New Quadtree Scheme Using Macro Set

I-Pin Chen

Dept. of CSIE, St. John's University
Taipei, Taiwan, ROC
E-mail: IPinChen@mail.sju.edu.tw

ABSTRACT

In computer applications such as computer–aided part manufacturing and object collision detection, quadtree is a popular representation for object description. However, the space occupied by an object quadtree is known to be rather sensitive to quadtree position. The same object may have different shapes at different positions, and thus different storage sizes. This study proposes a new quadtree scheme using macro set to solve the above problem. This scheme requires no extra memory to store the extra split nodes. Thus the scheme presented here is better than the quadtree in terms of node space saving.

1: INTRODUCTION

In computer applications such as computer–aided part manufacturing and object collision detection, quadtree [1-3] is a popular representation for object description because of its ease of set operations, e.g. intersection operation. The space is partitioned in a recursive quadtree manner, i.e., the space is divided into four square regions, each of which is recursively divided again into four sub-regions.

Linear quadtree [4-5] is usually used to mathematically represent the quadtree. Four directional codes {0,1,2,3} are used to represent the path of a quadtree node in the quadtree. For example, the directional code 0 indicates the north-west child node of the parent node. Moreover, the directional codes 1, 2 and 3 indicate the north-east, south-west and south-east child nodes of the parent node, respectively. Fig. 1 illustrates the linear quadtree representation of each pixel in an 8x8 space. For example, the artificial object in Fig. 2(a) contains four square blocks, which are represented as 0, 30, 330 and 3330. The term quadtree is used hereafter to describe the linear quadtree.

Besides these four directional codes, the quadtree also contains a special "don't care" code '-'. Overall, a quadtree node is expressed as a series of directional codes, q_i, for a $2^N x 2^N$ space, such as $q_{N-1} \ldots q_i \ldots q_0$, $q_i \in \{0,1,2,3,-\}$, $N-1 \geq i \geq 0$. The position of this "don't care" code indicates the size of its corresponding block. If q_{k-1} = '-', then this quadtree node represents a square block of size $2^k x 2^k$. For example, the code 3330 in Fig. 2 indicates a pixel size block; the code 30- indicates a 4x4 pixel size block. The "don't care" code generally is not displayed in the quadtree, that is, code 30- is shown as code 30 only. If q_{k-1} = '-', and $q_i \in \{0,1,2,3\}$, $N-1 \geq i \geq k$, then the subsequent directional codes, $q_{k-2} \ldots q_0$, are meaningless and thus ignored, and thus the representation of the "don't care" code shall not be mixed up with the directional codes {0,1,2,3}. In other words, five codes (four directional codes and one "don't care" code) exist for each q_i in the quadtree representation meaning three bits are required to represent them.

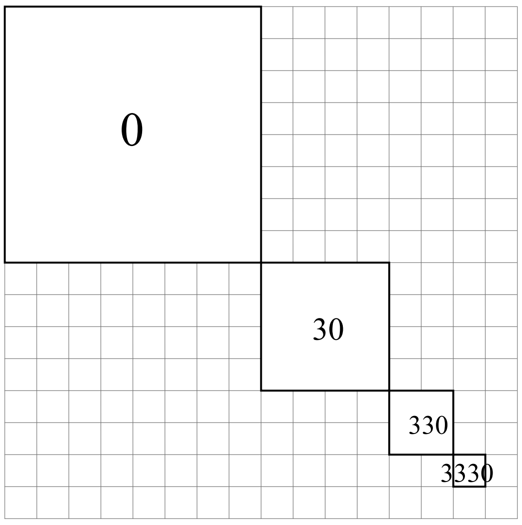
000	001	010	011	100	101	110	111
002	003	012	013	102	103	112	113
020	021	030	031	120	121	130	131
022	023	032	033	122	123	132	133
200	201	210	211	300	301	310	311
202	203	212	213	302	303	312	313
220	221	230	231	320	321	330	331
222	223	232	233	322	323	332	333

Fig. 1. The linear quadtree representation of an 8*8 space.

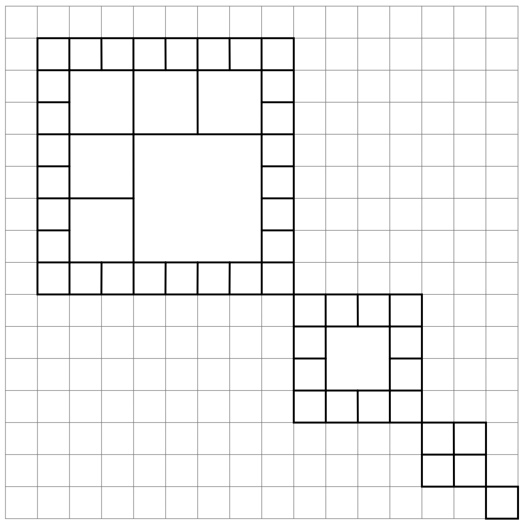
The space occupied by an object quadtree is known to be rather sensitive to quadtree position (where it is partitioned). The same object may have different shapes (that is, results of quadtree partition) at different positions, and thus different storage sizes. For example, when the object in Fig. 2(a) is translated even slightly and becomes the object in Fig. 2(b), the number of nodes in the quadtree changes significantly, from four to 52. Moreover, the directional code of each new generated quadtree node is re-assigned.

Since the quadtree is position-sensitive, the object can be translated within the space to reduce total node number. For example, assume that a quadtree exists with 52 object nodes, as displayed in Fig. 2(b), this quadtree can be translated to an optimal position to become a quadtree of just four nodes, as in Fig. 2(a). The procedure of identifying the optimal position for a quadtree can be divided into two parts. The procedure first determines the optimal grid resolution [6], and then determines the partition point [7]. The quadtree generated in this fashion is called the optimal quadtree, because the number of nodes is minimized. However, this optimal quadtree remains position-sensitive when the resulting quadtree is translated to another non-optimal position.

This study proposes a new quadtree scheme using macro set to solve the above problem. The proposal is based on the fact that a square block remains a square block even after translation. However, the quadtree representation is inadequate to represent such a translated block due to its position-sensitivity. Notably, three bits are required to represent five codes in the quadtree representation, meaning there are a total of eight combinations for the three bits rather than five. This scheme is used to establish a macro set. That is, eight codes are used to describe quadtree node position and size, and thus the translated quadtree node can still be represented as a complete node, rather than as numerous split nodes, in our scheme. This scheme is suitable for describing the quadtree node in a dynamic environment such as the object is frequently translated. Furthermore, this scheme is suitable for collision detection in robot path planning.



(a)



(b)

Fig. 2. The partition (a) before and (b) after translation.

Since the number of quadtree nodes is minimized after the process of quadtree normalization [7], and since the number of nodes remains unchanged after translation in the scheme presented here. Thus the scheme presented here is better than the quadtree in terms of node space saving.

The rest of this paper is organized as follows. First, Section 2 describes this new scheme. Then the expansion of a macro code is described, namely, the generation of all the child nodes of a macro quadtree node at different levels. This expansion is performed using a group of tables presented in Section 3. Subsequently, Section 4 then explains the expansion process using these tables. Section 5 then describes the collision detection. Conclusions are finally drawn in Section 6.

2: NEW SCHEME

This section describes this new quadtree scheme using macro set (or macro quadtree for short). The set of macro codes {a,b,c,d} is used to supplement the original directional code set {0,1,2,3}. Such macro codes not only indicate the size of the corresponding block, but also its position. For example, the macro code ‘a’ indicates that the upper-leftmost pixel in the block is the child 0 of its parent node. Meanwhile, the other macro codes b, c and d indicate that this pixel is the child 1, 2 and 3 of its parent node, respectively. The position of the macro code indicates the size of this block. For example, the macro quadtree representation 03cb indicates a block of size 4x4 whose upper-leftmost pixel is pixel 0321.

On the other hand, given the upper-leftmost pixel, $q^A_{N-1}\ldots q^A_0$, $q^A_i \in \{0,1,2,3\}$, $N-1 \geq i \geq 0$, of a square block of size $2^k x 2^k$, the macro quadtree representation of this block can be determined as $q_{N-1}\ldots q_0$. These macro codes are determined as follows:

(1) if $N-1 \geq i \geq k$ then $q_i = q^A_i$;

(2) if $k > i \geq 0$ then (q_i, q^A_i) = (a,0), (b,1), (c,2) and (d,3) respectively.

For example, for a square block of size 4x4 with upper-leftmost pixel 3002, the macro quadtree representation is 30ac.

Therefore, the artificial object in Fig. 2(a) is represented as 0aaa, 30aa, 330a and 3330 in the new scheme presented here. Notably, the new scheme does not require extra memory space. If this object is translated to become the object in Fig. 2(b), it still contains four macro quadtree nodes, which are 0aad, 30ad, 330d and 3333 in the new scheme.

3: EXPANSION/ MODIFICATION TABLE

Here in this section we will describe the expansion/modification tables used in the proposed scheme. Given a square block of size $2^k x 2^k$, $q^P_{N-1}\ldots q^P_0$,

(1) $q^P_i \in \{0,1,2,3\}$, N-1≥i≥k,

(2) $q^P_i \in \{a,b,c,d\}$, k>i≥0,

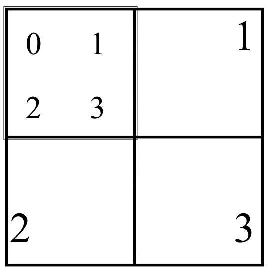
the aim is to generate all child nodes whose macro quadtree representation is $q^C_{N-1}\ldots q^C_0$.

When the macro quadtree node is split into four child nodes, say Child0, Child1, Child2 and Child3, the corresponding macro code of these child nodes, $q^C_{N-1}\ldots q^C_0$, is expanded or modified from the macro code of their parent node $q^P_{N-1}\ldots q^P_0$. First, the expansion/modification of some examples is considered. When macro quadtree node 330a is split into four child nodes 3300, 3301, 3302 and 3303, (q^C_3, q^C_2, q^C_1) of all these child nodes remains the same as (q^P_3, q^P_2, q^P_1) which is (3,3,0). Meanwhile, when the macro quadtree node 330d is split into four child nodes 3303, 3312, 3321 and 3330, the last macro code 'd' is expanded to become {3,2,1,0}, that is, q^C_0 of child node Child0 is 3; q^C_0 of child node Child1 is 2; and so on. Then the preceding directional codes are also modified as necessary. In this example, q^C_1 is modified, while (q^C_3, q^C_2) remains the same as (q^P_3, q^P_2).

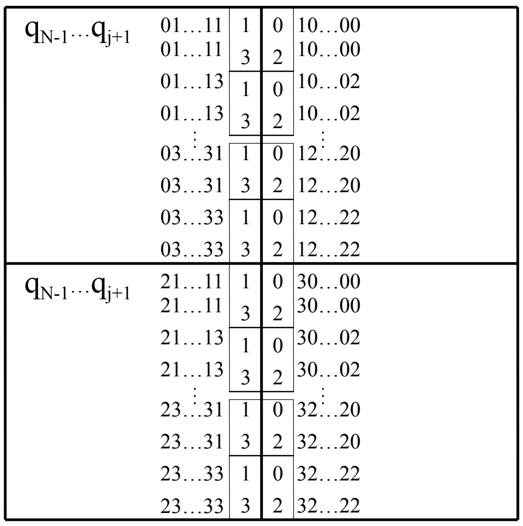
Since $q^P_{i+1} \in \{0,1,2,3\}$ and $q^P_i \in \{a,b,c,d\}$, 16 possible combinations exist for expansion of q^C_i. These 16 combinations are arranged as a table TableM of size 4x4 (see Table 1). The expansion of each macro code of q^C_i is displayed at the top of each column. For example, macro code 'c' is expanded to {2,3,0,1}, which are the directional codes of each of the four child nodes, respectively.

The modification of each macro code of q^C_{i+1} is shown in each row of the corresponding column. For example, when q^P_{i+1} = 3 and q^P_i = c then q^C_{i+1} is expanded to {3,3,1,1}, which are the directional codes of each of the child nodes respectively. That is, when (q^P_{i+1}, q^P_i) = (3,c), (q^C_{i+1}, q^C_i) will be expanded to be (3,2), (3,3), (1,0) and (1,1) respectively.

The modification of macro codes q^C_{i+1} is observed to affect the preceding directional codes. The modification of these directional codes q^C_k , N-1≥k≥i+2, is based on the tables TableB, TableC and TableD. After the expansion according to TableM, the content of column Action is checked. When the content is TableB or TableC or TableD, the modification of the preceding directional codes of each of these child nodes is continued. The modification resembles the process above, except that it is performed according to TableB or TableC or TableD rather than TableM. When the content of column Action is 'STOP', the modification is completed, and the preceding directional codes, if any, remain the same.



(a)



(b)

Fig. 3. The derivation of the expansion of (a) combinational code 'a' and (b) combinational code 'b'.

The derivation of the content of tables TableM, TableB, TableC and TableD is described below. This study observes that the directional codes before macro code ‘a’ remain the same, since the corresponding block of macro code ‘a’ meets the quadtree partition (see the small square in Fig. 3(a)). However, regarding the other macro codes, b, c and d, the preceding directional codes may be modified. Here the macro code ‘b’ is used as an example to illustrate the expansion (see Fig. 3(b)). If q^P_0 = b and $q^P_j = 0$, for some j, and $q_i \neq 0$, $j>i>0$, then the macro quadtree representation of child node, q^C_i, $N-1 \geq i>j$, remains unchanged. A similar situation applies for the case of $q^P_j = 2$. Consequently, for all q^P_i, $j>i>0$, q_i must not be 0 (or 2), since q^P_j already is 0 (or 2), and thus these q^P_i can only be 1 or 3. The relationship between macro code q^C_i , $j>i>0$, of each child node and macro code q^P_i, $j>i>0$, of the parent node can be determined from Fig. 3(b). For example, a macro quadtree node ‘$q_{N-1}\ldots q_{j+1}$01..13b’ is split to be four child nodes whose macro codes are ‘$q_{N-1}\ldots q_{j+1}$01…131’, ‘$q_{N-1}\ldots q_{j+1}$10…020’, ‘$q_{N-1}\ldots q_{j+1}$01…133’ and ‘$q_{N-1}\ldots q_{j+1}$10…022’. Notably, the directional codes $q_{N-1}\ldots q_{j+1}$ are unchanged. TableM and TableB list the derivation result. The cases of macro codes ‘c’ and ‘d’ are similar.

4: NODE EXPANSION

This section uses the expansion of the macro quadtree node 03cb as an example. Starting from q_3 of node 03cb, q_1 was found to be the first combinational code in {a,b,c,d}, and (q^P_2,q^P_1) =(3,c), from TableM, the directional codes (q^C_2, q^C_1) of each of the child nodes, Child0, Child1, Child2 and Child3, are (3,2), (3,3), (1,0) and (1,1) respectively. Since the content of column Action is TableC, q^C_3 of each of the child nodes can be further modified using TableC. Since (q^P_3 ,q^P_2) =(0,3), from TableC, directional code q^C_3 of each of the child nodes is 0,0,2 and 2, respectively. Accordingly, the macro codes of the child nodes Child0, Child1, Child2 and Child3 are 032b, 033b, 210b and 211b.

Furthermore, macro quadtree node Child0, for example, is split into four child nodes, say Child00, Child01, Child02 and Child03. Since (q^P_1,q^P_0) =(2,b), the directional codes q^C_0 of each of these child nodes are 1, 0, 3 and 2 respectively; and the directional code q^C_1 of each of these child notes is 2, 3, 2 and 3 respectively, from TableM. Since the content of column Action is STOP, (q^C_3, q^C_2) of each of these child nodes remain the same as (q^P_3,q^P_2) of their parent node, which is (0,3). Consequently, the macro codes of child nodes Child00, Child01, Child02 and Child03 are 0321, 0330, 0323 and 0332. Since the content of column Action is ‘STOP’, the expansion process of Child00 is completed. The expansion results of nodes Child01, Child02 and Child03 thus are similar.

5: COLLISION DETECTION

Collision detection for two objects can be implemented by calculating their intersection results. When an object is represented in a quadtree form, the intersection operation is easily done by checking whether a non-empty intersection result exists between any two constituent macro quadtree nodes of these two objects. The intersection of two macro quadtree nodes 03cb and 30ac serves as an example here. Node 03cb is expanded:

03cb→032b,033b,210b,211b.

Moreover, these child nodes are further expanded:

032b→0321,0330,0323,0332,

033b→0331,1220,0333,1222,

210b→2101,2110,2103,2112,

211b→2111,3000,2113,3002.

While node 30ac is expanded:

30ac→300c,301c,302c,303c.

Additionally, these child nodes are further expanded:

300c→3002,3003,3020,3021,

301c→3012,3013,3030,3031,

302c→3022,3023,3200,3201,

303c→3032,3033,3210,3211.

A collision thus detected between these two quadtree nodes, since the final result of their intersection is 3002, which is non-empty.

6: CONCLUSION

Quadtree is a popular representation for object description. However, this representation suffers the disadvantage of position-sensitivity. That is, the storage capacity of an object may vary according to object position. This study proposes a new quadtree scheme using macro set to solve this problem.

The proposed scheme does not require extra memory space. The number of nodes is optimal (minimum) following the quadtree normalization, and remains unchanged following translation, making the presented scheme superior to the quadtree in terms of space saving. This scheme is easy to apply to the collision detection in robot path planning. In the future, we will derive other manipulations of macro quadtree.

REFERENCES

[1] Samet, H., "The quadtrees and related hierarchical data structures," ACM Comput. Surveys 16, 187-260, 1984.

[2] Samet, H., "The design and analysis of spatial data structures," Addison-Wesley, New York, 1989.

[3] Samet, H., "Applications of spatial data structures," Addison-Wesley, New York, 1989.

[4] Atkinson, H.H., Gargantini, I., Walsh, T.R.S., "Filling by quadrants or octrants," Computer Vision, Graphics, and Image Processing 33, 138-155, 1986.

[5] Gargantini, I., Atkinson, H.H., "Linear quadtrees; A blocking technique for contour filling," Pattern Recognition 17, 285-293, 1984.

[6] Grosky, W.I., Jain, R., "Optimal quadtrees for image segments," IEEE Transactions on Pattern Analysis and Machine Intelligence 5, 1, 77-83, 1983.

[7] Li, M., Grosky, W.I., Jain, R., "Normalized quadtrees with respect to translations," Computer Graphics and Image Processing 20, 1, 72-81, 1982.

Table 1 The main table TableM

TableM		The i-th macro code, q^P_i							
		a		b		c		d	
		⇓ (0123)	Action	⇓ (1032)	Action	⇓ (2301)	Action	⇓ (3210)	Action
The (i+1)-th macro code, q^P_{i+1}	0	0000	STOP	0101	STOP	0022	STOP	0123	STOP
	1	1111	STOP	1010	TableB	1133	STOP	1032	TableB
	2	2222	STOP	2323	STOP	2200	TableC	2301	TableC
	3	3333	STOP	3232	TableB	3311	TableC	3210	TableD

Table 2 The modification table TableB

TableB		The i-th macro code, q^P_i		
		1	3	Action
The (i+1)-th macro code, q^P_{i+1}	0	0101	0101	STOP
	1	1010	1010	TableB
	2	2323	2323	STOP
	3	3232	3232	TableB