Efficient Coding of Quadtree Nodes

Mariano Pérez, Xaro Benavent, and R. Olanda

Instituto de Robótica. University of Valencia
Polígono de la Coma, s/n.
Aptdo. 22085, 46071-Paterna, Spain
Mariano.Perez@uv.es

Abstract. In this paper an alternative non-pointer quadtree node codification to manage geographical spatial data is presented. New codification is based on a variable sequence of z-ordered base four digits. Memory requirements of the new codification are lower than previous codifications, and in particular lower than FD codification, the most commonly used in linear quadtrees. Furthermore, z-ordering makes compatible new codification with most of the algorithms developed for FD.

1 Introduction

In recent years, a wide set of data structures has been developed to manage spatial data in Geographic Information Systems (GIS) field. The most popular approaches are based on quadtrees, bintrees, R-trees, the cell tree and the grid file [3]. The quadtree is probably one of the most extended for spatial information representation, and amongst the different kinds of quadtrees, the region quadtree is the most used [1].

Region quadtrees can be represented in memory as a dynamic tree (using pointers) or they can be stored using a set of linear codes (without using pointers). Pointer-based approach is ill-suited for implementing disk-based structures whereas linear codes approach is especially interesting when the quadtree size is quite large or when it is stored on secondary storage devices. Memory requirements in the case of linear codes is lower than when using pointers, but it is possible to regenerate the whole structure from only a few leaf nodes, as it happens for linear quadtrees [1].

2 Previous Quadtree Codifications

There are several ways of implementing linear codes that are able to identify the quadtree nodes. Usually the codification employs numbers in base four or five, called locational codes. The digits of these numbers are directional codes which locates the node within the quadtree from the root node. Depending on the implementation, apart from the location code, the node level (depth of the leaf node within the quadtree) is also stored. The most popular linear implementations are the FL (Fixed Length), the FD (Fixed Depth) and the VL (Variable Length) [3].

The FL and VL locational codes are based on a base five sequence of digits. Each digit indicates NW, NE, SW, SE and "don't care" directions. These codifications do not need an additional field to codify the node level. This information is implicit in the directional code; the "don't care" value indicates the node is a leaf. The main difference between VL and FL is that the VL locational codes have a variable length and it is usually shorter than the FL locational one.

The FD codification is usually implemented as a structure with two fixed size fields. The first field, called quadcode [2], stores the directional code and indicates the position and orientation of the node within the quadtree. The second field indicates the node level.

The quadcode is a z-ordered base four code and consists of a sequence of values:

$$q = q_1 q_2 ... q_N \tag{1}$$

where $q_i = 0, 1, 2, 3$, with $i = 1, 2, .., N$, being N the code length (which is the maximum number of tree levels). For each square block in which a quadrant is divided, a new value q_i is added to have a unique characterization of each node.

Among the three indicated codifications, the FD is the most referenced in scientific literature because the FD locational code (quadcode) is a base four code, and it is easily handled using two bits per digit, which allows the use of binary and logical operations to carry out the codification and de-codification processes. Furthermore, FD codification is the most used in linear region quadtrees [4].

3 Base 4 Variable Length Code (B4VL)

B4VL is an improved alternative version of the FD codification, where the locational code and the node level are integrated in a unique code, and where locational code has variable size. These properties are not present in the FD codification.

To carry out the packing process in B4VL, the bits that define the code are divided into two parts. The first part, corresponding to low bits, has a fixed size and is used to store the node level. The other part has a variable length, and stores the locational code (Figure 1).

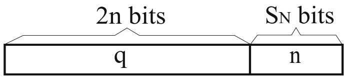

Fig. 1. Bits distribution in B4VL codification. First part correspond to node level (n). Second part correspond to locational code (q).

The B4VL locational code is similar to quadcode (equation 1), but its size is optimized. If n is the node level, the locational code is a length n sequence of values:

$$q = q_1 q_2 ... q_n \tag{2}$$

where $q_i = 0, 1, 2, 3$, being $i = 1, 2, .., n$.

If the maximum node level in the quadtree is N, the number of bits used to codify the first part of the node code is $\lceil \log_2(N+1) \rceil$, while the second part needs $2n$ bits, being n the node level. The length of the first part is independent of the node level, so it is only computed once and this value is stored in memory. The number of bits of the second part is variable and equal to twice the node level (see Figure 1).

The algorithm that generates the code from the node level (n) and the locational code (q) is based on the following equation:

$$code \leftarrow ((q \ll S_N) + n) \tag{3}$$

where $\ll$ is the left bitwise binary operator, and S_N is the number of bits in the first part of the code. This number is the same for every node in the quadtree and its value is $\lceil \log_2(N+1) \rceil$, where N is the maximum node level in the quadtree.

The inverse operation is based on the following equations:

$$\begin{cases} n \leftarrow (code \& M_S) \\ q \leftarrow (code \gg S_N) \end{cases} \tag{4}$$

where & is the AND binary operator, and $\gg$ is the right bitwise operator; M_S represents a binary mask computed once using the equation: $M_S = 2^{S_N} - 1$.

Both operations (equations 3 and 4) have an optimal computational cost, which is constant ($O(1)$) with node level.

4 Results

As we have indicated above, FD and FL codes have a fixed size, while VL and the proposed B4VL codification have a variable size depending on the depth of node. Table 1 shows the storage cost for the four codifications. The first part of the summation for the B4VL and the FD codifications is related to the number of bits used to store the node level and the second part to the quadcode.

Table 1. The number of bits used in B4VL, FD, FL and VL codifications based on node level n (depth) and based on the maximum achieved level of the quadtree N

B4VL	$E(n) = \lceil \log_2(N+1) \rceil + 2n$
FD	$E(n) = \lceil \log_2(N+1) \rceil + 2N$
FL	$E(n) = \lceil N \cdot \log_2 5 \rceil$
VL	$E(n) = \lceil (n+1) \cdot \log_2 5 \rceil$ (if $n < N$) **OR** $E(n) = \lceil (N) \cdot \log_2 5 \rceil$ (if $n = N$)

In order to compare the memory space required for each kind of codification, the average number of bits needed to codify the nodes included in fully occupied quadtrees will be considered (quatrees are selected based on its maximum level N). Results are shown in Table 2.

Table 2 highlights that B4VL has variable memory size, but it is always lower than the space required by FD. It also shows that B4VL size codes are higher than FL and VL for trees of small height, but they are much lower for deeper trees ($N \geq 7$).

Table 2. Average node bits for each codification based on the maximum quadtree node level N

N	1	2	3	4	5	7	10	13	17	20	25	30	40
B4VL	3.00	5.60	7.43	10.36	12.34	16.33	23.33	29.33	38.33	44.33	54.33	64.33	85.33
FD	3.00	6.00	8.00	11.00	13.00	17.00	24.00	30.00	39.00	45.00	55.00	65.00	86.00
FL	3.00	5.00	7.00	10.00	12.00	17.00	24.00	31.00	40.00	47.00	59.00	70.00	93.00
VL	3.00	5.00	6.91	9.80	11.83	16.77	23.77	30.77	39.82	46.82	58.77	69.83	92.83

5 Conclusions

In this paper a new approach to codify nodes in quadtree representation has been presented. The approach is based on a unique compact base four representation of the locational code and the level of the node.

It has been demonstrated that the storage cost of the proposed codification is better than previous codifications (table 2). Space memory requirements for B4VL codification are always lower than the memory needed for FD codification. Compared to FL and VL codifications, B4VL is similar for small height quadtrees and lower for deeper quadtrees, but B4VL uses a base four digits code, so its coding and decoding process is much more simple and efficient than FL and VL.

An important property is that the sequence of relevant bits in B4VL locational code (equation 2) and in FD locational code (equation 1) are equivalent in practice. So, it is possible that algorithms developed for FD works with B4VL making minor modifications.

Acknowledgements. This work has been supported by the project UV-AE-20050985 of the University of Valencia and the project GV2005-184 of the Generalitat Valenciana.

References

1. Gargantini, I.: An Effective Way to Represent Quadtrees. Communications of the ACM. 25(12), 1982, pp. 905–910.
2. Li, S., Loew, M. H.: The Quadcode and its Arithmetic. Communications of the ACM. 30(7), 1987, pp. 621–626.
3. Samet, H.: Applications of spatial data structures: Computer graphics, image processing, and GIS. Addison-Wesley Longman Publishing Co., Inc., Boston, 1990.
4. Tzouramanis, T., Vassilakopoulos, M., Manolopoulos, Y.: Overlapping Linear Quadtrees: a Spatio-temporal Access Method. Proc. 6th ACM Symposium on Advances in Geographic Information Systems (ACM-GIS'98), 1998, pp. 1–7.