

Performance Improvement of MX-CIF Quadtree by Reducing the Query Results

Wei Yusi and Shojiro Tanaka

Abstract—An MX-CIF quadtree is a variant of quadtree which is for efficient spatial queries such as whether objects are included by a spatial area. When query objects are indexed, a primary result with candidates which may intersect the query rectangle will be reported to have a successful precise inspection. We saved time from inspecting each of the objects intensively. The fewer the candidates are reported to the exact query; the less the time is used to accomplish a query. In this paper, we propose an improved MX-CIF quadtree, compared with the original MX-CIF quadtree. A filter with our structure will decrease the failure rate of result, that is, a query will get fewer uncertain objects, the mechanism of which accelerates the secondary query. Compare to original MX-CIF quadtree, with polygon data given by JTS Topology Suite (JTS) [1], 42.1%~67.5% incorrect results were filtered out by our improved MX-CIF quadtree, and its cost of tree-building time is only slightly higher than the original MX-CIF quadtree.

Index Terms—Distributed systems, recursive algorithm, spatial data structure, spatial index.

I. INTRODUCTION

MX-CIF quadtree was originally proposed in [2], and extensive literature available [3]-[5]. It is designed for a dynamic environment [5]. A distributed system based on MX-CIF quadtree has much less index-structure updates and traffic cost than system based on R-tree, which is demonstrated in [6]. Though MX-CIF quadtree has advantages over R-tree in a distributed environment, as a filter, it still needs to be more accuracy to speed up the precise queries. In Section 2, the MX-CIF quadtree structure is introduced. An implementation example of MX-CIF quadtree is given in Section 3. Improved MX-CIF quadtree is proposed in Section 4. Finally, experimental result of the improvement will be shown in Section 5.

II. MX-CIF QUADTREE

MX-CIF quadtree is a variant of quadtree data structure which supports area-based query. It is designed for storing a set of rectangles in a dynamic environment. Here we review the algorithm and structure based on [2], [4], [5]. The region of every node is one of the four quadrants of the region of the parent node. Comparing to other quadtrees, in an MX-CIF quadtree, each rectangle inserted to the quadtree node that

contains the rectangle totally, and more than one rectangle can be associated with both leaf node and nonleaf node. There is no limitation about the number of the rectangles associated to a node. The two-dimensional square space is considered to be decomposed into four sub-squares with equal area recursively. Subdivision stops when there is no more rectangles contained in a node or the depth of the tree gets to a set value. The planar partition and structure is given by Fig. 1.

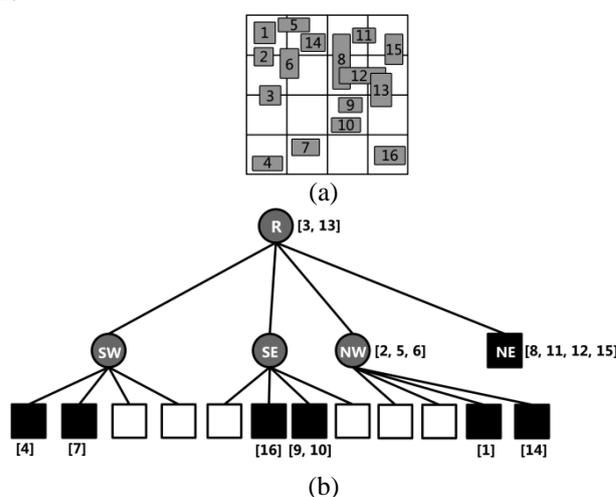


Fig. 1. The planar partition (a) and structure (b) of an MX-CIF quadtree for rectangles

The followings are the properties of MX-CIF quadtree.

- Rectangle of polygons allows association with a leaf or a nonleaf node.
- A rectangle will be associated with only one node. Once a rectangle is associated with a node, say N, then it will never be split to any sub nodes of N. For example, in Fig. 1, rectangle 6 overlaps node NW and its sub-nodes, but it is only associated with node NW.

III. A VIABLE IMPLEMENTATION OF MX-CIF QUADTREE

MX-CIF quadtree is used by JTS [7] for judging whether polygons are nested with the usage of [8].

A node of MX-CIF quadtree in JTS including five elements listed as follows:

- Center, the centric coordinate of a node's rectangle.
- MBR, or say minimum bounding rectangle, the lower left and upper right corner's coordinate of a node's extent.
- Items, the coordinates of every polygon associated with a node.
- Level, the level of an MX-CIF quadtree the node consists in. The level number is the power of two for the size of the node's MBR.
- Sub-node, points to the sub nodes of the node.

Manuscript received August 10, 2012; revised October 1, 2012.

W. Yusi is with the Computer Science, Graduate School of Science and Engineering, Shimane University Matsue, Japan (e-mail: wayis@live.com).

S. Tanaka is with the Computer Science, Faculty of Science and Engineering, Shimane University Matsue, Japan (e-mail: tanaka@cis.shimane-u.ac.jp).

To build an MX-CIF quadtree, first all of the polygons' MBRs are recalculated, an MBR of a polygon is a bounding box that is created with the minimum and maximum x and y coordinate of the polygon. With a polygon's MBR, which node of the quadtree that the polygon should be associated to will be found out by making query to the minimum node whose MBR contains the polygon's MBR. Once all of the polygons are added to the nodes they are associated to, an MX-CIF quadtree is built.

Following is an example with four polygons (Fig. 2).

Assume there are four polygons A, B, C, D. The planar partition is given by Fig. 2, and the structure of the quadtree is given by Fig. 3.

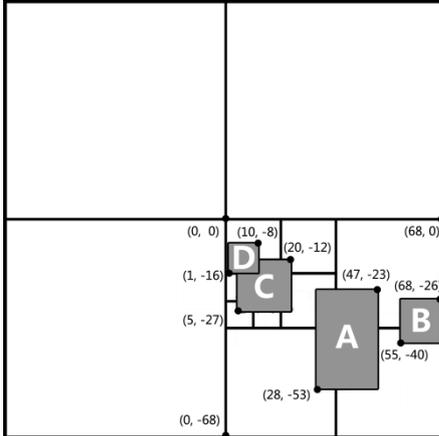


Fig. 2. The planar partition of an MX-CIF quadtree with polygon A, B, C and D

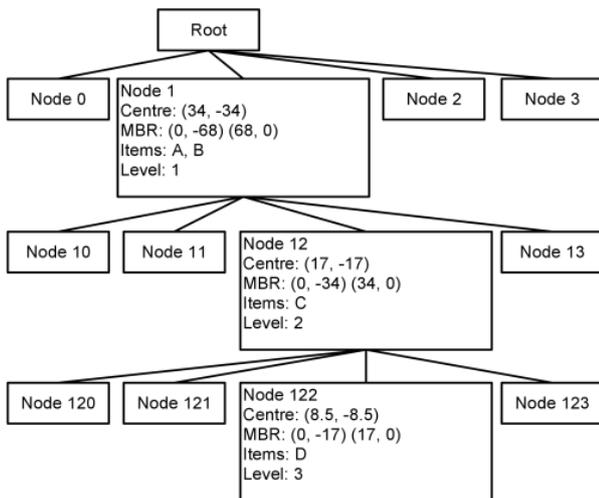


Fig. 3. The structure of the quadtree shown in Fig. 2.

Following is the MBR of these polygons:

- A: (28, -53) (47, -23)
- B: (55, -40) (68, -26)
- C: (5, -27) (20, -12)
- D: (1, -16) (10, -8)

Fig. 3 shows that the value of the MBR in every node is the sum of the range of all its sub-nodes below. Level 0 is considered as the lowest level of the tree. The program judges overlap by comparing every polygon's MBR with every node's MBR. For example, to judge if there is any polygon overlapped polygon C, the program will first use Polygon C's MBR (5, -27) (20, -12) to compare with Node 3's MBR (0, 0) (68, 68). According to the coordinates, Polygon C is covered by Node 1, so add the polygons in Node 1 (i.e. A, B) to a

candidate list then keep on comparing with all the nodes not null. When the recursive is finished, all the four polygons will be added to the list, because the MBR of Polygon C overlapped with the nodes' MBR they associated to. Now the primary filter's work is finished. Then the secondary filter is going to judge if the polygons are nested exactly.

There is a problem in the primary filter. All the MBR of polygons in Node 3 are not overlapped with polygon C, but they still be added to the list. This will lay a burden on the secondary filter, the more the objects report to the exact query; the more the time will be used to finish the inquiry.

IV. IMPROVEMENT OF THE SPEED PERFORMANCE BY RECURSIVE DIVISION OF MBRs

To decrease candidates generated by the primary filter, a "Region-MBR" element is proposed to add into a node of an MX-CIF quadtree structure. The Region-MBR is the sum of the range of the polygons only in the node itself. Improved quadtree structure is given by Fig. 4. When indexing spatial objects with the improved MX-CIF quadtree structure, first, which level and which node in a quadtree the object should be added to, will be calculated, the algorithm follows to the original MX-CIF quadtree structure. After adding the object to a node, the "Region-MBR" will be calculated, if the object is the first one that be added into the node, then the value of the MBR of the object will be the value of the Region-MBR of the node. Once there are more than one objects added into a node, there are two cases.

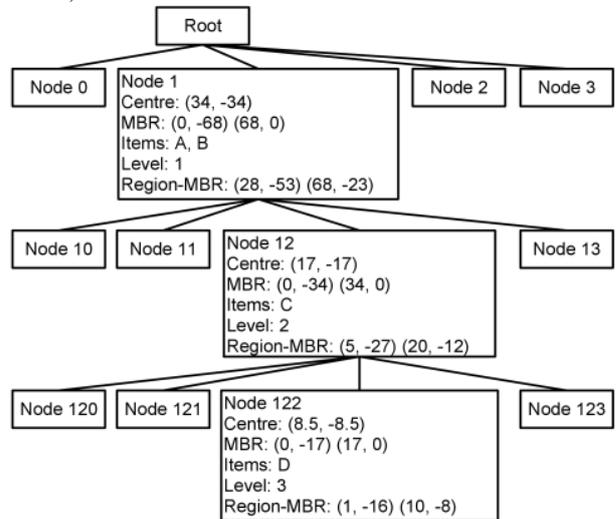


Fig. 4. The structure of improved quadtree.

If the MBR of the new object added into the node that is including the Region-MBR of the node, nothing will be changed for the Region-MBR, else, the area of the Region-MBR will be enlarged to include the new object. When executing query with a search window, first we pick up the non-null node(s) come from the four sub-nodes of the root, then judge if there is an overlap between the search window and all the picked up nodes MBR one by one, the overlapped node will be kept on judgment if the Region-MBR of the node is overlapped by the search window. Because of the area of the MBR of a node includes all the objects' MBR indexed in every sub-nodes in every level below, the program will ignore the sub-nodes of a node

which does not overlapped by the search window. Else the program will keep on collecting objects in all the valid nodes recursively.

The planar partition of the improved MX-CIF quadtree that includes polygon A, B, C and D is shown in Fig. 5. The black border of the polygons represents the area of the Region-MBR of every node. Both of polygon A and B are associated with node 3, so the value of the Region-MBR of node 3 is, lower left point: (28, -53), upper right point: (68, -23), comes from the lower left point's X-Y coordinate and the upper right point's Y coordinate of polygon A and the upper right point's X coordinate of polygon B. Both of Node 30 and Node 302 are associated with only one polygon, so the value of their Region-MBR is equal to their polygon's MBR. From Fig. 5 we see that polygon A and B will not be included as a result, because their node's Region-MBR(28, -53)(68, -23) is not overlapped by Polygon C's MBR(5, -27)(20, -12). Thus reduce the burden of the secondary filter.

Here we have a small experiment to prove that use our structure will reduce the results compare with the original MX-CIF quadtree. Plot (a) of Fig. 6 is a plot of a polygon dataset, plot (b) is a plot of the MBR of every inner hole of the polygon. The test is going to find the rectangle(s) overlap(s) the circled one in all the 30 rectangles. The comparison between MX-CIF quadtree and our structure is given by Fig. 7, which shows that the result given by MX-CIF quadtree includes 10 rectangles while our approach gives only one, the circled rectangle overlaps its self. We see that the right answer should be 0. The reason why the result of both of the two structures includes the circled rectangle itself is because all the 30 rectangles were sent to be judgment with the circled one includes itself.

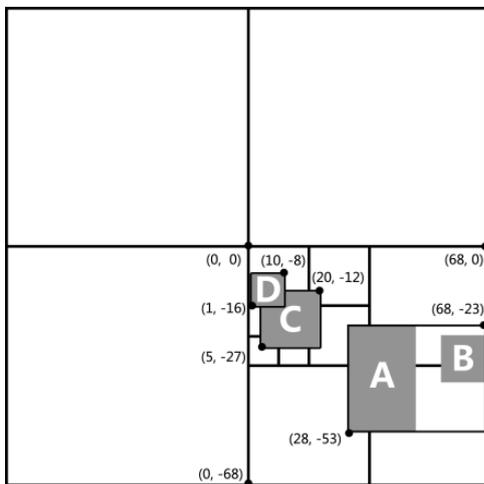


Fig. 5. The planar partition of an MX-CIF quadtree with polygon A, B, C and D

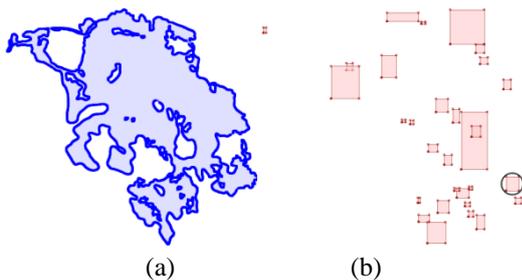


Fig. 6. A plot of a polygon with its inner holes' MBRs

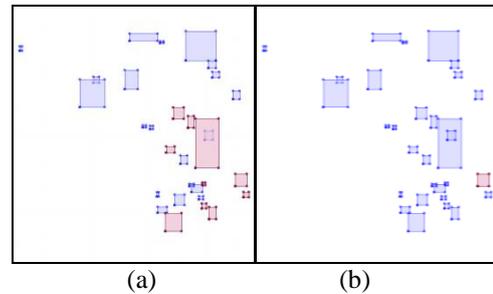


Fig. 7. Result comparison between MX-CIF quadtree (a) and improved MX-CIF quadtree (b)

The reason of the two results is so different is because of the Region-MBR. Fig. 8 shows the polygons of result and their associated node's MBR. Each node and its polygons are painted with different color, the black lines are split lines of the structure. We know from the figure that all the colored MBRs are intersecting with the circled one. In the original MX-CIF quadtree, all of the node's MBR intersect the search window; their associated polygons will be added as a result. Thus all the polygons in Fig. 8 are added as a result. But in our improved structure, only the polygon intersecting both of a node's MBR and Region-MBR can be added as a result. Fig. 9 shows the Region-MBR of the nodes which the polygons associate to. Only the circled Region-MBR intersects the search window, so it will be the only one added as a result.

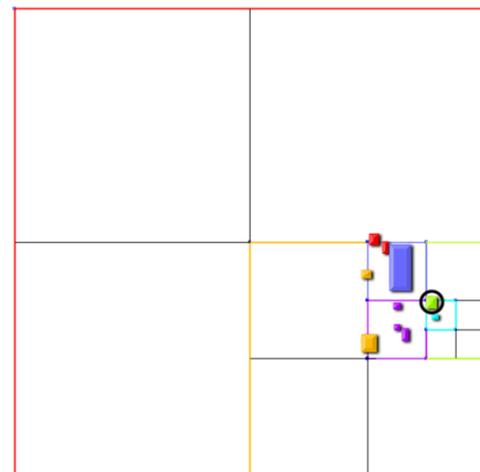


Fig. 8. Polygons of result and their node's MBR

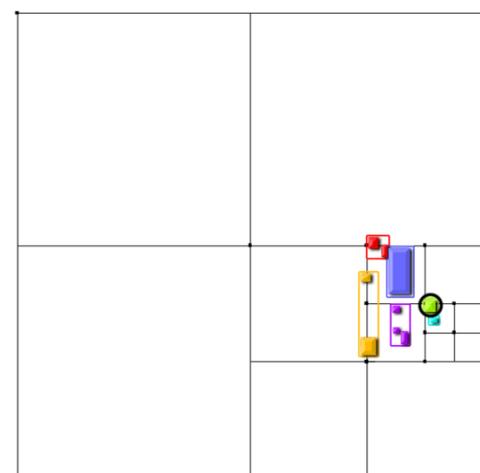


Fig. 9. Polygons of result and their node's Region-MBR

```

Algorithm insertRegionMBR (Node node, Rectangle MBR)
if node.Region-MBR ≠ null then
    Enlarges the boundary of node.Region-MBR so that it contains
    MBR. Does nothing if MBR is within the boundaries;
else
    node.Region-MBR := MBR;
end if;
end insertRegionMBR;
    
```

Fig. 10. The algorithm of insertRegionMBR

The algorithm to calculate the value of Region-MBR of a node when add an associated MBR is listed in Fig. 10. In the expansion, the Region-MBR extends to include an MBR out of its boundary and ignores the smaller ones. The value of Region-MBR should be calculated every time when add an MBR into a node.

```

Algorithm query (Node node, Rectangle searchMBR)
if node.MBR intersects searchMBR do
if node.Region-MBR intersects searchMBR do
    add all the MBRs associated with node into a list defined
    out of this algorithm;
end if;
    for each not null sub-node of node do
        query recursively;
    end for;
end if;
end query
    
```

Fig. 11. The algorithm of query

Another algorithm for making a query with Region-MBR is listed in Fig. 11. Compare to original query method, this algorithm adds one more judgment about if the Region-MBR of a node is overlapped with the search rectangle. Though a

search rectangle intersects with the MBR of a node, but not intersects with the Region-MBR of the node, the objects associated with the node will not be added to the results. Thus this reduces the failure rate of result.

The comparison of tree-building and query will give in section 5.

V. EXPERIMENTAL RESULTS

In this section, we compare the performance of our proposed approach with the original MX-CIF quadtree. The experiment is realized with a nested judgment method [9] of JTS. This method is used to test if any polygons are nested inside another polygon in a dataset. An MX-CIF quadtree structure is used as a filter to speed up the secondary filter of the exact judgment. The process of the nested judgment is introduced as follows: first build an MX-CIF quadtree, then make query for the result of polygons in the index intersected with the first polygon, collecting the candidate polygons as a list then report them to the secondary filter. Then query for the second polygon, then the third and so forth till the last one. The time cost of index, first query, second query and the number of reported polygon by the first query are compared between the original and the improved MX-CIF quadtree. The dataset we used in this experiment includes five WKT (well-known text) data files for polygons with different sizes. Information about the dataset is given in Fig. 12.

File name	Image	Holes in polygon
A.wkt		30
B.wkt		332
C.wkt		789
D.wkt		3276
E.wkt		6937

Fig. 12. Information of the dataset used in experiment.

TABLE I: RESULT OF EXPERIMENT

File name	Structure	Index time	The First Query time	Number of Reported Polygons	The Second Query time	Total time
A.wkt	<i>MX-CIF quadtree</i>	0.0388 ms	0.0318 ms	197	0.0209 ms	0.0915 ms
	<i>Improved MX-CIF quadtree</i>	0.0409 ms	0.0234 ms	64	0.0146 ms	0.0789 ms
B.wkt	<i>MX-CIF quadtree</i>	0.1812 ms	0.8993 ms	9538	0.7692 ms	1.8497 ms
	<i>Improved MX-CIF quadtree</i>	0.1826 ms	0.5987 ms	4156	0.6645 ms	1.4458 ms
C.wkt	<i>MX-CIF quadtree</i>	0.3317 ms	3.1356 ms	38248	2.6131 ms	6.0804 ms
	<i>Improved MX-CIF quadtree</i>	0.3384 ms	2.0763 ms	22144	2.0665 ms	4.4812 ms
D.wkt	<i>MX-CIF quadtree</i>	1.5398 ms	23.6822 ms	256831	17.9644 ms	43.1864 ms
	<i>Improved MX-CIF quadtree</i>	1.5790 ms	12.1951 ms	126061	13.4892 ms	27.2633 ms
E.wkt	<i>MX-CIF quadtree</i>	15.2138 ms	67.2680 ms	757729	66.4776 ms	148.9594 ms
	<i>Improved MX-CIF quadtree</i>	16.2269 ms	47.0981 ms	344162	59.1549 ms	122.4799 ms

TABLE II: RELATIVE PERCENTAGE DETAILS OF THE RESULT

File name	Index time	The First Query time	Number of Reported Polygons	The Second Query time	Total time
A.wkt	+ 5.4123%	- 26.4150%	-67.5126%	-30.1435%	-13.7704%
B.wkt	+ 0.7726%	- 33.4259%	-56.4269%	-13.6115%	-21.8359%
C.wkt	+ 2.0198%	- 33.7830%	-42.1041%	-20.9176%	-26.3009%
D.wkt	+ 2.5457%	- 48.5052%	-50.9167%	-24.9114%	-36.8706%
E.wkt	+ 6.6590%	- 29.9843%	-54.5798%	-11.0152%	-17.7738%

The time costs of result given by TABLE I are average values, which are calculated by twenty time queries.

TABLE II shows the relative percentage details of time cost and polygons reported.

The results indicate that 42.1041%~67.5126% of candidates are deducted to be reported by the first query in an improved MX-CIF quadtree, the indexing time raised by 0.7726% to 6.6590%, but the total query time effectively reduced by 13.7704% to 26.3009%. Due to our proposed approach adds one more parameter to every non-empty node and regulates the value of the parameter when add new object to the node to make the result more accurate, it uses a little more tree-building time, but that is a trade-off with the total query's performance.

VI. CONCLUSION

We have introduced an improved MX-CIF quadtree that provides more precise results to speed up exact queries based on those results. The experiment demonstrated and revealed that our proposed structure reduces the sum of time cost in total for indexing and query. Because of our new MX-CIF quadtree creates one more element in structure, the indexing time rises, but instead of the increased time cost, time cost of query is greatly saved. In our experiment, from 13.7704% to 36.8706% of total cost of time is reduced, compared with the original MX-CIF quadtree.

In our future work, we shall investigate the performance of our improved structure with other systematic tests such as The SEQUOIA 2000 storage benchmark [10].

REFERENCES

- [1] Java Topology Suite-GoldenMap. [Online]. Available: http://www.vividsolutions.com/jts/caseStudy_largePolyValidation.htm
- [2] G. Kedem, "The quad-CIF tree: A data structure for hierarchical on-line algorithms," *Proc. 19th Conference on Design Automation*, pp. 352-357, 1982.
- [3] J.-P. Dittrich and B. Seeger, "Data redundancy and duplicate detection in spatial join Processing," *Proc. 16th International Conference on Data Engineering*, pp. 535, 2000.
- [4] H. Samet, "The design and analysis of spatial data structures," *Addison-Wesley Longman Publishing Co., Inc.*, Boston, MA, pp. 200, 1990.
- [5] E. Tanin, A. Harwood, and H. Samet, "Using a distributed quadtree index in peer-to-peer networks," *The VLDB Journal-The International Journal on Very Large Data Bases*, vol. 16 no. 2, pp. 165-178, 2007.
- [6] R. Zimmermann, W.-S. Ku, and W.-C. Chu, "Efficient query routing in distributed spatial databases," *Proc. 12th ACM Int'l Symp. Geographic Information Systems*, pp. 176-183, 2004.
- [7] The Concept. Characteristics and System Architecture. [Online]. Available: <http://www.vividsolutions.com/jts/JTSHome.htm>
- [8] Source Forge.net. Repast. Repast-Interest. [Online]. Available: <http://www.vividsolutions.com/jts/javadoc/com/vividsolutions/jts/index/quadtree/Quadtree.html>
- [9] Quadtree Nested Ring Tester. [Online]. Available: <http://www.vividsolutions.com/jts/javadoc/com/vividsolutions/jts/operation/valid/QuadtreeNestedRingTester.html>
- [10] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, "The SEQUOIA 2000 storage benchmark," *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, Washington DC, pp. 2-11, 1993.