Efficient and Automatic Production of Periodic Updates of Cadastral Maps

Christiaan H.J. Lemmen
Cadastre Netherlands, Company Staff Surveying and Mapping,
P.O. Box 9046, 7300 GH Apeldoorn, The Netherlands.

and

Peter J.M. van Oosterom
TNO Physics and Electronics Laboratory,
P.O. Box 96864, 2509 JG The Hague, The Netherlands.
Email: oosterom@fel.tno.nl.

This paper contains two new contributions in the production process of periodic update files of cadastral maps: 1. both parcels and boundaries get unique identifiers, which avoid geometric searching at the receiving client side; 2. changes are stored in the cartographic main databases, instead of partial database snap-shots (for every different user and period) in an archive. The latter implies a lot of redundant data storage. The new production process deals with aspects such as planar topology and locking of work area's during editing.

1 Introduction

Since one year the Dutch Cadastral and Public Registers Agency provides geometric updates in cadastral maps in digital format to interested users (Municipalities, District Water Boards, etc). After a first delivery of a complete cadastral map of a specified area, update files can be produced according to the wishes of users concerning delivery period and region.

The update files contain the boundaries and parcel identifiers of all new, modified and deleted parcels of this area over this period. The detection of the updates is based on geometric comparison of cadastral objects in the area, requested by the user, for the situation at begin and end of the delivery period. This implies databases in which cadastral objects are represented, have to be put in an archive to compare later for each subscribed user. The has two drawbacks: 1. a lot of redundant data and 2. a lot of computation. Section 2 gives an overview of the current production of periodic updates.

The most important aspects of the Cadastral data model are described in Section 3.

In a new version of the system the update files will be produced directly from the cadastral cartographic main database using time attributes and unique identifiers for each boundary. The cartographic main database contains all historic information. The unique feature identifiers enable much more efficient communication with users of the update files, because difficult geometric searching and comparing of update files with existing data at users side can be avoided. The adapted data model for the new production method is described in Section 4.

The locking procedures of work areas are related to both topology and historic data. The check-out and check-in procedures are described in Section 5.

2 Current Production Update Files

Maintenance of Cadastral Maps in the Dutch Cadastre is supported by the LKI system [4].
LKI stands for *Landmeetkundig Kartografisch Informatiesysteem* (in Dutch): 'Information System for Surveying and Mapping'. Besides Cadastral Maps this system contains the Large Scale Topographic Map of the Netherlands. Geometric data manipulation is done in Fingis workstations [7]. Fingis is the Finnish Geographic Information System and is the basic system for geometric updating and analyses of spatial objects represented in databases of the Cadastre.

In the early days of the availability of digital Cadastral Maps, geometric data could be updated at users side only by delivery of complete new digital maps. However, most users want to receive only the updates over a certain period of time. The Dutch Cadastre expanded the LKI system in order to meet those requirements. Update files are generated automatically now for a specified area, triggered by the update frequency of delivery for a user.

Data processing for production of update files is based on the following datasets [9]:

- 'cartographic main database': a seamless database per province; network database technology; the spatial index is a Field-tree [6],
- 'work database': rectangular area’s can be checked-out of the main database to Fingis work databases; geometric editing of objects in the work database is based on terrestrial surveys; objects in the main database are locked for write access until the work database has been checked-in,
- 'spatial index table': a table with one row per grid cell, this grid structure is based on a quadtree-like subdivision [12] (Fig.1); if a work database is checked-in each grid cell which overlaps the work database area is marked. The production of update files is based now on 'geometric comparison' of the geometric objects per grid cell (Fig. 2). Non marked cells are not included in this process,
- 'archive': to execute the comparison between the actual information in the main database with information in the same area (per grid cell) at an earlier moment (last delivery to the user) the 'older' information has to be stored per cell in an archive system.

One disadvantage of this way of production of update files is multiple storage of the information of the same grid cell, if there are several users, whom want update files of the same area, but with different frequencies. Another disadvantage in the available system is lack of unique feature identifiers for cadastral boundaries, lines, texts and symbols. This implies a complex process for updating the delivered objects at users side, e.g. deleted objects have to be retrieved based on comparison of coordinates. Those disadvantages will be solved in a new ver-

\[\text{Fig. 1: The gray grid cells corresponds to the region of user A; the update files have to be produces for grid cells 1, 2, 3, 5, 7, and 10}\]

\[\text{Fig. 2: The update files are produces by comparing the LKI data at two different moments in time}\]
sion of the system for delivery of update files as described in the next sections. There are performance problems in the current system, those problems will grow because the market is really interested in the product.

3 Cadastral Data Model

The described data model of the new version of the system will be implemented in an extensible Relational Database Management System (eRDBMS, in our case CA-OpenIngres). We will use the Object Management Extension (OME) and the Spatial Object Library (SOL): box, polygon, polyline, point, variable length integer list, and spatial index [2]. The GIS front-end and spatial data edit program is X-Fingis [8]. Though the data is maintained by X-Fingis, other GIS front-ends, e.g. GEO++ [14, 15] can also access the data in the DBMS. Only the relevant parts of the LKI data model are described in this section.

Lines (in the table BOUNDARY) are represented by an Abstract Data Type (ADT) line; see Fig. 3. The spatial extend in the tables BOUNDARY and PARCEL is indicated with a minimal bounding rectangle of type box. There is no need for a type polygon, because the area features are stored topologically in PARCEL using the CHAIN-method. The text/label location in the parcel table (PARCEL) is represented with an ADT point. The ADT’s point, line and box should preferably be based on integer4 coordinates.

The following attributes are included in the data model for all spatial features (e.g. BOUNDARY and PARCEL): id (unique feature id), sel_code (indicates to which map a feature belongs), source (of data), quality (data accuracy, method of measurements), vis_code (visibility code), and akr_area (official area; only for PARCEL).

The parcel is based on a planar topological structure, called the CHAIN-method. The edges contain references to other edges (‘winged edge structure’ [3]), which are used to form the complete boundary chains; see Fig. 4. Further, signed (+/- ) references from the area features to the first edge of their boundary chain and, if islands are present, signed references to the

<table>
<thead>
<tr>
<th>Fig. 3: Part of the Cadastral data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>create table PARCEL(</td>
</tr>
<tr>
<td>pid  integer4, // parcel and text point</td>
</tr>
<tr>
<td>parea float,  // area of related polygon</td>
</tr>
<tr>
<td>bids  intlist, // Boundary Ids (CHAIN)</td>
</tr>
<tr>
<td>bbox  box,  // Bounding box of polygon</td>
</tr>
<tr>
<td>text  char(80), // Text</td>
</tr>
<tr>
<td>sel_code char(6), // Belongs to map: cad/GBKN</td>
</tr>
<tr>
<td>source char(5), // Source of data</td>
</tr>
<tr>
<td>quality char(1), // Data quality: method/acc.</td>
</tr>
<tr>
<td>vis_code char(1), // Visibility code</td>
</tr>
<tr>
<td>akr_area integer4, // Official AKR area parcel</td>
</tr>
<tr>
<td>);</td>
</tr>
<tr>
<td>create table BOUNDARY(</td>
</tr>
<tr>
<td>bid  integer4, // boundary identifier (KEY)</td>
</tr>
<tr>
<td>beg_l_bid integer4, // Line Id left, begin pnt</td>
</tr>
<tr>
<td>beg_r_bid integer4, // Line Id right, begin pnt</td>
</tr>
<tr>
<td>end_l_bid integer4, // Line Id left, end pnt</td>
</tr>
<tr>
<td>end_r_bid integer4, // Line Id right, end pnt</td>
</tr>
<tr>
<td>l_pid  integer4, // Parcel Id left side</td>
</tr>
<tr>
<td>r_pid  integer4, // Parcel Id right side</td>
</tr>
<tr>
<td>bbox  box, // Bounding box</td>
</tr>
<tr>
<td>sel_code char(6), // Belongs to map: cad/GBKN</td>
</tr>
<tr>
<td>source char(5), // Source of data</td>
</tr>
<tr>
<td>quality char(1), // Data quality: method/acc.</td>
</tr>
<tr>
<td>vis_code char(1), // Visibility code</td>
</tr>
<tr>
<td>);</td>
</tr>
</tbody>
</table>

first edge of every island-chain is stored. The sign is necessary in order to determine in which direction the edge has to be traversed.

Besides the references from areas to boundaries, and from boundaries to boundaries, there are also references from boundaries to left and right areas. These are not necessary for forming polygons, but have other useful purposes; e.g. finding neighbor areas.

The advantages and drawbacks of the CHAIN-method are very related to locking, Therefore, they are discussed in Section 5.

4 Storage of Historic Data

Recently, quite a lot of attention has been paid to methods of storing and manipulation spatiotemporal data; e.g. in [1, 5, 10, 16]. Though very complex solutions have been described, our solution is based on a simple extension with 2 attributes per record (tmin and tmax, as done in the research DBMS Postgres [13]) and the possible introduction of successor/predecessor tables. There is a difference between the system tmin/tmax (when is it changed in the main database) and the user tmin/tmax (when did
The CHAIN-method:

Signed references from area feature to first edge of outer-boundary and every island: E1, -E5, E8

Fig. 4: The CHAIN-method for an area feature with islands

the situation change). However, only system tmin/tmax is stored, because it is very hard to maintain consistent user tmin/tmax.

In the data model every table is extended with two additional attributes: tmin and tmax. When a new entity is inserted (during check-in), it gets the current time as value for tmin, and tmax remains unset (gets a special value). Note that unchanged features, even if they are completely inside the locked rectangle, are not affected at all in the main database. The future version of the check-in procedure of X-Fingis will take care of this.

When an attribute of an existing entity changes, it is not updated, but the complete record is copied with the new attribute value. The old version gets current check-in time for its tmax value and the new version (record) gets this time value for tmin. This is necessary in order to be able to retrieve the correct situation at any given point in history. The pair (id,tmin) forms primary key. Note that 'id' is a generic term for the different id's: pid or bid. Alternatives for this time-stamp per record method are [11]: time-stamp per table (simple but highly redundant) or time-stamp per attribute (compact, but requires variable length attributes or difficult changes in the data model).

Fig. 6 shows the example contents of the main database. This database contained on '12 jan' one line (with id 1023) with three points. On '20 feb' this line was split into two parts: one part kept the old id (but with a new tmin value), the other part got a new id (1268); see Fig. 5. Finally, the attribute 'quality' of one of the lines was changed on '14 apr'. The two SQL-queries show how easy it is to produce update information: one query for all deleted/changed lines, and one query for all new/changed lines.

In CA-OpenIngres, time attributes can be represented by the 12 byte date type, but as this attribute is used very often it might be more efficient to use the 4 byte integer. Assuming that every time stamp has to be specified with 1 minute accuracy, then an integer contains enough numbers for more than 8,000 years: $2^{32}/365 \times 24 \times 60 > 8,000$. An alternative might be to use an integer, which is incremented by 1 every time a new check-in occurs. In a specific table, the real update time is stored (together with this number) and other information related to this update (user, rectangle).

Note that the maintenance of the time and historic information is the responsibility of the application, in our case the X-Fingis check-in. However, it is very important that all changes within the same check-in get the same time
stamps; see Section 5.

A lot of information concerning predecessors and successors can be obtained by selecting on id and not on tmin/tmax, because such a query will produce all historic versions of a given object. However, this does not work for splits, joins, or more complicated area reorganizations. Real cadastral objects could be candidates for maintaining explicit lists (in .HIST tables) with predecessors and successors. For the time being no BOUNDARY_HIST will be maintained (perhaps sometime in the future this might change). Using these tables many-to-many (n:m) ‘parent-child’ relationship can be registered; e.g. find all other parcels that were created from the same cluster of parent(s) as parcel with id ’X’. Fig. 8 shows the .HIST’ of the previous BOUNDARY example. Additionally the .HIST of the PARCEL is shown with two non-simple edit events: 3 parcels (1234, 1235, and 1236) are created from 2 parent parcels (10 and 31) on ’01 apr’, further 2 parcels (2363 and 2364) are created from one parcel (77) on ’10 jun’; see Fig. 7. Both parent parcels are deleted, that is, they get a tmax value and not a new record.

One final question related to historic data is: How long should the history be kept inside the main database? The current proposal from the Dutch Cadastre is to keep the information for at least 4 years, before it is removed from the database and put on some kind of long term archive medium. Note that the historic data will not only increase the size of the database, but is will also slow down the response times as the data has to be selected out of a larger data set. It has to be investigated if the response times

Fig. 7: A cluster of PARCELS reorganized: 2 old parcels replaced by 3 new parcels

Fig. 8: Tables for storing predecessor/successor information

<table>
<thead>
<tr>
<th>BOUNDARY_HIST</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>parent_bid</td>
<td>child_bid</td>
<td>time</td>
</tr>
<tr>
<td>1023</td>
<td>1268</td>
<td>20</td>
</tr>
<tr>
<td>1023</td>
<td>1023</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PARCEL_HIST</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>parent_pid</td>
<td>child_pid</td>
<td>time</td>
</tr>
<tr>
<td>10</td>
<td>1234</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>1235</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>1236</td>
<td>01</td>
</tr>
<tr>
<td>31</td>
<td>1234</td>
<td>01</td>
</tr>
<tr>
<td>31</td>
<td>1235</td>
<td>01</td>
</tr>
<tr>
<td>31</td>
<td>1236</td>
<td>01</td>
</tr>
<tr>
<td>77</td>
<td>2363</td>
<td>10</td>
</tr>
<tr>
<td>77</td>
<td>2364</td>
<td>10</td>
</tr>
</tbody>
</table>

// find all parcels created from
// the same parents as parcel ’X’:
select distinct ph1.child_pid
from PARCEL_HIST ph1, PARCEL_HIST ph2
where ph1.parent_pid = ph2.parent_pid and
 ph2.child_pid = ’X’

// or formulated in a different way:
select distinct child_pid
from PARCEL_HIST ph
where ph.parent_pid in (select parent_pid
from PARCEL_HIST
where child_pid = ’X’)

// An alternative data model with 3 tables:
// PARCEL_HIST1(cluster_edit_nr, parent_pid)
// PARCEL_HIST2(cluster_edit_nr, child_pid)
// PARCEL_HIST3(cluster_edit_nr, time)
are in the order of \( \log(n) \), or linear \( (n) \), or even worse \( (n) \) is the total number of objects).

5 Work areas, Locking, Check-out/-in

A GIS is different from many other DBMS-based applications, because the topology edit operations can be very complicated and related to many old and new features. This results in 'long transactions' as it may take up to several hours of editing before the new situation is correct again. During this period other users are not allowed to edit this rectangular region. They must be allowed to view (the last correct state before the editing) of this region. They must also be allowed to edit other non-overlapping rectangular regions.

The main database should always be in a consistent state. It may therefore not be used to manage the 'temporary' changes which are required during the topology edit operations. This is the motivation for the introduction of a temporary copy outside the database for the GIS-edit program, e.g. X-Fingis. The copy is made during check-out and is registered in the table LOCK (only allowed if no other work areas overlap the requested region). The main database is brought from one (topologically) consistent state to another consistent state during a check-in. Without this architecture, it would be extremely difficult to manage the work of the different users working at the same time on the main database. This architecture has three main advantages:

1. enable the required long transactions;
2. enable different users to work at the same time;
3. enable an easy implementation of a high level 'cancel' operation (rollback).

What exactly should be locked when a user specified a certain work area (rectangle)? Of course, everything which is completely inside the rectangle must be locked. The features in the tables PARCEL (parcels) and BOUNDARY (boundaries) do cross the work area boundary should not be locked. In this way all simultaneous edit operations (in different regions) are additional and not conflicting. However, during check-in special care has to be taken with references, because neighbors of non-locked features might change. Simply replacing BOUNDARY 'x' by BOUNDARY 'y' in Fig. 9 implies that the PARCEL outside the lock rectangle has to be updated. The same applies to the neighbor BOUNDARYs 'u' and 'v', their references to 'x' are now incorrect. Beside references, also other attributes may have to be updated (in PARCEL): area, bbox, and tmin/tmax.

Though some attributes may change, the geometry of a BOUNDARY crossing the work area boundary does not change. This is an important concept together with the fact that the rectangular work area's can never overlap. This means that the changes to the BOUNDARYs and PARCELS that cross a border of two work areas are additional and can be merged in the main database. Therefore these features do not have to be locked, but have to be checked-in with some additional care. Remark that, once they cross the border of one work rectangle, this feature can never be completely inside another work rectangle (and can therefore not be locked). The tmin/tmax attributes in BOUNDARY or PARCEL are very important in check-in mechanism (one database transaction), which is described over here:

1. All changed features completely inside work area follow the normal check-in procedure. This is save because there is a lock on the rectangle.
2. Only changed features crossing the rectangle boundary have to be treated with care because they are not locked. It is possible that 2 check-in's want to modify the same feature. Note that the changes are complementary, because they apply to different areas. However, if no care is taken and both check-in's replace the feature, then only the second version is stored and the changes from the the first are lost. Therefore, the following steps must be taken for every changed feature crossing the work area boundary:

   2.a re-fetch the feature from main database (including it's tmin t1);
   2.b if other changes have occurred (e.g.
feature had \( t_{\min} < t_{\max} \), then 'merge' these with the work area version of features;
2.c reinser the 'merged' feature in database.

To ensure that the main database goes from one consistent state to another consistent state, all \( t_{\min}/t_{\max} \) should have the same value for the check-in of one work area.

6 Conclusion

In this paper some improvements are presented on an existing system for delivery of geometric cadastral update files. The improvements concern the maintenance of time attributes during the check in of (modified) geometric objects in a main database. Consequences for topological relationships, locking and unique feature id’s are included. Unique feature id’s for cadastral boundaries, lines, texts and symbols will simplify the check in process of update files at users side.

Acknowledgements

Many valuable comments and suggestions were made by Tapio Keisteri, Esa Mononen and Tom Vijnbrief. Without their contributions we would not have obtained the current system design.

References


