School of Engineering in Geodesy, Cartography and Surveying
Dept. of Cartographic Engineering, Geodesy and Photogrammetry
Universitat Politècnica de València
Camino de vera, s/n. 46022. Valencia. Spain

IMPROVING DISSOLVE SPATIAL OPERATIONS IN A SIMPLE FEATURE MODEL

Jose Martinez-Llario[1], Jens H. Weber-Jahnke[2], Eloina Coll[1]
[1]Department of Cartographic Engineering, Geodesy and Photogrammetry. Universidad Politecnica de Valencia. Spain.
[2]Department of Computer Science. University of Victoria. British Columbia. Canada.

Abstract: This paper presents an algorithm to improve the performance of a spatial operation called 'dissolve' widely used in Geographic Information System (GIS) through spatial database systems. In simple feature models (lacking of persistent topology) executing some common spatial operations requires a high amount of system resources. Such common operations occur for example in the 'OpenGIS Simple Features for SQL' protocol (SFS), a client-server interoperability standard defined by 'The Open Geospatial Consortium, Inc.' (OGC). The specific spatial operation studied in this paper is called 'dissolve'. It is carried out using the union spatial operator defined by OGC) and consists of removing the boundaries between adjacent polygons. The proposed algorithm improves substantially the performance of this spatial operation and it needs between 100 and 1000 times less amount of resources. This way it enables the database server to carry out this spatial operation on huge datasets containing up to millions of geometries. To check and to validate this algorithm a new open source software package (PGAT) has been developed.

Key word: Geographic Information System; Spatial Databases; PostGIS; Dissolve

1. Introduction

The use of spatial databases in Geographic Information System (GIS) like Oracle Spatial[1] or PostGIS[2] has increased substantially in the recent years. One of the reason of this behavior has been the adjustment of these systems to well-known standard protocols defined by the 'Open Geospatial Consortium, Inc.' (OGC) like the 'OpenGIS Simple Features for SQL' (SFS) [9]. This implementation specification defines interfaces that enable transparent access to geographic data held in heterogeneous processing systems on distributed computing platforms using the SQL language. When the geographic objects are stored using a simple feature model (SFM) the geometries do not share arc or nodes [8], that is, they do not hold the topology spatial relationships in a persistent way [4]. The SFM is not a best choice for operations taking into account relationships between features (such as spatial relations, topological predicates) [6], in fact, some of the spatial operations defined in the SFS specifications do not work well because they do not consider the spatial relationships between different features.

The motivation of this work is to get an algorithm that can work in a proper way with medium and huge datasets especially performing spatial operations as removing boundaries between adjacent polygons. This way, the institutions (especially public institutions which might be more interested in using open source software) [2] can use the open source spatial databases and analyze the geographic information even if it is made up of millions of geometries. So far, this could not be possible using free software and/or standard protocols (SFS and other OGC protocols). The aim of this research is to make it possible.

One of the operators defined by SFS that does not work in a proper way is the spatial operator *union* defined according to the OGC as "*Union (anotherGeometry : Geometry) : Geometry - Returns a geometric object that represents the Point set union of this geometric object with anotherGeometry*" (Fig. 1) [10]. This spatial operator is used to remove the boundaries between adjacent geometries. It can be applied to polygons, arcs and points features. Despite the fact that the standard name of this spatial operator (according to the OGC) is called 'union', in GIS terminology the resulting operation of applying this operator is commonly known as 'dissolve'.

It is necessary to say that in any moment we are talking about an overlapping function but some readers can get confused because the OGC 'union' spatial operator has the same function name that the GIS 'union' overlapping operation.

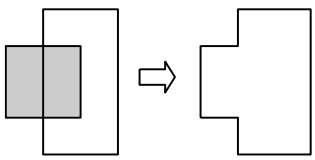

Fig. 1. OGC Spatial operator *union* applied to two polygons

The dissolve spatial operation is a common useful operation in GIS [3]. Take for example a layer containing urban areas: obtaining the block boundaries starting from information about lots requires carrying out this spatial operation by grouping the polygons contained in each block [7] (obviously the lots spatial table does not contain any attribute column with information about the corresponding blocks). As it is described in the next section this spatial operation does not have an obvious solution in a simple feature model because the spatial

1 Oracle Spatial extension. A feature of Oracle Database. Oracle Corporation. http://www.oracle.com
2 Spatial database extension for PostgreSQL. Refraction Research, Inc. http://www.postgis.org

database does not know which lots belong to each block unlike a GIS with persistent topology [3,12]. In other words the spatial database does not contain any information about what the disjointed polygons are.

Fig. 2. Dissolving lots to obtain blocks

To improve the performance of the dissolve spatial operation we need to collect the spatial relationship grouping of the disjointed polygons. According to OGC the spatial operator *union* can be used for joining (dissolving) two features. The spatial databases like Oracle Spatial or PostGIS define a SQL aggregate operator based on the *union* operator. This aggregate function enables these databases to join more than two features [13]. For example to perform a dissolve operation in the whole layer lots the SQL sentence is:

```
INSERT INTO "public"."blocks" ("geom") SELECT
multi (geomunion ("geom")) FROM "public"."lots"
```

This SQL aggregate (called *geomunion* in PostGIS and *sdo_aggr_union* in Oracle Spatial) works in the following way: in a first step it joins the first two geometries (A, B) to obtain just one geometry (c), then it joins this new polygon (c) with a third geometry (C) to obtain a new polygon again (d). The process is repeated as many times as geometries are stored in the spatial table. This way the new geometries obtained are bigger than the previous ones. The process uses an increasing amount of computing resources (memory, time) in each iteration. The final result is a huge geometry (multi polygon in this case). Even though the source spatial table contains just a few thousand of geometries, this final geometry could be made up of millions of vertexes stored in just one row in the spatial table. The resulting geometry is very complex, thus, of limited usefulness for carrying out other spatial operations. Furthermore, the use of a spatial index in subsequent operations does not make help because the table has just one row.

To test the performance of the dissolve operations a computer with the following characteristics was used: Pentium Dual Core 2 1600 Mhz with 1 Gb Ram, running Open Suse Linux 10.2, PostgreSQL 8.1 and PostGIS 1.2.

2.- Approaching the problem

Fig. 3. charts the time to dissolve a spatial table corresponding to a real cadastral dataset like the one showed in Fig. 2. The tests have been carried out only with 10,000 geometries in order to limit the resources needed for the computation. As it is pointed out in Fig. 3 (non-fragmented), PostGIS takes around 1600 seconds (almost half an hour) just for dissolving 10,000 polygons (lots). The resulting spatial table contains only one row. This geometry is a complex multipolygon made up of more than 1,400 polygons.

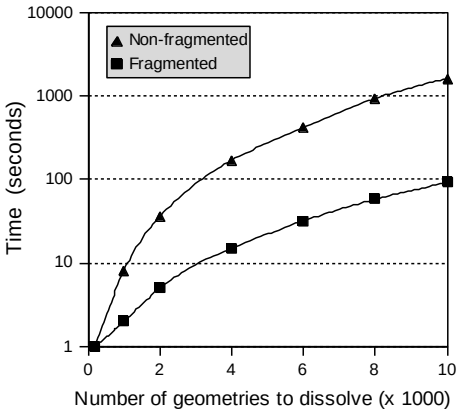

Fig. 3. Dissolving lots (without using spatial relationships)

One approach to improve the performance consists in fragmenting the spatial table into several groups and use the aggregate function in each one of this groups [11,14]. The following code clarifies this option (the column *pgatgid* corresponds to an unique integer value):

```
INSERT INTO "public"."lots_dissolve" ("geom")
  SELECT multi (geomunion ("geom")) AS geom FROM (
      SELECT multi (geomunion ("geom")) AS geom,
            max(pgatgid) AS pgatgid FROM (
        SELECT multi (geomunion ("geom")) AS geom,
            max(pgatgid) AS pgatgid FROM (
          SELECT multi (geomunion ("geom")) AS geom,
                max(pgatgid) AS pgatgid FROM (
            SELECT multi (geomunion ("geom")) AS geom,
                max(pgatgid) AS pgatgid
              FROM "public"."lots"
                GROUP BY mod (pgatgid, 16)
            ) AS FOO8 GROUP BY mod (pgatgid, 8)
        ) AS FOO4 GROUP BY mod (pgatgid, 4)
      ) AS FOO2 GROUP BY mod (pgatgid, 2)
  ) AS FOO;
```

This way the computation time can be improved by more than a factor of 10 (fragmenting the spatial tables in at least 12 groups) compared with the non-fragmented approach. Even though the improvement has been significant the principal problem is remaining, i. e., the operation still results in a very complex single geometry. The resulting spatial table is inappropriate as a basis for any further common operation, e. g., measuring the lots areas or any spatial operation that involves disjointed lots.

A better approach is to create an algorithm that explicitly deals with the disjointed polygons (lots inside a block). To design this algorithm, we needed to take the spatial relationships between the geometries into account as it is explained in the next section.

3.- Solution

As the reader can notice, this article does not talk about how to deal with the object attributes during the dissolve process. Actually it does not offer any difficulty and it is completely solved just using the aggregate and statistic SQL standard functions. The software package developed to test this algorithm considers all of these options (see the bottom of the screen capture in Fig. 8). Consequently the rest of the article the dissolve process referrers just to the geometry component.

Fig. 4 shows the flowchart used to carry out the dissolve operation according to the exposed premises in the previous section (using only the geometry component).

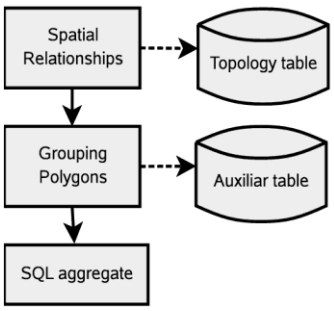

Fig. 4. Flowchart of the proposed algorithm

The first task consists in calculating the spatial relationships between every polygon in the spatial table. The result is stored in a table with two columns where each row represents a pair of polygons that intersect (or touch) each other.

```
SELECT s1."pgatgid", s2."pgatgid"
    FROM
            "public"."lots" AS s1,
            "public"."lots" AS s2
    WHERE
          (s1."geom" && s2."geom")
          AND s1."pgatgid" <> s2."pgatgid"
          AND INTERSECTS (s1."geom",s2."geom")
```

It is crucial that the spatial table (lots) has a spatial index because the above SQL statement makes an intensive use of it (s1."geom" && s2."geom") [1]. The next step consists in grouping the non disjointed polygons. The result is stored in an auxiliary table with two columns: the polygon identifier and the group number. Every polygon inside the same block will belong to the same group number (therefore there will be as many groups as there are blocks).

G_N = group number of each geometry.
P_N= flag indicating that the geometry has been processed.
L_N= set of geometries which intersect with the n geometry (topology data with the spatial relationships).

```
Initialize conditions:
            group_number = 0
            P = false

// Main function
For each geometry g in the spatial table  {
            if Pg = false then fill_geometry (g);
            group_number++;
}

// Recursive function
fill_geometry(g) {
            Pg = true
            Gg = group_number

            For each geometry g' in Lg
                        if Pg' = true then fill_geometry (g')

            }
}
```

Array G_g is stored in an auxiliary table. Then, the aggregate SQL function will group the geometries by using this auxiliary table.

The SQL statement corresponding to the last step of the algorithm is:

```
INSERT INTO "public"."lots_dissolve" ("geom")
  SELECT multi (geomunion ("geom")) FROM
    "public"."lots","pgat"."tmpDissolve_public_lots_geom"
    WHERE

("pgat"."tmpDissolve_public_lots_geom"."pk_public_lots_pgat
gid" = "public"."lots"."pgatgid")  GROUP BY
"pgat"."tmpDissolve_public_lots_geom"."group"
```

where:
'tmpDissolve_public_lots_geom' is the auxiliary table with the column 'group' containing the group numbers of each lot.

4.- Experiments

To obtain reliable conclusions and make an exhaustive analysis some tools have been developed under an open source package called PGAT [5]. This software package has been developed by the authors of this paper. To apply the algorithm showed in this paper this package creates spatial datasets simulating the structure of spatial clustered polygons according to the user defined parameters. Then the designed algorithm is applied and the new dissolved layers can be displayed using PGAT.

4.1.- Software developed

PGAT (PostGIS Analysis Tool) is a graphical interface to PostGIS focused on mapping the spatial operators defined in PostGIS to an intuitive user interface. The spatial operations are performed in the server side unlike most open source GIS. PGAT is implemented with Java and uses GeoTools[1] (to render the graphics) and db4o[2] (to store and manage the log system), both of them are open source solutions. The main difference between PGAT and other programs is that PGAT is focused in performing the

1 An Open Source Library for the manipulation of geospatial data. http://geotools.codehaus.org
2 An Open Source Object Oriented Database. http://www.db4o.com

spatial operations (buffer, dissolve, etc.) on the server side [16] (PostgreSQL / PostGIS / SFS). This way, PGAT commits to the interoperability guidelines defined in the SFS protocol about geospatial operations.

4.2.- Datasets used

The designed spatial tables for testing contain up to hundred of thousand of simple features distributed in a spatial matrix as illustrated in Fig. 5 where the layer is made up of a matrix that divides the space in 10 x 10 zones, each one with 5 polygons (500 geometries in total). Polygons inside one zone do not touch any polygon located in any other zone. Therefore the polygons inside a zone can be considered as lots inside a block as in the previous example. The dissolve layer is shown in Fig. 6.

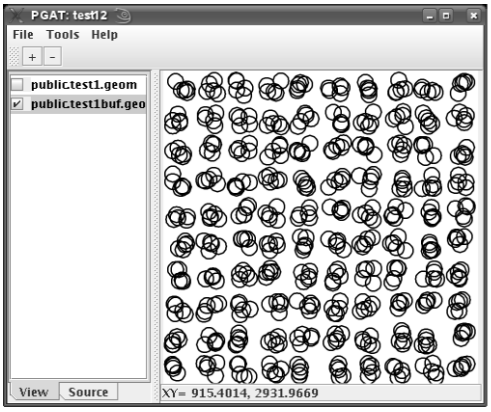

Fig. 5. Test layer

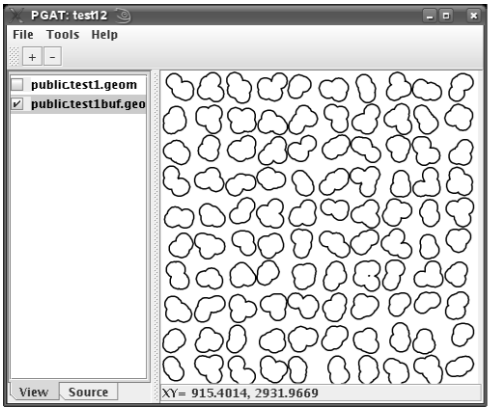

Fig. 6. Test layer after dissolving the polygons

We have implemented the proposed algorithm on the PGAT platform. Fig. 7. shows the configuration box to carry out the necessary spatial operations. As it can be seen, the check box 'not join disjointed geometries' is checked; this way the software will use the proposed algorithm.

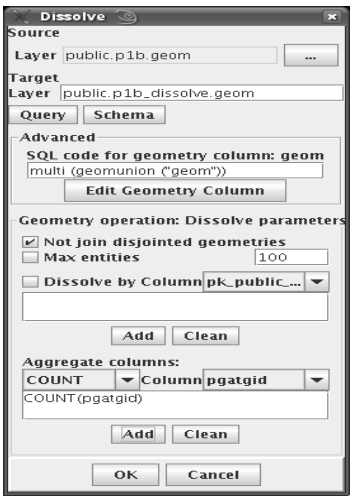

Fig. 7. Dissolve dialog in PGAT

4.3.- Results

These tests have been carried out using PostgreSQL 8.1 / PostGIS 1.2 and Linux (kernel 2.6). PostgreSQL has been configured to use 64 MB of shared memory and the working memory used by the tested algorithm has been calculated inspecting the server processes by monitoring them and creating a log file in an automatic way [15]. The latter is the amount of memory that appears in Fig. 9 and Fig. 11.

The spatial tables follows a typical OGC structure as it is shows in the *psql* terminal, e. g., the next schema is similar to all of the spatial tables used in this paper to test the algorithm:

```
                    Table "public.e1"
    Column     |  Type   |                  Modifiers
---------------+---------+-----------------------------
 pgatgid       | integer | not null default nextval ...
 geom          | geometry |

Indexes:
    "public_e1_pkey" PRIMARY KEY, btree (pgatgid)
    "e1_geom_gistidx" gist (geom)
Check constraints:
    "enforce_dims_geom" CHECK (ndims(geom) = 2)
    "enforce_geotype_geom" CHECK (geometrytype(geom) =
'POLYGON'::text OR geom IS NULL)
    "enforce_srid_geom" CHECK (srid(geom) = -1)
```

The size of these spatial tables (relation size + spatial indexes + toast size) depends on the number of geometries. The size of a table with 500 000 geometries (the biggest one used is this paper to test the proposed algorithm) is 314 MB.

In a first step the fragmented and the proposed algorithm are compared. For it the spatial tables used to compare these two methods contain up to 100 000 geometries (a matrix with 141 rows by 141 columns with 5 polygons in each cell). Fig. 8 shows the run times taken by the fragmented and the proposed algorithm to dissolve these spatial tables. In this case the proposed algorithm reduces the run-time required by a factor of 100 compared to the fragmented one.

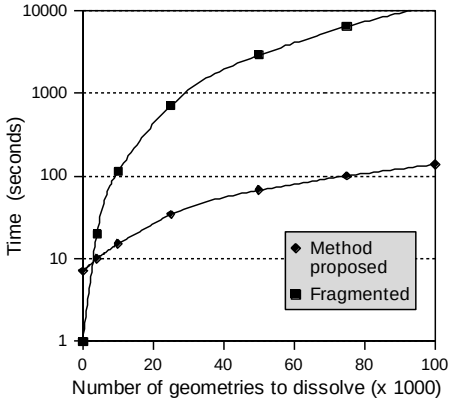

Fig. 8. Run time used (comparison)

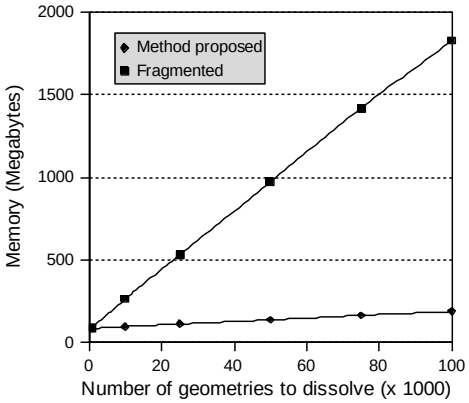

Fig. 9. working memory (comparison)

Analogously the amount of memory needed is huge if this algorithm is not used (Fig. 9). The fragmented option needs about 1,800 MB and the algorithm proposed needs less than 200 MB for dissolving the same number of geometries (100,000 geometries).

In a second step just the proposed algorithm is tested but this time with much bigger spatial tables. Fig. 10 and Fig. 11 show the result of dissolving up to 500,000 geometries using the proposed algorithm and changing the number of polygons to dissolve in each zone (5, 10 and 20 polygons).

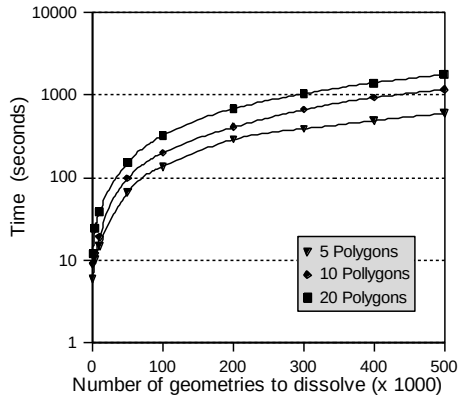

Fig. 10. Run time used (proposed algorithm)

Obviously the algorithm performance gets worse when the number of disjointed geometries is increased (see Fig. 10). But even in that case the results keep being advantageous compared with the fragmented option. Moreover in typical real cases the number of non disjointed geometries are not usually bigger than a few tens.

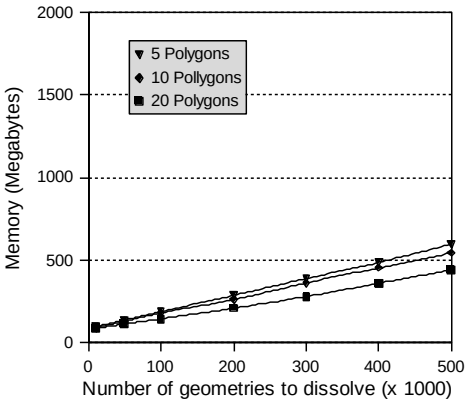

Fig. 11. Working memory (proposed algorithm)

4.4.- Real case

The tests carried indicate that the algorithm is very useful for applications which need to dissolve adjacent polygons. The last step in this analysis is to make sure that the spatial model followed is appropriate to be used with real cases. To that purpose a spatial table containing all the lots of the city of Valencia (Spain) has been used. The algorithm will remove the adjacent boundaries between the lots (around 30 000) to rebuild the blocks of all the city.

Fig. 12. shows the improved performance of the proposed algorithm compared with the fragmented one. Furthermore the resulting spatial table is made up of single polygons corresponding to each block (dissolved lots). The dissolve table corresponding to dissolve 30,000 lot polygons contains about 2,000 rows (one row per block). The Non-fragmented did not work with more then 10 000 geometries because the huge amount of resources needed to run (dashed line in the legend).

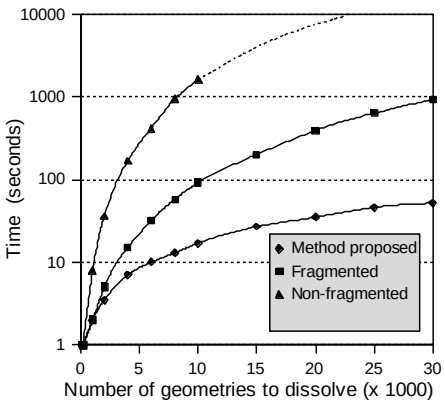

Fig. 12. Run time used for dissolving the lots

The working memory used to dissolve 30,000 polygons is around 119 MB (method proposed), 602 MB (fragmented) and 935 MB (non-fragmented). As the user can check the run-time and memory values are consistent with the results showed in Fig. 8 and Fig. 9 validating the algorithm for real cartographic cases.

5.- Conclusions and future work

The authors have designed and evaluated an algorithm for dissolving polygons that uses much less resources than the current approaches, e. g., SQL aggregate dividing the spatial table into several groups. For dissolving a spatial table made of 100,000 geometries our algorithm requires 200 MB, whereas 1,800 MB are needed for the fragmentation algorithm. Our tests have been made comparing the proposed algorithm with the fragmented one that is already an improvement of using only one aggregate function. If the proposed algorithm were compared with the non-fragmented (Fig. 3) the improvement would be 10 times more (around 1,000 times). The run time performance improvement of the proposed algorithm is at similar magnitude.

The main conclusion is that the designed algorithm enables to use spatial databases with big datasets to dissolve adjacent polygons. This task is not possible using either just an aggregate function like most users do or even grouping the aggregate function and fragmenting the original spatial table because of the huge resources that the server needs.

Another important advantage is that the resulting spatial table contains individual polygons, that is, one polygon or multipolygon for each group of disjointed geometries. The resulting layer takes advantage of the spatial index for the next spatial operations that the user may want to perform. Hence, the resulting individual polygons are more suitable for further queries of a GIS user.

The algorithm has been developed using standard SQL and the SFS protocol, therefore it can be implemented easily in other spatial database systems expecting similar results. The algorithm has been tested using Oracle Spatial (a proprietary solution) obtaining satisfactory results too but the license of this product forbids to public them.

The performance of the proposed algorithm could be further improved if the implementation would use trigger functions to calculate the topology relations and the adjacent polygons. This way it would not be necessary to calculate these spatial relationship each time a dissolve operation is needed. Another interesting work would be to compare this spatial operation in a simple feature model with a system that persist topology information. We intend to study these ideas in our future work.

6.- Acknowledgment

This project has been developed in the University of Victoria (British Columbia, Canada) thanks to the grant awarded by "La Secretaria de Estado de Universidades e Investigacion del Ministerio de Educacion y Ciencia" from Spain (ref. 2006-0264).

7.- Bibliography

[1] Aref W, Ilyas I. SP-Gist: An extensible database index for supporting space partitioning trees. Journal of Intelligent Information Systems 2001;17(2-3):215-240

[2] Coll E, et al. Information and Management in Local Administration. Research project BIA2003-07914 sponsored by the Spanish Government (CICYT) and the European Union (ERDF funds).

[3] Davis B. GIS: A Visual Approach, 2nd Edition. New York: OnWord Press, 2001.

[4] Galdi D. Spatial Data Storage and Topology in the Redesigned MAF/TIGER System. U.S. Census Bureau. Geography division. Available online from http://www.census.gov/, 2005.

[5] Martinez-Llario J. PGAT open source software. Available online at http://sourceforge.net/projects/pgat, 2007.

[6] Oosterom P, et al. The balance between geometry and topology. In : Proc of 10th International Symposium on Spatial Data Handling. Otawa, Canada; 2002.

[7] Oosterom P, Lemmen C. Spatial data management on a very large cadastral database. Computers, Environment and Urban System 2001;25(4-5):509-528.

[8] Oosterom P, Verbree E. Storing and manipulating simple and complex features in database management systems. Proceedings of the 3rd AGILE Conference on Geographic Information Science. Helsinki/Espoo, Finland; 2000.

[9] Open GIS Consortium Inc. Project Document 99-049. OpenGIS® Simple Features Specification For SQL. Available online from http://www.opengeospatial.org, 1999.

[10] Open GIS Consortium, Inc. Project Document OGC 05-126. OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture. Available online from http://www.opengeospatial.org, 2006.

[11] Oracle® Spatial. Oracle Spatial User's Guide and Reference, 10g Release 2. Part III - D Complex Spatial Queries: Examples. Available online from http://www.oracle.com, 2006.

[12] Rigaux P. et al. Spatial Databases: With Application to GIS. London: Morgan Kaufmann, 2001.

[13] Refractions Research, Inc. PostGIS documentation.

Available online from http://www.postgis.org, 2007.

[14] Samet H. The design and analysis of spatial data structures. Boston: Addison-Wesley, 1989.

[15] PostgreSQL Global Development Group. The PostgreSQL Reference Manual Volume 3: Server Administration Guide. Bristol: Network Tehory Limited, 2007.

[16] Zhong-Ren P, Ming-Hsiang, T. Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks. New Jersey: John Wiley & Sons Canada, 2003.