattanzi and Shaffer [4]. These methods are all a two-phase method. The major drawback in these two-phase method is the need of merging of smaller nodes into larger nodes. This is due to the lack of some mechanisms for judging whether the border grid cells are legal quadtree black nodes? It is not difficult to figure out in Fig.1(a) that the parent node containing the border grid cells 0231, 0232, 0233 is passed through by the chain code during the chain code tracing, so the three cells can be outputted as quadtree black nodes. However, it is not so obvious to assess that the border grid cell other than those shown in Fig. 1(a) are contained in a larger quadtree black node, so neither of them is qualified as a quadtree black nodes.

We shall present a simple method to determine on the fly which border grid cells should be outputted as legal quadtree black nodes, when we tracing the chain. The removal of these black nodes yields a new object border. This new object border is represented by a new chain code with the length-two code elements. Generation of quadtree black nodes at one level and generation of the chain code elements of the new object border constitute a basic cycle of a conversion process. The cycle of process is repeated until that the quadtree black nodes at all levels are generated. Figs. 1(a)-1(c) illustrate the result of this recursive generation process.

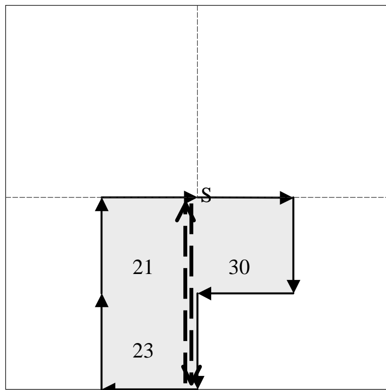
2. RECURSIVE GENERATIONS OF QUADTREE BLACK NODES AND NEW CHAIN CODE ELEMENTS



(a)



(b)



(c)

Figure 1. (a) An object and its chain code CC_4. The shaded area is the pixel level black nodes of QN_4. The center points in the parent nodes are marked as solid dots. The dash-line arrows indicate the new code elements generated. (b) The chain code CC_3 and the quadtree nodes QN_3. (c) The chain code CC_2 and the quadtree nodes QN_2.

Let QN_i, i = N, N-1,...,0, be the set of quadtree "black" nodes at the i-th level. Let CC_i denote the chain code of the object at the i-th level. Here, we will classify the grid points at the current resolution level into three types, depending on whether it is a grid point at the next coarser level, or the center point of the parent node, or a midpoint of the grid line of the parent node. We generate all levels of black nodes, QN_i, i=N, N-1,...,0 in a bottom up fashion. In our method after the black nodes are generated at each level, the other nodes at the same level are automatically classified as white nodes.

We generate the quadtree black nodes and the new chain code element side by side. We have observed that if parent node is cut through by the chain code, then the black son nodes are legal quadtree black nodes. Also the new code elements associated with the new object border after removal of black nodes can be determined right away. Furthermore, we find that only the four nodes belonging to a common parent are needed in deciding whether they are the legal quadtree black nodes or not? Therefore, it will be shown that each time only four consecutive code elements are required in a basic cycle of the conversion process.

On the other hand, the starting point of the chain code better be a grid point at the parent level, after the removal of all black nodes found at the current level because the chain code at the next coarser level is defined over such grid points. There are two lemmas concerning the validation of quadtree black nodes during the chain code scanning (refer to Fig. 2).

Lemma 1: If and only if any code element of CC_i enters the center point of a parent node, then there will be one to three black nodes of QN_i at

the i-th level. The exact number of black nodes depends on the actual pattern of the succeeding code elements.

Lemma 2: If two consecutive code elements of CC_i are in the same direction and lie on the border of a common parent node, then there will be two to four black grid cells of QN_i at the i-th level. The exact number of quadtree black nodes depends on the actual pattern of the succeeding code elements.

So the two code elements are replaced with a code element of a double length in the same direction to be used at the next coarser level.

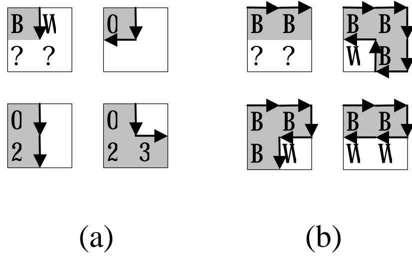


Figure 2. Illustrations for Lemma 1 (a) and Lemma 2 (b).

When the above black nodes of QN_i are outputted, then a new object border is exposed which can be represented by some (or none when finished) new code element(s) at the next coarser level, each with a double length. It should be noticed that the new chain code may contain pairs of code elements in opposite directions and these pairs of elements cancel out by themselves (see examples in Figs. 1(b) and (c)). So we need a post-processing stage to remove such code element to make sure that the processed chain code will satisfy our assumption of no self-intersection.

At the initial stage of the conversion, if the starting point of the chain code of CC_i at the current level is not a grid point of its parent node, then it is adjusted a new starting point in the way as shown in Fig.3. Once we choose a grid point as the starting point for scanning the chain code, we can fetch four succeeding code elements to analyze.

Now consider a typical code element pattern. Fig. 4 shows all the 19 possible combinations of at most four code elements, assuming a starting code element (marked by s) is in direction 1. Fig. 5 shows the generated black nodes (shown by shaded areas), the generated code elements at the next coarser level (shown by long arrows), and the possible adjusted code elements at the current level (shown by short arrows). They are collected in a lookup table in Table 1. Therefore, we can design a table lookup to generate the black quadtree nodes and the new chain code elements.

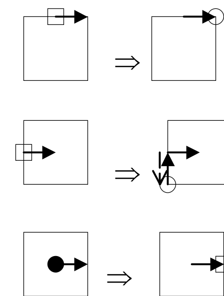


Figure 3. The different adjustments for the starting point.

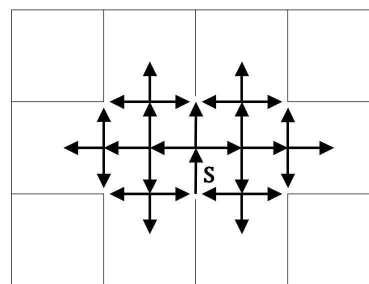


Figure 4. The 19 possible patterns of four consecutive code elements whose starting code element (labeled by S) is in the upward direction.

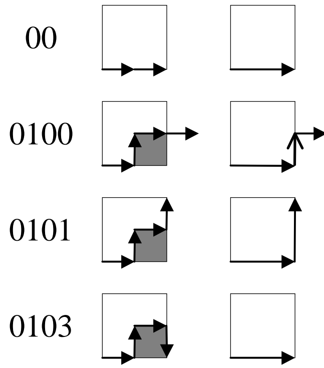


Figure 5. Four of the 19 possible code patterns beginning with 0, together with the corresponding quadtree black nodes (the shaded region(s) at left), and the new code elements at the next coarser level (the long arrow(s) at right).

3. PERFORMANCE COMPARISON

Basically, both the Samet's method and our method use the color filling operations to generate the quadtree from the chain code. However, these two methods are different in the way of producing the true black nodes in the quadtree. The following lemmas regarding the performance can be obtained.

Table 1. The lookup table. The symbol "X" denotes any value in the set (0,1,2,3) and the symbol "-" denotes no output.

Input code pattern of CC_i	Output Code Pattern of CC_{I-1}	Remaining code elements of CC_i	Black quadtree nodes of QN_i
00XX	0	XX	-
0100	0	10	3
0101	01	-	3
0102			
0103	0	-	3
….	….	….	….
33XX	3	XX	-

Lemma 3: The total number of nodes expanded (or generated) by the Samet's method is larger than that of the regular quadtree by a number that is equal to the total number of nodes that are deleted due to any node merges taking place. The total number of nodes expanded by our method is equal to that of the regular quadtree; in fact, only the black nodes are generated and the white nodes are added by default.

Lemma 4: The total number of color filling operations in the Samet's method is larger than that of our method by a number that is equal to the total number of nodes that are merged multiplied by 1.25. (i.e., the total number of black nodes deleted plus their re-colored parent nodes)

Lemma 5: The total number of pointer links constructed in the Samet's method is larger than that of our method by a number that is equal to the total number of nodes that are merged multiplied by 3. (i.e., the total number of pointer links constructed between the parent nodes and their child nodes before and after the node merges)

Fig. 6 shows three representative object images in which our method is overall better than the Samet's method in terms of the total numbers of major operations including node allocations, node color filling, and node pointer linking (refer to Table 2). The overhead of our method is the need of generating the chain code for each level. We generated the new chain code using the look-up table, so the processing is fast.

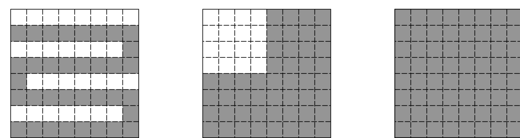


Figure 6. Three object images used for performance comparison.

4. CONCLUSION

In this paper we have presented a simple recursive method for converting a chain code representation into a quadtree one. Lemmas for the determination of the quadtree black nodes are stated. The generation of the quadtree black nodes and the generation of the chain code for the new object border constitute a basic cycle of the conversion process. We show the generation can be done with a table lookup. Generally speaking, our conversion method is better than the Samet's method in terms of the total numbers of major operations including node allocations, node color filling, and node pointer linking.

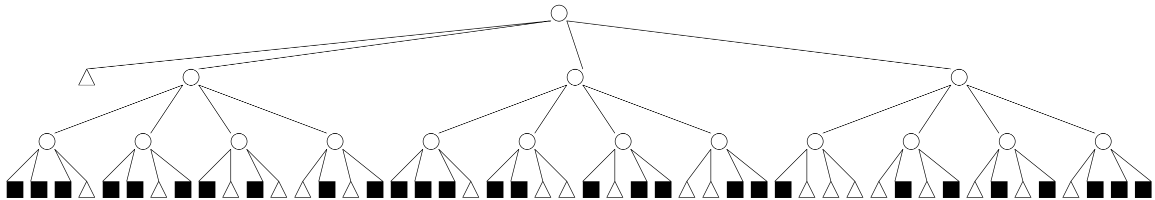
5. REFERENCES:

[1] H. Samet, "Region representation: quadtree from chain codes," Communications of ACM 23, pp. 163-170, 1980.

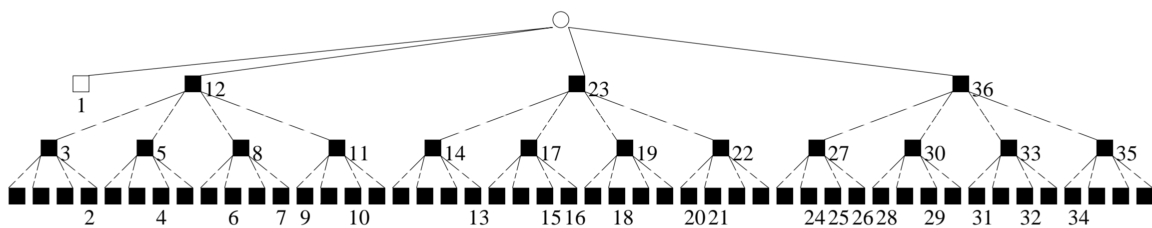
[2] D. M. Mark and D. J. Abel, "Linear quadtrees from vector representations of polygons," IEEE Transactions on Pattern Analysis and Machine Intelligence 7, pp. 344-349, 1985.

[3] R.E. Webber, "Analysis of Quadtree Algorithms," Ph. D. dissertation, TR-1376, Computer Science Department, University of Maryland, College Park, MD, 1984.

[4] M. R. Lattanzi and C. A. Shaffer, "An optimal boundary to quadtree conversion algorithm," CVGIP: Image Understanding 53, pp. 303-312, 1991.



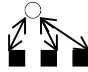
(a) The tentative quadtree obtained after phase one of the Samet's method.



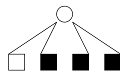
(b) The final quadtree obtained by Phase 2 of the Samet's method. Those nodes at the bottom two levels are deleted after node merging.

(Empty tree)

(c) The generation results obtained after passes 1 and 2 of our method; the intermediate quadtree is empty, containing no black nodes.



(d) The generation results obtained after pass 3 of our method



(e) The final quadtree obtained after the inclusion of the default white node.

Fig. 7 The quadtree generation for Object 2 by the two methods. (a)-(b) The generation results obtained by Phase 1 and Phase 2 of the Samet's method. (c)-(e) The generation results obtained by our method.

Table2 The comparison of the numbers of operations used by the Samet's method and our method.

Method	Object / Number of operations	Object 1	Object 2	Object 3
Samet's method	Node allocation	85	65	69
	Node coloring	85	80	86
	Pointer writing	168	188	204
	Node deallocation	0	60	68
Our method*	Node allocation	85	5	1
	Node coloring	85	5	1
	Pointer writing	168	8	0
	Node deallocation	-	-	-

*Refer to Lemmas 3 to 5.