Road Collapse in Magnum

Annita N. Wilschut° Roelof van Zwol° Jan Flokstra° Niek Brasa* Wilko Quak◇

° University of Twente
P.O.Box 217, 7500 AE Enschede, the Netherlands
{annita, zwol, flokstra}@cs.utwente.nl

* Smallworld Systems BV
the Netherlands
niek@smallworld.nl

◇ University of Amsterdam
the Netherlands
quak@wins.uva.nl

Abstract

This paper describes the implementation of a triangulation based collapse algorithm in the general-purpose object oriented DBMS Magnum. The contribution of the paper is twofold. First, we show that true integration of complex spatial functionality in a DBMS can be achieved. Second, we worked out a collapse algorithm to be used in the complex area of map generalization.

1 Introduction

Map generalization [BrB96,Wei96b] is a difficult problem, which has been on the agenda of GIS researchers for a long time. Many papers have appeared that either review the complexity of the problem from a theoretical viewpoint [DeP96,Wei96a] or give a technical solution of a part of the problem [PTM96,RuP96].

Although the current state-of-the-art does not allow a technical solution that covers all aspects of map generalization, the problem is urgent. Manual map production is very costly, and even computerizing certain aspects of this process may lead to a reduction of those costs. Also, automatic map generalization paves the road towards producing a larger variety of map products.

This paper tackles a single issue in map generalization from a practical viewpoint: generalizing roads from 2-dimensional area features (in a very detailed topographic map) into 1-dimensional line features keeping the underlying polygonal map closed. The work on this problem was inspired by the map production process of the Dutch topographic society that produces topographic maps at various scales.

Integration in a general purpose DBMS. The work is done in the context of the Magnum research project [BWK98,WQB97]. In the Magnum project, a structurally object-oriented DBMS prototype is developed. The work in this project is inspired by the fact that it has become apparent that integration of spatial and thematic data in a single data manager has many advantages. Such a system offers a single data model for spatial and thematic data; base-type extensibility is typically used to implement spatial types, like points, polylines, and polygons. Research prototypes that elaborate this idea are GEO++ [OoV94], GeO_2 [ScV92], and Paradise [DKL94]. However, it is shown in [OoV94] and [ScV92] that full and efficient integration of GIS functionality in an extensible relational or an object-oriented DBMS is difficult, because it is hard to model and fully exploit the structure that is typically imposed on spatial data.

In Magnum, we are building a DBMS prototype that is specially suited for the management of spatial data. As a DBMS, the system is able to manage all sorts of data. However, the fact that spatial data is managed by the system has influenced its design. The goal is to study true and efficient integration of a wide range of possibly complex GIS-functionality in the a DBMS. *Decomposition* and *extensibility* are the key features of the design of the Magnum prototype.

The contribution of this paper is twofold. We have worked on an algorithm that describes the road generalization proces in five phases, but more importantly, this algorithm has been adapted such that it could be implemented as part of the Magnum DBMS prototype.

This paper is organized as follows: Section 2 describes the map products of the Dutch topographic society. Section 3 describes the road collapse algorithm in general terms. Section 4.1 outlines the Magnum prototype DBMS, its physical storage server, Monet, and the ways in which Magnum provides support for spatial applications. Section 4.2 is on the representation of polygonal maps in Magnum and it shows that the algorithm of Section 3 can directly be implemented on that representation. Section 4.4, shows the results of our algorithm on a sample of a real topographic map, and Section 5 concludes the paper.

2 TDN topographic maps

The "Topografische Dienst Nederland" (TDN)[1] is a Dutch company that produces topographic digital raster and vector maps at different scales for the Netherlands. In this paper, we look at their vector source map, which has scale 1:10000 and is called TOP10Vector, and its first derivative, the TOP50Vector, at scale 1:50000. The TOP50Vector map is derived from the TOP10Vector map via manual and automatic generalization. Figure 1 shows a sample of both data sets.

[1]The "Topografische Dienst Nederland" (TDN) kindly supplied us with a sample of their data to test the algorithms described in this paper. The TDN has the copyright on the data used to produce the figures in this paper.

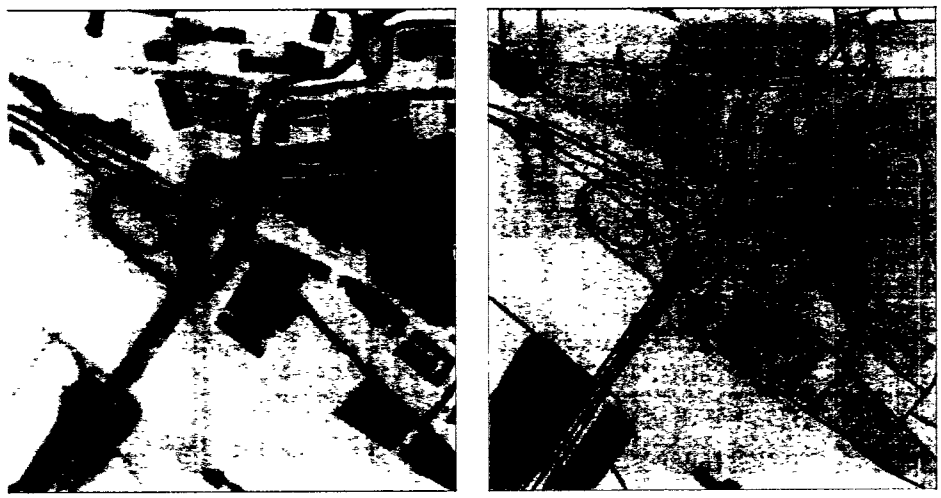


Figure 1: An example of the TOP10Vector and TOP50Vector map of the same area

Both the TOP10Vector and the TOP50Vector maps consist of a base map which, is a polygonal map (a complete and disjoint subdivision of the plane into polygons). The parts of the subdivision are area features. A describing code is associated with each area feature. The boundaries are labeled with codes as well: some boundaries just serve as boundary of an area feature, others also represent a line feature. This is indicated by their label. All line features are integrated in the base map. Apart from the polygonal base map, there are separate layers that contain houses (represented as polygons that are not integrated in the polygonal base map) and various symbols. This paper concentrates on the area features making up the base map.

An important difference between the TOP10Vector and the TOP50Vector maps is the fact that roads are stored in the former as area features in the polygonal map and in the latter as line features. So, the derivation of the TOP50Vector from the TOP10Vector requires (among many other generalizations):

- Generalizing the area features that represent roads into line features, which are the road hartlines.
- Propagating the description codes associated with the roads in the generalization process to the line feature roads.
- Restoring the polygonal base map.

This paper shows how a general approach, described in [BaW97,JBW95] in a different context, is refined into a road collapse algorithm: the roads are collapsed from area features to line features. The result consists of a closed road network and a new complete polygonal base map in which the area covered by roads is distributed over neighboring area features.

3 The road collapse algorithm

The road collapse algorithm uses triangulation to find the skeleton of the road and to restore the polygonal base map. The algorithm consists of 5 phases. The phases are described below and illustrated in Figure 2. Figure 2a shows the initial map, which represents a road (grey) enclosed by three white polygons.

Triangulation (see Figure 2b) First, a triangulation of the road polygon is constructed. Any triangulation of the polygon will result in a skeleton, however, a Delauney triangulation, that is constrained or conformed to the polygon boundaries, yields the best results. This is a skeleton that is close to the hartline of the road. The triangles in the triangulation may have 0, 1, or 2 edges that coincide with the boundary of the polygon. We will call them 0-, 1-, and 2-triangles. The edges that coincide with the boundary will be called boundary edges, the others are internal edges.

Construction of the skeleton (see Figure 2c) The middles of the internal edges and their connections form the skeleton. So, in each 1-triangle, the middles of both internal edges are connected by a line segment. In the 0-triangles, the middles of the internal edges (which are all three edges) are calculated. There are three possible segments to connect them. The shortest two of these are chosen. A 2-triangles adds a point to the skeleton. As can be seen from Figure 2c, the segments drawn in the road polygon in this way (dotted line in the figure) form a skeleton for the polygon. Also note that a 2-triangle marks an endpoint of the road, a 1-triangle is part of a linear piece of the road, and a 0-triangle marks a three-way junction. Two adjacent 0-triangles mark a 4-way junction, and three adjacent 0-triangles mark a 5-way junction, etc. The skeleton that is produced in this way is marked as a line feature and it is labeled with the label that was associated with the original road area feature.

Integration of the skeleton in the triangulation (see Figure 2d) The introduction of the skeleton leaves a subdivision of the road polygon in triangles and quadruples. Each quadruple is split into two triangles by inserting the shortest diagonal. In this way, the skeleton is integrated in the triangulation of the road polygon. We are now ready to distribute the area covered by the road over the neighboring area features.

Restoration of the polygonal base map (see Figure 2e) The triangles making up the road are now distributed over the neighboring area features. Each triangle is added to the feature with which it shares the longest edge that is not part of the skeleton.

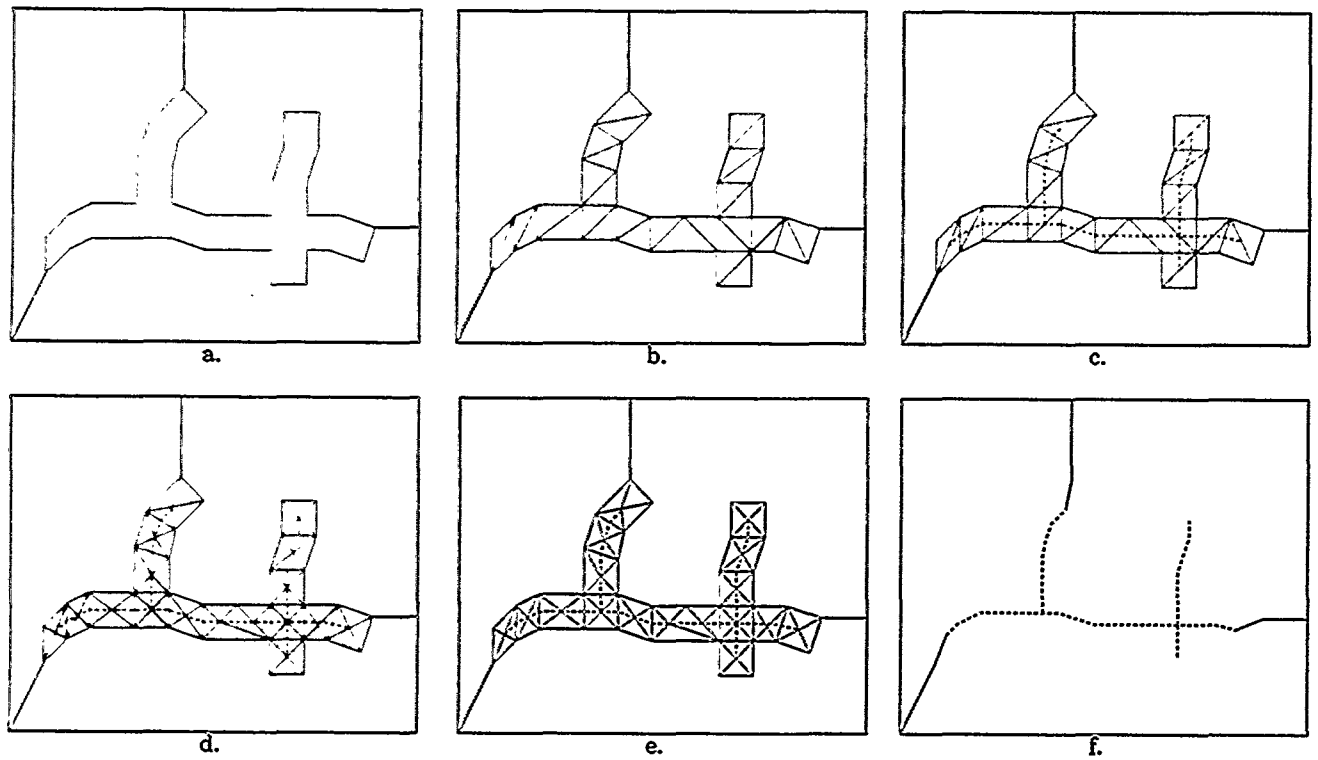


Figure 2: Road collapse

Cleaning up (see Figure 2f) Finally, the edges that are not needed anymore (the boundary of the original polygon), and the internal edges of the triangulation are removed from the database. Now, the road collapse algorithm is ready.

3.1 Integration of the skeletons

Application of the collapse algorithm to a collection of roads results in a collection of skeletons, one for each road, that are not connected. This is caused by the fact that, as can be seen in Figure 2d, the skeletons do not fully reach the endpoints of the roads. The skeletons produced may easily be connected, though, by adding those boundaries from the polygonal map that connect two skeleton endpoints, to the collection of skeletons [Zwo98]. In this way, a closed road network is produced. An alternative way to produce a closed road network is to first integrate all roads that need to be collapsed into one area feature in the map and then apply the collapse algorithm to the result to produce a single closed road network.

4 Road collapse in Magnum

The road collapse algorithm has been implemented in the Magnum DBMS for spatial applications [BQK96,BWK98, WQB97]. This section describes the implementation. First, the Magnum prototype environment is described. Then we show how a suitable datastructure for polygonal maps and triangulations, the Double Connected Edge List (DCEL) [BKO97] is implemented as a structure in the Magnum prototype DBMS. Finally, we show how this datastructure is used to support the road collapse algorithm.

4.1 The Magnum DBMS

Magnum is a DBMS prototype that is developed by two cooperating universities in The Netherlands. It is a structurally Object Oriented DBMS that is specially suited for the management of spatial data. As a DBMS, the system is able to manage all sorts of data. However, the fact that spatial data is managed by the system has influenced its design. The eventual goal of the Magnum research project is to study efficient integration of a wide range of complex GIS-functionality in a general DBMS. Magnum has successfully been extended with a wide range of spatial base types [BQK96] and it performs well on the Sequoia benchmark [SFG93]. We equipped the system with a structurally Object Oriented data model and implemented the Object Algebra MOA [BWK98]. In this paper, we illustrate the structural extensibility of the system with the implementation of the road collapse algorithm. This section will sketch the the architecture of Magnum as far as needed to understand the implementation of the road collapse algorithm.

4.1.1 Architecture

Figure 3 shows the architecture of the prototype. The system is built in a modular extensible way. Monet, the storage server is implemented in C, all other components are in implemented in Java. The functionality of the components in Figure 3 is as follows.

User Interface The User Interface, at the top of the architecture provides an interactive interface to users. Users have access to the various tools that have been built in the system.

Monet Monet is the storage system used in Magnum. Monet is described in more detail below.

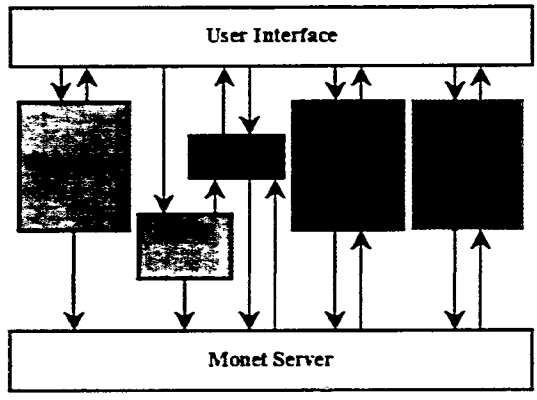


Figure 3: The architecture of the Magnum prototype DBMS

MOA The Magnum Object Algebra, implements a structurally Object Oriented data model and an algebraic query interface to the data model. The datamodel allows base type and structural extensibility.

TextTool The text tool provides a standard textual interface to MOA and Monet.

MapTool The maptool is the mapping interface to the system. It allows the user to present query results as a flat map that consists of several superimposed layers. MapTool supports an extensible legend system. A legend is a software module that specifies how data is to be presented on the map. Users may add new legend's to the system for specialized graphical presentation of query results.

ReliefTool This tool is currently being constructed. It allows 3-dimensional presentation of digital terrains.

MonetTool The monet tool allows the user to directly access the Monet physical database. This tool is meant for debugging only.

Essential for the work described in this paper are the binary relational data storage of Monet, and the extensible object data model in MOA.

4.1.2 Monet, a binary relational database system

Monet is a high-performance [BQK96,BWK98] extensible parallel database kernel that has been developed at the UvA and CWI since 1993. Monet implements a binary relational data model, in which all data is stored in binary tables. Structured data is decomposed over narrow tables [CoK85]. This fragmentation helps reduce chunk sizes to fit memory and saves a lot of IO. Monet accesses only those attributes that are actually used in a query. The interface to Monet consists of an algebra on binary tables, called the binary algebra. This algebra comes down to a simplified version of the relational algebra, adjusted to work on binary tables only. The simplicity of the data model and the algebraic interface allows simple and very efficient implementations of the operations in the algebra. The superior performance of Monet has been demonstrated in various contexts [BQK96, BWK98]. In this paper, we use Monet's binary tables to store the highly structured data that describe a polygonal map.

4.1.3 An extensible object data model

MOA provides the user with a structurally object oriented data model that can be extended in two ways. First, the system can be extended with new *base types* and their access functions. This sort of extensibility is quite common in modern DBMSs [BQK96,DKL94,Gue89,ScV92,SRH90]. Magnum's base type extensibility is implemented by Monet. It is used to provide primitive spatial types like points, lines and polygons, and large set of spatial operations on the primitive spatial type. A description of the spatial extensions to Monet is found in [BQK96].

Magnum also provides a novel feature, which we call structural extensibility. It is possible to extend the data model with structuring primitives, other than then the conventional tuple, set, list, etc [Cat94]. So, in fact, the structural extensibility allows users to extend the data model supported by the system. We use structural extensibility in the context of spatial applications to support structures like polygonal maps, triangulations, and rasters.

MOA implements the structures in the Magnum data model, the standard ones and the extensions, via mapping structured data on Monet's binary datamodels. This means that highly structured data is decomposed over many binary tables. The structural extensibility is implemented by MOA via the implementation of extensions to this mapping of structured data on binary tables. A detailed description and a formalization of the mapping process is found in [BWK98].

In this paper, we extend the Magnum data model with the FeatureMap. To do so, we show how Polygonal Maps are stored in Monet's binary tables. Also, operations on the Polygonal Map are translated into binary algebra programs. In this paper, we focus on the implementation of the collapse algorithm.

Example

This section illustrates the data modeling and the integrated query facilities of the system. Space limitations prohibit to give an extensive description of the model and query language. The standard data model provided is similar to the structural part of the ODMG model [Cat94]. The query language [BWK98] is a standard object algebra. It contains the select, project, join, semijoin, union, intersection, difference, subset, in, nest, unnest, and aggregates that operate on sets; it allows access to attributes of tuples and objects.

A database around the TOP10Vector base map may look as follows:

```
P : FeatureMap

class Map : TUPLE<code : int,
                  owner : Person,
                  feature : int>

class Person : TUPLE< name : string,
                      age  : int >
```

This code segment states that there is a variable P referring to a polygonal map. FeatureMap is a structure that has been added to the system as a structural extension.[2] Then, there is a class Map with TUPLE structured objects. The tuples have three attributes, one contains a describing code, one describes the owner, and the last one contains the

[2]We assume that a filled database is available. Currently the data is loaded via a loader directly into the storage system Monet.

identifier of an area feature in P. Class Person describes the people who may be owning land.

Let's now assume that we want to collapse all streets, that is all area features with code 3533. The algebra expression

```
map[TUPLE<code,
        collapse(P, feature)>](
          select[code = 3533](Map))
```

first selects those tuples from the map that describe street and then collapses the area feature from the selection. The result consist of a set of TUPLES containing the describing code and the identifier of a line feature in P. This piece of code uses the collapse function on extension structure FeatureMap. The implementation of this function is described below.

Another example is a query that wants to know the areas of the features owned by old people:

```
map[TUPLE<code,
       owner,
       area(P, feature)>](
         select[owner.age > 65](Map))
```

Here we assume that a function area is implemented on extension structure FeatureMap.

The examples in this section show how the extension structure FeatureMap and its operations is integrated in a standard object algebra.

4.2 A datastructure for Polygonal Maps

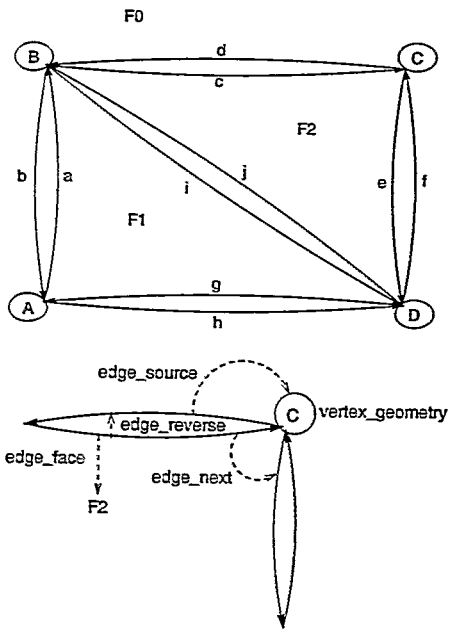


Figure 4: A Doubly Connected Edge List

The Doubly Connected Edge List (DCEL) is a well known main memory data structure to store and manipulate polygonal maps. An extensive and formalized description is found in [BKO97]. The data structure records data about the vertices, the edges, and the faces in the subdivision (see Figure 4. Vertices are points in the plane, edges are straight line segments connecting two vertices. A face is part of the plane enclosed by edges, the interior of which does not contain edges or vertices. In a DCEL, each edge is represented by two half-edges, directed in opposite directions. The datastructure records for each vertex, the geometry of its point. For each edge, the reverse edge, the next edge, the source vertex, and the right-side face are recorded as illustrated in the right-hand diagram of Figure 4.

vertex-geometry	
A	(0.000000,0.000000)
B	(0.000000,100.000000)
C	(200.000000,100.000000)
D	(200.000000,0.000000)

edge-src	
a	A
b	B
c	B
d	C
e	C
f	D
g	D
h	A
i	B
j	D

edge-next	
d	b
g	a
f	d
j	c
h	f
c	e
b	h
i	g
e	j
a	i

edge-rev	
a	b
b	a
c	d
d	c
e	f
f	e
g	h
h	g
i	j
j	i

edge-face	
h	F0
d	F0
b	F0
f	F0
g	F1
a	F1
j	F2
c	F2
i	F1
e	F2

Figure 5: Storage of a DCEL in Monet's binary tables

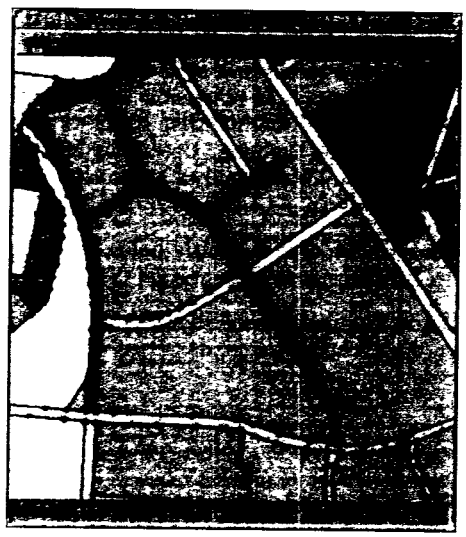


Figure 6: Data set used in the example

In our prototype, Monet's binary tables are used to store a DCEL, as described in [QuK98]. To do this, each vertex, each half-edge, and each face gets a unique identifier. In this text, we will use capital letters to indicate an identifier associated with a vertex, small letters are used for half-edges, and faces are indicated with a number prepended with F. Figure 5 shows how the DCEL from Figure 4 is stored in Monet's binary tables. Note, that identifier F0 indicates the infinite outer face.

A Feature Map is modeled in a level on top of the DCEL. A number of adjacent faces may form an area feature. Each area feature gets its own identifier and there is an binary table called `feat-face` that records which face belongs to which feature. In a polygonal map, each face belongs to exactly one feature. Often, an area feature is represented with a single face. In that case, the Feature Map is called

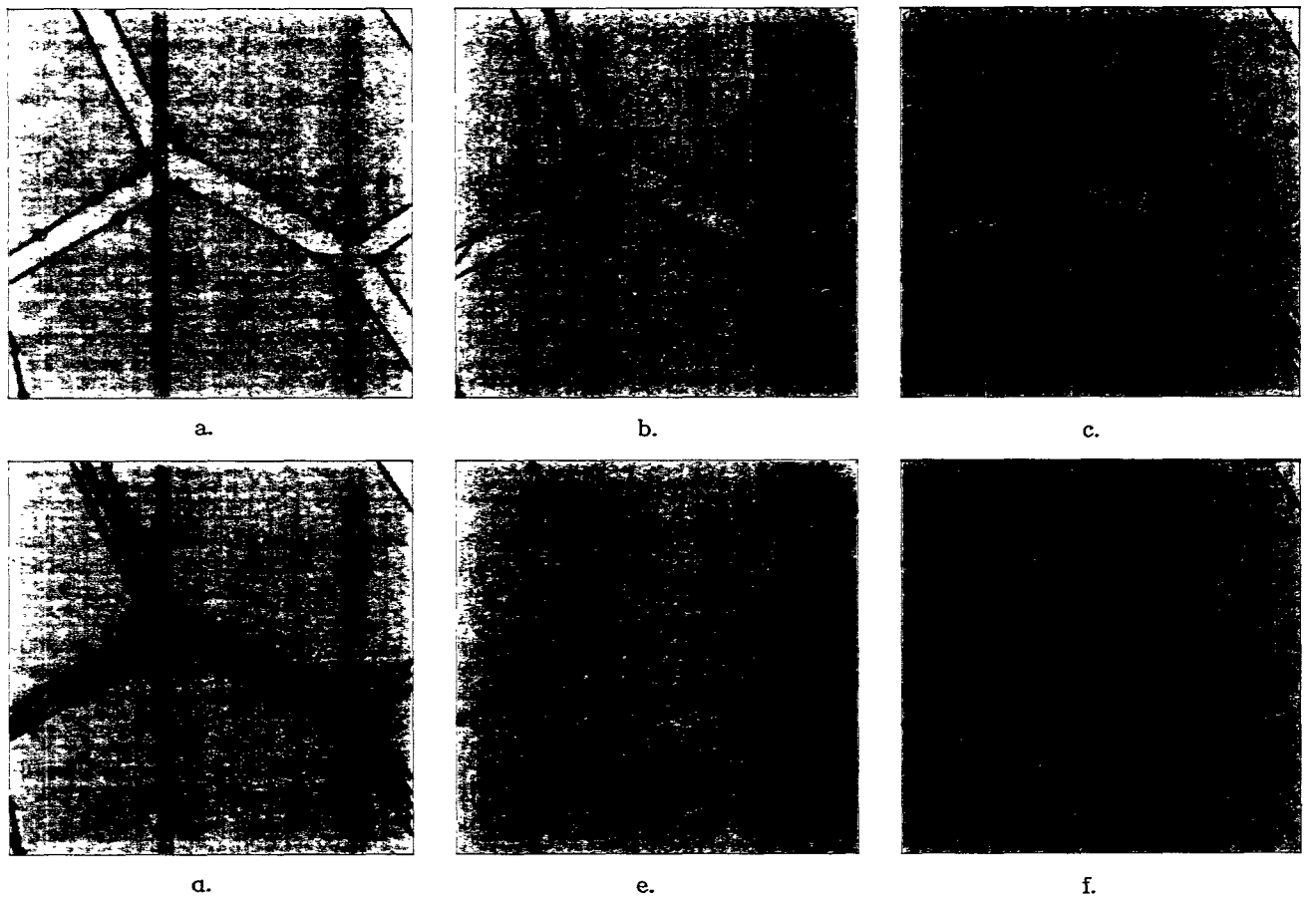


Figure 7: Road collapse as performed by Magnum

simple. Line features that are integrated in the polygonal map are represented by a number of connected edges. Again, line feature gets a unique identifier, and there is a binary table called `feat-edge` that records which edge belongs to which line feature.

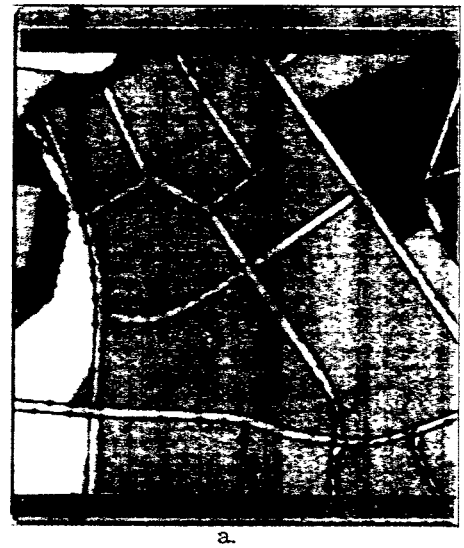
So, in total, 7 binary tables are used to store a Feature Map, 5 for the underlying DCEL, and 2 for the area features and line features. We implemented a variety of operation on the Feature Map. Among those operations are simple ones: split an edge at a new vertex, merge two adjacent faces into one, and there are more complex ones like collapsing an area feature. As the data belonging to the Feature Map is stored in Monet's binary tables, all Feature Map operations result in updates on those binary tables. Many of those updates use Monets set oriented executions techniques. A more detailed description is found in [Zwo98].

4.3 The road collapse algorithm

The implementation of the road collapse algorithm is illustrated with the small part of the TDN-TOP10Vector map that shows the streets in the Dutch village "Boekelo" (see Figure 8a). This figure shows the MapTool presentation of the database. In the figure, area features are filled with different grayscales or colors. The DCEL edges are presented with lines, and the vertices are indicated with tiny circles. The roads are easily recognizable in the map. We illustrate the collapse process on the road indicated dark in Figure 6 and zoom in to the box indicated in that figure. Figure 7a shows the starting point, which is a simple Feature Map. In Figure 7b the road feature is triangulated. Obviously, this triangulation results in updates on the binary tables that store the DCEL and the Feature Map. The road feature is not simple anymore because it consists of a number of triangular faces. The creation of the skeleton and the restoration of the triangulation is done in one step, in which the triangular faces are split in 2, 3, or 4 triangles. The result of this step is in part c of the figure. Remark, that in the steps described until here, three different representations of the same polygonal map are computed. Figure 7d shows how the road triangles are distributed over the neighboring area features. This step does not affect the DCEL, only the association of faces to features in the Feature Map is updated. Yet, this step changes the polygonal map in the way needed to collapse the road. This steps results in the neighboring faces to consist of their original face, and a number of triangles that originally belonged to the road. In the following step, the faces that make up a single feature are integrated into a single face (e). Finally (Figure 7f), the resulting roads may be simplified via a Douglas-Peucker Line-Simplification Algorithm [HeS92,ViW95]

4.4 Results

In Figure 8 the polygonal base map of Boekelo is used for the road collapse algorithm. Part *a* of the figure shows the original base map, and in part *b* the result after the road collapse algorithm is showed. Part *b* is the result of executing query (see the example in Section 4.1.3)



a.

b.

Figure 8: Road collapse on the roads in the Dutch village "Boekelo"

```
map[TUPLE<code,
      simplify(collapse(P, feature))>](
        select[code = 3533](Map)).
```

After applying the algorithm, all the road areas have disappeared and all the surrounding areas have been extended so that a new polygonal base map is constructed. The vertices and edges, belonging to the old road areas have been removed and the new vertices and edges belonging to the boundary of the areas have been inserted. Although the number of area features has been reduced, the size of the database is only reduced slightly. As can be seen in Figure 7 the vertices on the resulting hartline are very densely located. In fact, the algorithm almost doubles the density of vertices along the hartline, compared to the density of vertices along the boundary of original road area feature. Therefore we used a line simplification algorithm to reduce the density of vertices. This algorithm significantly reduces the density of the vertices, and thus the size of the database, even if applied with a very tight tolerance.

We plan to do an extensive evaluation of the performance of the collapse implementation in Magnum. Just to give an indication of various aspects of the performance we give the following figures. The original map of Boekelo consists of 669 vertices. There are 15 road polygons in the map. The road collapse algorithm takes about 4 seconds [3]. The resulting map contains 648 vertices. Applying a Douglas-Peucker algorithm on the result with a tolerance of 50 cm reduces the number of vertices to 423.

This paper presents the result of a straightforwardly implemented fairly simple algorithm for the road collapse. The simplicity results in a robust implementation which also is fast. In most cases, the results of the algorithm are quite good, however, sometimes the results leaves room for improvement. This is especially true for road junctions, as the reader should be able to find out from Figures 7 and 8. We plan to experiment with various extensions to the straightforward base algorithm. Basically, there are three ways to improve the result:

- Increase the density of vertices along the boundary of polygons that need to be collapsed. A careful study of the "Boekelo" map, shows that anomalies in the result may be caused by a low density of vertices along the road boundaries. In the DCEL structure it is quite simple to split long edges prior to applying the road collapse algorithm. We expect better results, obviously at the cost of more processing time.

- Adjust the algorithm. We chose to split 0-triangles along the two shortest segments that connect the middles of its edges. In the model map in Figure 2, this is the right choice, but in the junctions in Figure 7, we may also connect the 0-triangle vertices to the point of gravity of this triangle to produce a better result. It might be a good idea to have the algorithm choose between these two and may be other choices.

- Improve the result. The final option is to recognize bad solutions in the result and correct them. The problem in this option is that it may be hard to find out whether a presumed anomaly is produced by the algorithm or whether it is an odd feature of the map.

5 Conclusions and Future work

In this paper, we described the implementation of a road collapse algorithm in Magnum, an extensible object oriented DBMS. The goal of the Magnum project is to provide true and efficient integration of spatial functionality in a DBMS. The contribution of this paper is twofold. First, we showed that the collapse algorithm, which we regard to be an example of complex spatial functionality, can be integrated in Magnum. Second, we think that the algorithm per se provides a new piece for the complex map generalization puzzle.

The main achievement of this paper is the fact that we showed that complex spatial functionality, in this case generalization in a topological environment, can be integrated in a DBMS in a natural way. To do this, we developed an extensible object data model. The data model is extensible at the level of base types, and this sort of extensibility is used in Magnum to provide support for atomic spatial types, like points, lines and polygons. As a novel feature, we also supply structural extensibility. This feature may be used to support structures other than the standard SET, TUPLE,

[3] For these experiments, Magnum runs on a SUN SPARCstation 20 with 128 Mbytes of memory

and LIST, which are the main structural building blocks for many object-oriented data models. In this paper, we show how a new structure, the FeatureMap, may be supported in Magnum. We illustrated its use with the implementation of a collapse algorithm on area features in such a map.

The collapse algorithm described in this paper is based on triangulation. It is a simple straightforward algorithm that allows robust and efficient implementation and the result is satisfactory. As described in the previous section, the results of the algorithm leave room for improvement and obviously this is on our research agenda. Also, we plan to do an extensive evaluation of the performance of the algorithm. Finally, we will use our approach to support other complex spatial functionality in the Magnum OO DBMS.

References

[BaW97] M. Bader & R. Weibel, "Detecting and Resolving Size and Proximity Conflicts in the Generalization of Polygonal Maps," in *Proceedings of 18th International Cartographic Conference, Stockholm, Sweden, 1997.*

[BKO97] M. de Berg, M. van Kreveld, M. Overmars & O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, New York–Heidelberg–Berlin, 1997.

[BQK96] P. A. Boncz, C. W. Quak & M. L. Kersten, "Monet and its Geographic extensions," in *Proceedings of the 1996 EDBT Conference, Avignon, France.*

[BWK98] P. A. Boncz, A. N. Wilschut & M. L. Kersten, "Flattening an Object Algebra to Provide Performance," in *Proceedings of the 14th International Conference on Data and Knowledge Engineering, Orlando, Florida, USA, February 1998.*

[BrB96] A. Bregt & J. Bulens, "Application-Oriented Generalization of Area Objects," in *Methods for the Generalization of Geo-Databases, Netherlands Geodetic Commission, Delft, The Netherlands, February 1996*, 57–64.

[Cat94] R. G. G. Cattell, ed., *The Object Database Standard: ODMG-93*, Morgan Kaufmann Publishers, San Mateo, California, USA, 1994.

[CoK85] G. P. Copeland & S. N. Koshafian, "A decomposition storage model.," in *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data, Austin, TX, May 28–31, 1985*, 268–279.

[DKL94] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel & J. B. Yu, "Client-Server Paradise," in *Proceedings of Twentieth International Conference on Very Large Data Bases, Santiago, Chile, September 12–15, 1994.*

[DeP96] G. Dettori & E. Puppo, "How Generalization Interacts with the Topological and Metric Structure of Maps," in *Proceedings of 7th International Symposium on Spatial Data Handling, SDH'96, Delft, The Netherlands, 1996*, 9A27–9A38.

[Gue89] R. H. Gueting, "Gral: an extensible relational database system for geometric applications," in *Proceedings of Fifteenth International Conference on Very Large Data Bases, Amsterdam, The Netherlands, August 22–25, 1989.*

[HeS92] J. Hershberger & J. Snoeyink, "Speeding Up the Douglas-Peucker Line-Simplification Algorithm," in *Proceedings of 5th International Symposium on Spatial Data Handling, SDH'92, 1992*, 134–143.

[JBW95] C. B. Jones, G. L. Bundy & J. M. Ware, "Map generalization with triangulated data structures," *Cartography and Geographic Information Systems* 22 (October 1995), 317–331.

[OoV94] P. van Oosterom & T. Vijlbrief, "Integrating complex spatial analysis functions in an extensible GIS," in *Proceedings of the 6th International Symposium on Spatial Data Handling, Edinburgh, Scotland, September 1994*, 277–296.

[PTM96] W. Peng, K. Tempfli & M. Molenaar, "Automated Generalization in a GIS Context," in *Proceedings of International Symposium on Remote Sensing, Geographic Information Systems and Global Positioning Systems, GEOINFORMATICS'96, Florida USA, 1996.*

[QuK98] W. Quak & M. Kersten, "Combining Computational Geometry and Databases," technical report, University of Amsterdam, The Netherlands, submitted to the BNCOD 1998 conference, 1998.

[RuP96] A. Ruas & C. Plazanet, "Strategies for automated generalization," in *Proceedings of 7th International Symposium on Spatial Data Handling, SDH'96, Delft, The Netherlands, 1996*, 601–619.

[ScV92] M. Scholl & A. Voisard, "Geographic Applications: An experience with O_2," in *Building an Object-Oriented database System. The story of O_2*, F. Bancilhon, C. Delobel & P. Kanellakis,eds., Mergan Kaufmann, San Mateo, California, 1992.

[SFG93] M. Stonebraker, J. Frew, K. Gardels & J. Meredith, "The Sequioa 2000 storage benchmark," in *Proceedings of ACM-SIGMOD 1993 International Conference on Management of Data, Washington, DC, May 26–28, 1993*, 2 – 11.

[SRH90] M. Stonebraker, L. A. Rowe & M. Hirohama, "The implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering* 2 (March 1990).

[ViW95] M. Visvalingam & P. J. Williamson, "Simplification and Generalization of Large Scale Data for Roads: A Comparison of Two Filtering Algorithms ," in *Cartography and Geographic Information Systems, volume 22 number 4, American Congress on Surveying and Mapping, Bethesda, Maryland, USA, 1995.*

[Wei96a] R. Weibel, "A typology of constraints to line simplification," in *Proceedings of 7th International Symposium on Spatial Data Handling, SDH'96, Delft, The Netherlands, 1996*, 9A1–9A14.

[Wei96b] R. Weibel, "Generalization of Spatial Data," in *Course Notes for the CISM Advanced School on Algorithmic Foundations of Geographic Information Systems, 1996*, 1–44.

[WQB97] A. N. Wilschut, C. W. Quak & P. A. Boncz, "Magnum, an Object-Oriented DBMS for spatial applications," *Nexpri INFO* (june 1997).

[Zwo98] R. van Zwol, *Generalisatie van GIS data*, MSc-Thesis, University of Twente, 1998.