IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 24, No. 3, June, 2019, pp. 241-247
Journal homepage: http://iieta.org/journals/isi

A Quadtree Spatial Index Method with Inclusion Relations and Its Application in Landcover Database Update

Hongsong Wang[1], Jiangyan Zhu[2*]

[1] School of Geosciences and Info-physics, Central South University, Changsha 410083, China
[2] Network Information Center, Guangdong University of Finance & Economics, Guangzhou 510320, China

Corresponding Author Email: zhu_jiangyan@126.com

https://doi.org/10.18280/isi.240303

Received: 20 February 2019
Accepted: 18 May 2019

Keywords:
spatial index, landcover database, inclusion relation, quadtree, incremental update

ABSTRACT

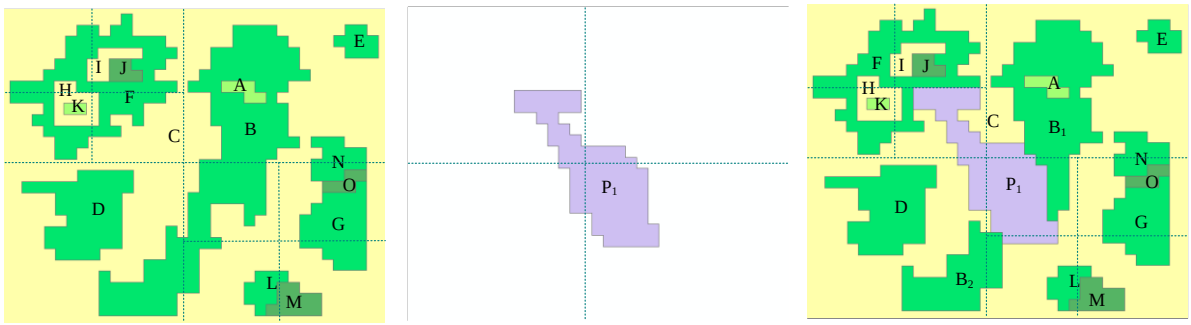
Landcover database often has numerous nonuniform polygons that contain thousands of holes, and even nesting holes. During incremental update, the new changed polygon is used to clip the base state complex polygons, and may intersect a few holes in the latter ones. The traditional update tools, mainly clipping algorithms, must traverse all the holes of complex polygons, which seriously affects the update efficiency. To solve the problem, this paper improves the quadtree spatial index considering the inclusion relations between polygons. In this method, the polygons are divided into two categories: intersecting polygons (intersecting the quadrant axes) and disjointed polygons (disjoint to the quadrant axes). The intersecting polygons are stored on the root nodes on different levels, while the disjointed polygons are stored in the leaf nodes on the index tree. Then, the author introduced the construction of the spatial index and the table of inclusion relations, and explained the operations of the improved quadtree spatial index, namely, insertion, deletion and query. After that, the proposed method was applied to the incremental update of landcover database, and compared with the MX-CIF quadtree index through experiments. The results show that the update efficiency of our method was several times better than that of the contrastive method, and that the efficiency of our method increased with the data volume and complexity.

1. INTRODUCTION

Recent years has seen the emergence of many landcover databases, namely, GlobeLand30 [1]. The databases must be updated regularly to reflect the continuous changes in real-world land use and environment. Therefore, the focus of landcover data handling has shifted from data generation to data maintenance, and an increased attention has been directed towards the update problem. The common update strategy is incremental update, i.e. only renewing the data reflecting the real-world changes [2]. During incremental update, a new changed polygon often has 2D intersections (e.g. overlap, equals and within) with several existing polygons. These polygons must be identified to achieve automatic update. If they are complex polygons (i.e. polygons holes, even nesting holes), which are common in landcover databases of vector data, then it is necessary to determine whether the holes are within the 2D intersections. The holes within and without the intersections are respectively called the involved holes and the non-involved holes. The traditional update tools, mainly clipping algorithms, must traverse all the holes of complex polygons, which seriously affects the update efficiency. The current spatial index methods, grouped by the minimum bounding box (MBB) to minimize the size of the index file and enhance filter efficiency, cannot identify the non-involved holes rapidly. This is because these methods only store the MBB of the exterior ring of the complex polygon, failing to present the inclusion relation between complex polygon and its holes.

The spatial index is used in geographic information system (GIS) to make refined geometric operations that target a limited number of objects. Popular spatial index methods include Quadtree [3-4], R-tree [5-8], R+tree [9], R* tree [10], Grid (spatial index) [11], Hilbert R-tree [12], k-d tree [13] and quadtree [14]. An example of intersecting polygons is given in Figure 1, where C (Figure 1a) is a complex polygon with over 1,000 holes, denoted as B, D, E, F…, P_1 (Figure 1b) is a new changed polygon. The 2D intersection between P_1 and C only covers holes B and F. If treated by the existing spatial index methods, the refined geometric operation must be performed between P_1 and each of the 1,000 holes of C. Coupled with the high cost of the operation, it is extremely complex and inefficient to identify the polygons with the existing spatial index methods.

The Quadtree spatial index is dynamic and efficient in memory space and response time. It has been widely adopted in commercial spatial database management systems (DBMSs) [15]. In this paper, an improved quadtree spatial index is designed based on the inclusion relation between the complex polygon and its holes, aiming to identify the non-involved holes and realize efficient incremental update. In this method, the polygons are divided into two categories: intersecting polygons (intersecting the quadrant axes) and disjointed polygons (disjoint to the quadrant axes). The intersecting polygons are stored on the root nodes on different levels, while the disjointed polygons are stored in the leaf nodes on the index tree.



(a) Data before update (b) New changed polygon (c) Data after update

Figure 1. An example of intersecting polygons in incremental update of a landcover database

The remainder of this paper is organized as follows: Section 2 introduces the construction of the improved quadtree spatial index, including determining the inclusion relations between the complex polygon and its holes and optimizing the traditional quadtree spatial index; Section 3 modifies the improved quadtree spatial index based on the dynamic operations (e.g. insertion, deletion and query) of spatial objects; Section 4 applies the improved quadtree spatial index to the incremental update of landcover database; Section 5 experimentally compares the improved quadtree spatial index and the MX-CIF quadtree index; Section 6 wraps up this paper with meaningful conclusions.

2. CONSTRUCTION OF THE IMPROVED QUADTREE SPATIAL INDEX

The inclusion relation between the complex polygon and its holes must be clarified before the incremental update of landcover database. However, this relation is not stored in the current geological information system (GIS) models. Thus, the first step of this section is to determine this relation. The inclusion relation is indirect if the hole is nested, and direct if otherwise. As shown in Figure 1(a), the generalized region of F (i.e. the union of F and its holes) is one hole of C; the generalized region of H (i.e. the union of H and its hole- K) is one hole of F, and the union of I and J is another hole of F. There is no direct inclusion relationship between C and H and K. In other words, F is the child polygon of C, and the parent polygon of H and the union of I and J. Then, the direct inclusion relation can be expressed as [16]:

{CP, PP, RIP, CPL}

where CP is the current polygon; PP is the parent polygon; RIP is the ring of the current polygon in its parent; CPL is the children polygon list of the current polygon. For the traditional quadtree spatial index, the search space is recursively decomposed into four quadrants: northwest (NW), northeast (NE), southwest (SW) and southeast (SE). The four quadrants are not overlapped at the same level. Meanwhile, the index size will increase significantly due to the duplication of object in neighbour cells. As shown in Figure 1(a), object C must be stored in NW, NE, SW and SE; object B must be stored in NE, SW and SE; object N must be stored in NE, SE on the first level. The storage principles lay a heavy burden on index size and dynamic operations.

To solve the problem, this paper improves the quadtree spatial index method by dividing the polygons into disjointed polygons and intersecting polygons. The intersecting polygons are further divided into five types, depending on the intersection position (only on the positive X-axis, only on positive Y-axis, only on the negative X-axis, only on the negative Y-axis, and both on X- and Y-axes). The five types are stored separately in five buckets, respectively denoted as *XpB, XnB, YpB, YnB* and *XYB*. Meanwhile, the disjointed polygons are stored in leaf nodes. The data structure of each node can be illustrated as [16]:

{NID, NMBB, Subtrees, PPtr, Depth, *XpB, XnB, YpB, YnB, XYB*}

where NID is the identity of the node; NMBB is the union of the MBB of all polygons; Subtrees are the subtrees of the current node; PPtr is the parent pointer of the tree; Depth is the depth of the node level. The relation between the complex polygon and its holes is stored as: {PID, PP, ER, LIPs}, where PID is the identity of the polygon, PP is the parent polygon, ER is the exterior ring, and LIPs is the list of the interior polygons. Figure 2 presents the improved quadtree index and the table of inclusion relations of the data in Figure 1(c).

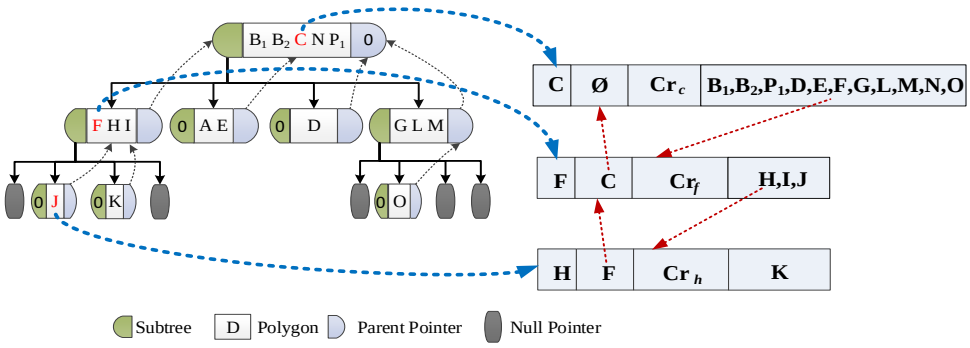


Figure 2. The improved quadtree index and the table of inclusion relations of the data in Figure 1(a)

The improved quadtree was established in the following steps. Firstly, the MBB of all polygons and holes were calculated [17] and the table of inclusion relations was constructed. Then, the search space was decomposed into quadrants recursively until the number of MBBs in each node fell below the pre-set threshold. The polygons whose MBB overlaps the quadrant axes were stored to the five buckets, and those whose MBB falls within a quadrant are stored to the quadrant nodes on the respective level. In the improved quadtree, each object is stored in one node only.

3. IMPROVED QUADTREE INDEX METHOD

Queries can be divided into point query and window query. The latter is related to the quadtree improved in our research. The window query of the improved quadtree contains the following steps:

Step 1: From the root to leaf, check if any polygon overlaps the argument MBB in the five buckets and the quadrant nodes on each level, and set up the list of candidate MBBs.

Step 2: For complex polygons, add the children polygon with RIP overlapping the argument MBB to the result set.

Step 3: For the other polygons, add the polygon itself into the result list, if any point of the argument window falls in the exterior ring and outside the candidate children polygon.

The incremental update involves many dynamic insertion and deletion operations. The two types of operations are introduced concisely below.

To insert a polygon to the improved Quadtree, the first step is to select a bucket or leaf. If the page is not full, a new entry should be inserted; otherwise, the quadrant should be split. The table of inclusion relations must not be changed in the insertion process. The insertion operation is illustrated by an example in Figure 3.

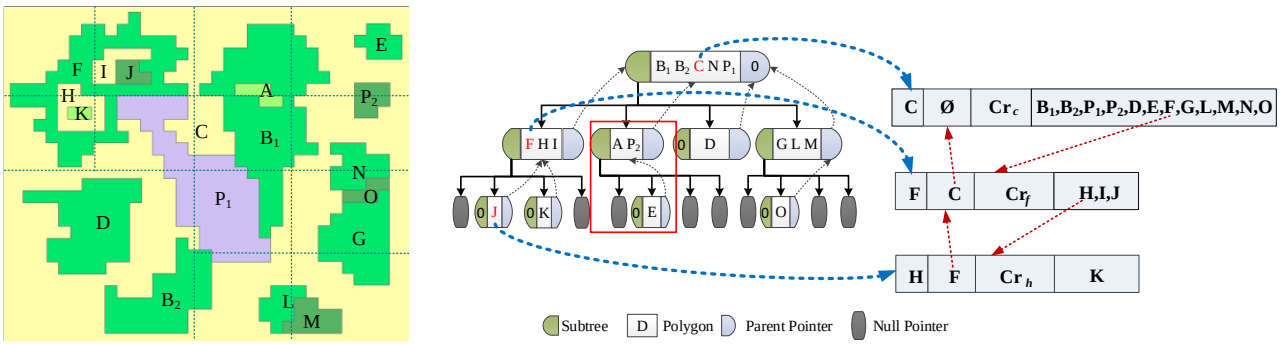
Step 1: Polygon P_2 was inserted into the NE quadrant node on the second level. Since the number of objects (3) is greater than the threshold (2), the NE quadrant was split.

Step 2: Polygon P_2 was stored in the bucket on the second level, and polygon E in the NE quadrant node on second level were stored to the corresponding quadrant node on the third level.

Step 3: The parent polygon C of polygon P_2 was searched for upward from the NE quadrant node on the second level, and the polygon C was found as the parent polygon.

Step 4: Polygon C was replaced with new polygon C, and the table of inclusion relations was updated.

Deletion is the reverse operation of insertion.



(a) Inserting P_2 to the data in Figure 1(c); (b) The post-insertion quadtree

Figure 3. Example of insertion operation

4. APPLICATION IN LANDCOVER DATABASE UPDATE

During incremental update, a new changed polygon often has 2D intersections with several existing polygons or several holes in the complex polygon. Most of the disjointed holes can be filtered with our improved quadtree index, considering the inclusion relation. It is assumed that the changed landcover object is much smaller than the current database, and the changed objects are stored in the main memory during the update. Then, the landcover database can be updated incrementally in two steps: setting up the improved quadtree index and reconstructing the objects after the update. The first step has been detailed in the previous section. The second step is explained in this section with the data in Figure 1 as the example. As shown in Figure 4, the reconstruction operation consists of six steps:

Step 1. A new changed polygon P_1 was selected, and the involved existing polygons (B, C and F) were identified by the Quadtree index. As shown in Figure 4(a), C is a complex polygon with several holes.

Step 2. Only two interior rings of C, i.e. the boundary of B and F, are involved in the update, because B and F are the children polygons of C.

Step 3. A new polygon C' was created with one exterior ring and two interior rings, i.e. the boundary of B and F (Figure 4(b)).

Step 4. The two holes of the new polygon C' was clipped with P_1, forming polygon C'' with only one hole (Figure 4(c)).

Step 5. The polygons B and F were clipped with P_1, using the inclusion relation in the improved quadtree, forming the new B and F (Figure 1(c)). After C'' (Figure 5(a)) was updated, the non-involved interior rings were backfilled into C'' (Figure 5(b)), creating the new complex polygon C (Figure 5(c)).

Step 6. The new objects were stored in the set of temporary objects.

Using our improved quadtree index, the incremental update process of landcover database can be implemented in the following steps (Figure 6):

Step 1. The improved quadtree index was developed for the existing landcover database.

Step 2. One changed objected was selected from the set of changed data.

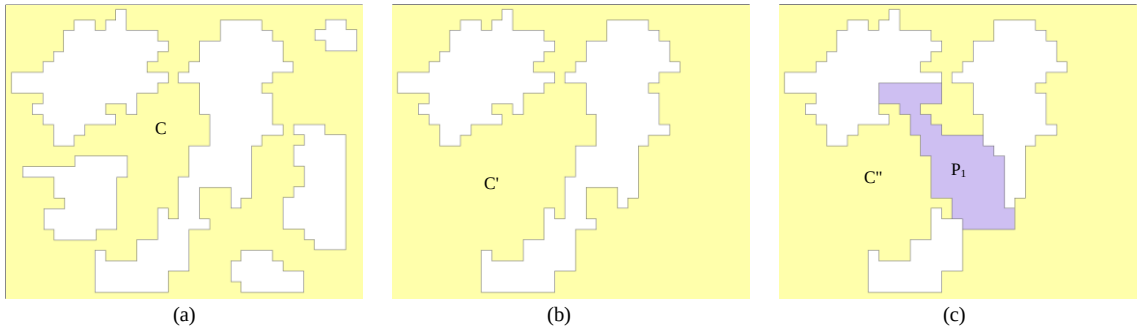


Figure 4. Example of reconstruction operation

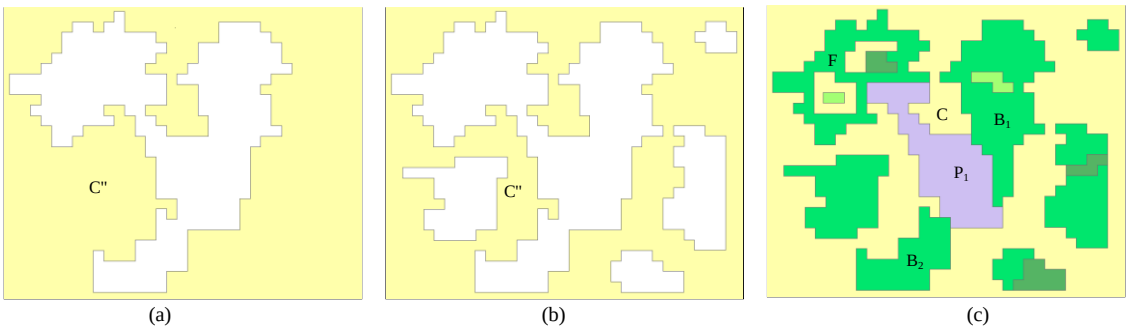


Figure 5. Reconstruct the complex polygon after updating

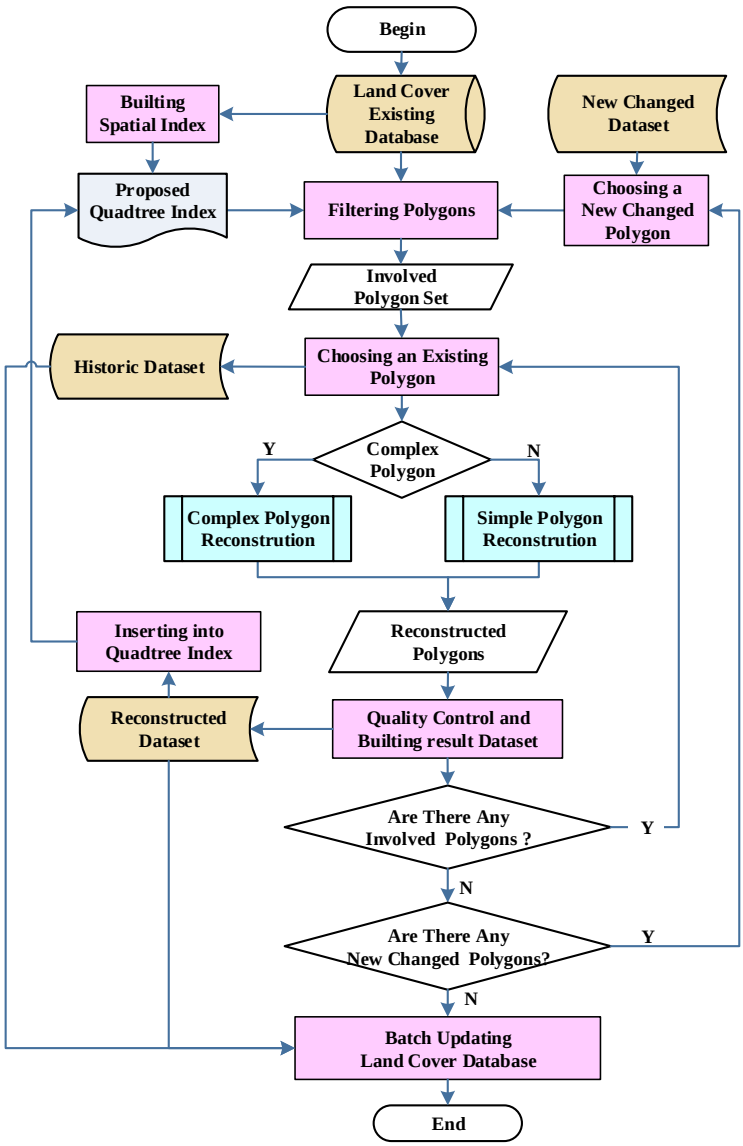


Figure 6. Incremental update process of landcover database by improved quadtree index

Step 3. The improved quadtree index was employed to filter out the non-involved polygons in the existing database, and the involved polygons were stored in a set.

Step 4. One polygon was selected from the involved object set, and checked if it is a complex polygon.

Step 5. If the polygon is a complex one, the updated object was obtained through reconstruction operation. Otherwise, the updated object was obtained by clipping the existing polygon with the new changed object.

Step 6. The quadtree index was updated, without changing the inclusion relation.

Step 7. Steps 4-6 were repeated until all existing involved objects had been handled.

Step 8. Steps 2-7 were repeated until all new changed objects had been handled.

Step 9. The topological consistency was checked and the conflicts were solved one by one.

Step 10. The landcover database was updated to form the new state data, and the outdated data were stored in the history database.

The above update process eliminates the need to traverse the non-involved interior rings in the update of complex polygon, thus improving the update efficiency. In the above process, one of the core algorithms is the update of complex polygons. The general idea of this algorithm is shown as Figure 7, and can be implemented in the following steps:

Step1: Choose an existing involved polygon, and split it into relevant and irrelevant interior rings according to inclusion relation stored our improved index;

Step 2: Build the temporary polygon with relevant interior rings and update it with update rules for complex polygons;

Step 3: Rebuild the reconstructed polygons and insert it into result dataset;

Step 4: Maintain the inclusion relation in our improved quadtree;

Step 5: Steps 1-4 were repeated until all existing involved polygon s had been handled.

The traditional way to update the complex polygon has a time complexity of $O((e+k)\log e)$ [4,17], where e is the number of all segments of the rings of the polygon, and k is the number of intersections. By contrast, the time cost of the improved quadtree index is only related to the number of interior rings involved in the update. This is attributable to the inclusion relation in the spatial index, which filters out the non-involved interior rings.

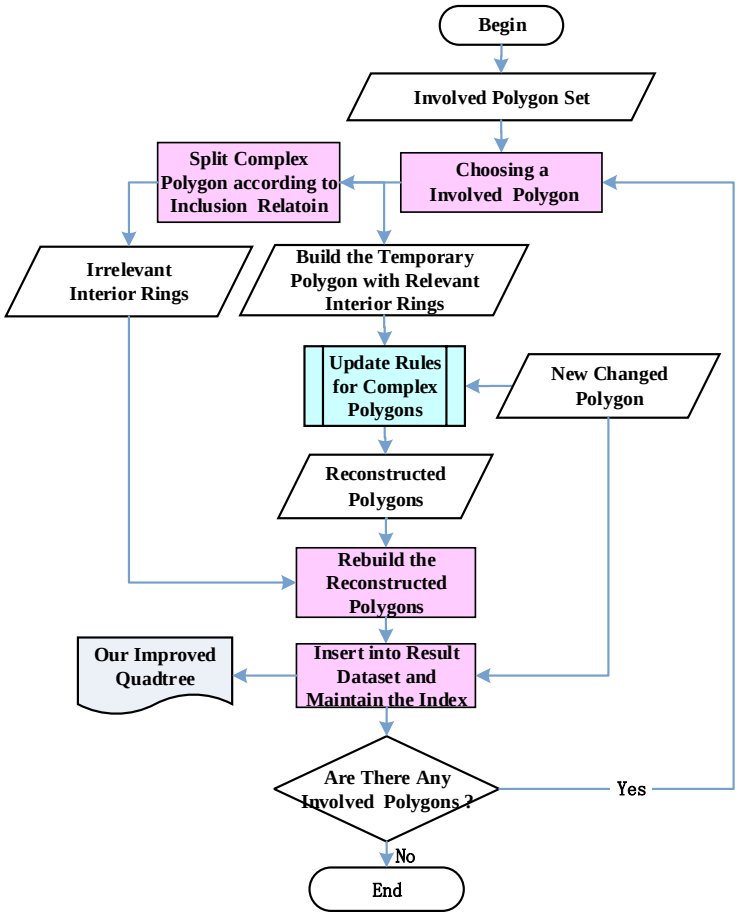


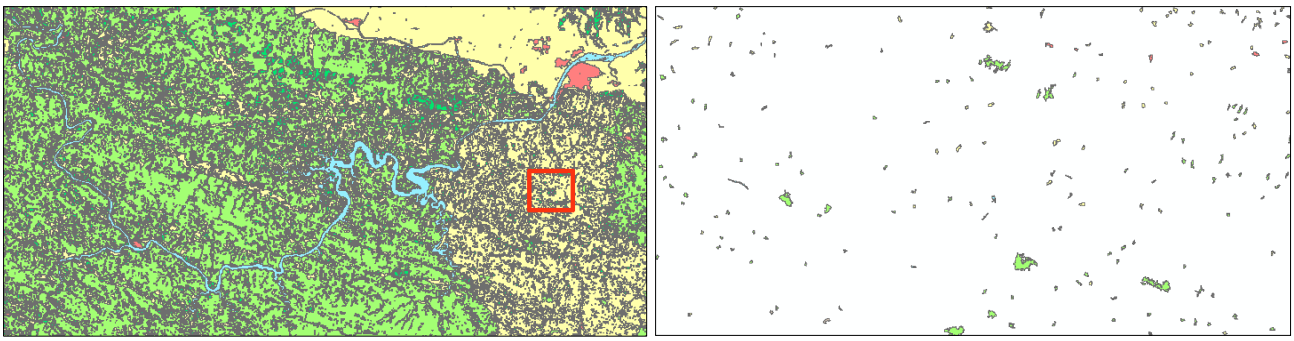
Figure 7. Incremental update process of landcover database by improved quadtree index

5. EXPERIMENTAL VERIFICATION

The improved quadtree index was adopted for the incremental update of complex polygons on Visual Studio 2013. The test landcover database includes the vector data (Figure 8(a)) on a region (N: 36°54’12’’~38° 39’87’’; E: 108° 32’41’’~110°83’01’’) in northern China’s Shanxi Province. The data were collected from the remote sensing images of Landsat ETM+/TM (spatial resolution: 30m). To diversify the test data, the small polygons were merged into large ones with different thresholds. The polygons in the database generally have many holes. The most complex polygon includes about

6,000 holes. The new changed data include 181 new polygons added (Figure 8 (b)) from 2009 to 2010. These data were produced with a change detection software for remote-sensing images. For comparison, the improved MX-CIF quadtree [4] was also selected for the experiment.

The query efficiency was measured through five tests. In each test, 100 query points and 100 polygons were produced randomly in the test area. The test results are listed in Table 1.



(a) Vector landcover data in 2009 (b) The new changed data

Figure 8. The test landcover database

Table 1. Query time of the improved quadtree index and the MX-CIF quadtree index

	Point Query		Window Query	
	Our method ms	MX-CIF ms	Our method ms	MX-CIF ms
1	2.348	5.256	40.542	73.940
2	2.118	4.717	40.085	58.581
3	2.478	4.617	37.342	57.597
4	2.183	5.658	46.536	66.745
5	2.075	5.055	36.892	60.659
Average	2.241	5.061	40.280	63.505

Figure 9 compares the incremental update time of the two methods. Note that the most complex polygon in the test area has more holes than any other polygon, and the time cost of the update includes the time for reconstructing the index and the table of inclusion relations.

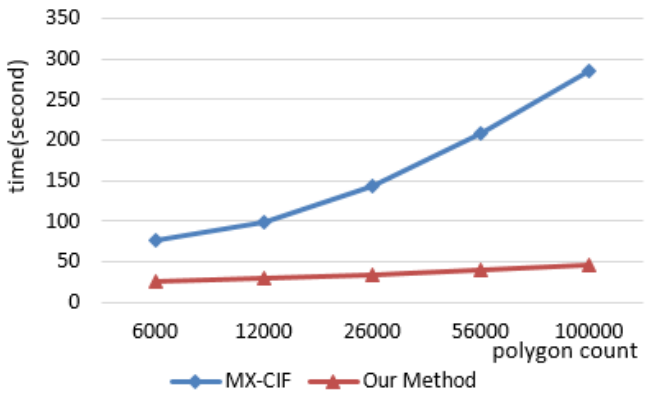


Figure 9. Incremental update time of our method and the MX-CIF methed

From Figure 9, it can be concluded that our method consumed much less time to update the landcover database than the MX-CIF quadtree index. The update efficiency of our method was several times better than that of the contrastive method. The experimental results also show that the efficiency of our method increased with the data volume and complexity. The update efficiency of our method was more than 6 times that of the MX-CIF quadtree index, when the number of original polygons was above 100,000 and the number of holes in the most complex polygon stood at 6,000.

6. CONCLUSIONS

This paper improves the quadtree spatial index for incremental update of landcover database. In this method, the intersecting polygons are stored in five buckets of root nodes on different levels, the disjointed polygons are stored in the leaf nodes, and the inclusion relations between the complex polygons and their holes are stored in a separate table. Then, the author introduced the construction of the spatial index and the table of inclusion relations, and explained the operations of the improved quadtree spatial index, namely, insertion, deletion and query. After that, the proposed method was applied to the incremental update of landcover database, and compared with the MX-CIF quadtree index through experiments. The results show that the update efficiency of our method was several times better than that of the contrastive method, and that the efficiency of our method increased with the data volume and complexity.

Compared with the existing methods for landcover database update, our method boasts the following advantages:

(1) The non-involved holes can be filtered efficiently, as the MBB of the holes of complex polygons are stored explicitly.

(2) The query efficiency is high thanks to the storage of intersecting polygons in the five buckets of root nodes on different levels. This storage pattern reduces the duplicate storage of objects in adjacent quadrants.

(3) The complex polygons can be updated efficiently, for

the inclusion relations between complex polygons and their holes are stored in a separate table.

Our research only targets the objects (i.e. complex polygons and their holes) stored in the same layer. However, the objects may span across different layers in actual landcover database. This problem will be tackled in future research.

ACKNOWLEDGMENT

The work described in this paper was supported by the National Key Research and Development Program of China (NO.2016YFB0501403) and the National Natural Science Foundation of China (No. 41371366).

REFERENCES

[1] Chen, J., Cao, X., Peng, S., Ren, H. (2017). Analysis and applications of global land 30: A review. International Journal of Geo-Information, 6(8): 230-230. https://doi.org/10.3390/ijgi6080230

[2] Zhou, X.G., Chen, J., Zhan, F.B., Li Z.L., Madden, M., Zhao, R.L., Liu, W.Z. (2013). A Euler-number-based topological computation model for land parcel database updating. International Journal of Geographical Information Science, 27(10): 1983-2005. https://doi.org/10.1080/13658816.2013.780607

[3] Hjaltason, G. R., Samet, H. (2002). Speeding up construction of Pmr quadtree-based spatial indexes. The VLDB Journal, 11(2): 109-137. https://doi.org/10.1007/s00778-002-0067-8

[4] Wei, Y., Tanaka, S. (2012). Performance improvement of MX-CIF quadtree by reducing the query results. International Journal of Computer Theory & Engineering, 4(6): 902-906. https://doi.org/10.7763/IJCTE.2012.V4.603

[5] Murakami, S., Takemoto, T., Ito, Y. (2012). Data updating methods for spatial data infrastructure that maintain infrastructure quality and enable its sustainable operation. ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Melbourne, Australia. https://doi.org/10.5194/isprsarchives-XXXIX-B4-29-2012

[6] Warekuromor, T., James, A., Anifowose, B., Trodd, N. (2017). A distributed, scalable and provenance-enabled data access protocol for spatial data infrastructure. 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD). https://doi.org/10.1109/CSCWD.2017.8066691

[7] Abramic, A., Kotsev, A., Cetl, V., Kephalopoulos, S., Paviotti, M. (2017). A spatial data infrastructure for environmental noise data in Europe. International Journal of Environmental Research and Public Health, 14(7): 726-726. https://doi.org/10.3390/ijerph14070726

[8] Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts. https://doi.org/10.1145/971697.602266

[9] Sellis, T. K., Roussopoulos, N., Faloutsos, C. (1987). The R+-tree: A dynamic index for multi-dimensional objects. International Conference on Very Large Data Bases, Brighton. http://hdl.handle.net/1903/4541

[10] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B. (1990). The R*-tree: An efficient and robust access method for points and rectangles. ACM SIGMOD international conference on management of data, Atlantic City, New Jersey. 19(2): 322-331. https://doi.org/10.1145/93597.98741

[11] Zhang, X., Du, Z. (2017). Spatial Indexing. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2017 Edition). J. P. Wilson. https://doi.org/10.22224/gistbok/2017.4.12.

[12] Wang, L., Ma, Y., Yan, J., Chang, V., Zomaya, A. (2018). PipsCloud: High performance cloud computing for remote sensing big data management and processing. Future Generation Computer Systems, 78: 353-368. https://doi.org/10.1016/j.future.2016.06.009

[13] Rigaux, P., Scholl, M., Voisard, A. (2002). Spatial Databases with Application to GIS. ACM SIGMOD Record, Academic Press, USA. ISBN:1-55860-588-6

[14] Kedem, G. (1982). The Quad-CIF tree: A data structure for hierarchical on-line algorithms. 19th Design Automation Conference, Las Vegas. https://doi.org/10.1109/DAC.1982.1585523

[15] ORACLE. Developer's Guide. 2019, https://docs.oracle.com/en/database/oracle/oracle-database/19/spatl/loe.html.

[16] Zhou, X.G., Wang, H.S. (2018). A quadtree spatial index method with inclusion relations for the incremental updating of vector landcover database, ISPRS Technical Commission IV Midterm Symposium, October 1-5, 2018, Delft, The Netherlands (Oral Presention, the second best paper).

[17] Zimmermann, R., Ku, W. S., Chu, W. C., (2004). Efficient query routing in distributed spatial databases. ACM International Workshop on Geographic Information Systems. 176-183. https://doi.org/10.1145/1032222.1032249