Simultaneous & topologically-safe line simplification for a variable-scale planar partition

Martijn Meijers

Delft University of Technology (OTB – Department of GIS Technology)
Delft, the Netherlands
b.m.meijers@tudelft.nl

Abstract. We employ a batch generalization process for obtaining a variable-scale planar partition. We describe an algorithm to simplify the boundary lines after a map generalization operation (either a merge or a split operation) has been applied on a polygonal area and its neighbours. The simplification is performed simultaneously on the resulting boundaries of the new polygonal areas that replace the areas that were processed. As the simplification strategy has to keep the planar partition valid, we define what we consider to be a valid planar partition (among other requirements, no zero-sized areas and no unwanted intersections in the boundary polylines). Furthermore, we analyse the effects of the line simplification for the content of the data structures in which the planar partition is stored.

1 Introduction

A planar partition is a tessellation of the 2D plane into polygonal areas. These polygonal areas form a complete covering of the domain without overlaps and gaps. To obtain a variable-scale planar partition stored in the topological Generalized Area Partition (tGAP) data structures, we employ an off-line map generalization process as a pre-processing step (before online use, see for more details Van Oosterom 2005; Meijers et al. 2009). The boundaries of the partition are stored in the form of polylines and topological references. The tGAP structure can be used in a networked context (e.g. the Internet) to supply a vector map at a variety of map scales

S.C.M. Geertman et al. (eds.), *Advancing Geoinformation Science for a Changing World*,
Lecture Notes in Geoinformation and Cartography 1, DOI 10.1007/978-3-642-19789-5_17,

(many more than usually stored in a Multi-Resolution Database (MRDB)). Initially, for storing the geometry of the boundaries, the use of a forest of Binary Line Generalisation (BLG) trees was proposed (Van Oosterom 2005). A BLG tree (Van Oosterom 1990) is a binary tree and stores the result of the Douglas-Peucker line simplification algorithm. Advantages of employing the forest of BLG trees would be that the data structure would contain as little geometric redundancy as possible. However, the Douglas-Peucker algorithm does not give any guarantees on topological correctness and we noticed that the use of the BLG trees would create communication overhead in a situation where a client application retrieves the map data from a server when the trees are only partially transmitted to obtain the right amount of vertices in the polylines.

An alternative we investigated was not to simplify the boundaries at all, but to keep the original geometry of the boundaries with which we started. We quickly noticed that during use of the resulting variable-scale planar partition in a network context, the number of vertices in the boundaries was too high, especially for the smaller map scales, leading to a slow performing user interface.

Therefore, we turned back to simplify the boundaries, but now store the result of the simplification explicitly (thus *not* deriving the geometry dynamically from a special reactive data structure, like with the BLG trees, as this leads to administrative overhead) and allow some redundancy in the data structure (but preferably as minimal as possible). The line simplification has to be performed without violating any of the requirements for a valid variable-scale planar partition. In this paper, we give an overview of how we perform the simplification on a *subset* of the boundary polylines in the planar partition (the resulting lines from a higher level generalization operation, such as aggregation or splitting a polygonal area over its neighbours). The removal of points leads to short cuts in the polylines. It is ensured that taking such a short cut does not lead to topological problems (the boundaries are not allowed to change their relative position). However, we also observed that a spatial configuration that leads to a problem (e.g. a change of side or occurrence of an intersection) at first might be changed later, because another point has been removed. Our approach also deals with these issues. The simplification is stopped when a certain criterion is reached (enough points have been removed or there are no more points that can be removed without violating the requirements for a valid planar partition).

The research questions that we try to answer are:

- How to prevent topological errors and what are sufficient conditions to guarantee topological correctness in a variable-scale environment Plümer and Gröger 1997)?

- Which lines do we simplify after applying a generalization operator on the polygonal areas?
- When to stop the simplification?
- What are the effects on the contents of the data structures when applying the line simplification?

The remainder of the paper is structured as follows. We review related work in Section 2. In Section 3, we formalize the requirements for a variable-scale planar partition. In Section 4, we improve a known method for topologically safe simplification for more than one polyline as input (cf. Kulik et al. 2005). The input polylines will be simplified simultaneously – thus not one after the other. Our improvements focus on an efficient implementation, keeping the polygonal areas explicitly valid (note that the areas can contain holes in their interior) and the tGAP structure as context. Furthermore, we use a step-wise generalization process in which we only simplify a subset of the boundaries, thus not all polylines will be simplified at the same time. Section 5 shows how we tested and analysed the algorithm. Section 6 concludes the work and gives some suggestions for future work.

2 Related work

In literature, a multiplicity of methods is known to simplify (cartographic) lines. Saalfeld (1999) gives a classification of polyline simplification methods:

in vacuo modifies one polyline in isolation, possibly leading to topological conflicts that have to be resolved by post-processing;

en suite modifies a single polyline in context (looking at topological relationships with nearby features); and

en mass modifies the complete collection of polylines and all other features of a map, taking the topological relationships into consideration during adjustment.

Apart from the classification given by Saalfeld, the algorithms can be divided in two main groups: using *refinement* (i.e. an approach from coarse to fine, starting with a minimal approximation of a polyline and then adding the most significant points, until a prescribed tolerance is met) or using *decimation* (i.e. an approach which starts with the most detailed version of a polyline and then eliminates the least important points first, thus going from fine to coarse).

The most known algorithm for simplifying lines, *in vacuo* using a refinement approach, is the Douglas-Peucker line simplification (Ramer

1972; Douglas and Peucker 1973). It was modified by Saalfeld (1999) to work on a polyline *en suite*. Da Silva and Wu (2006) argued that topological errors could still occur and gave an extension to the suggested approach. However, their approach is not explicitly designed for keeping a planar partition valid as they cannot ensure that polygonal areas keep size.

Another *en suite* algorithm is developed by De Berg et al. (1998). The core of the algorithm is also used for simplifying polylines in a planar subdivision (*en mass*), but each polyline in the main loop of their algorithm is still simplified *en suite* (so the simplification outcome depends on the order of processing the polygonal chains).

A better approach in this respect is the one given by Kulik et al. (2005), which simplifies the polylines simultaneously (thus not one after the other). The basis for their recipe is the algorithm described by Visvalingam and Whyatt (1993). It is using decimation for simplifying lines *in vacuo*. The algorithm of Visvalingam and Whyatt was extended by Barkowsky et al. (2000) using different criteria for the order in which points will be removed (leading to different shapes as output). Kulik et al. (2005) developed the approach for simplifying polylines *en mass*, but they consider only a connected graph for the topology aware simplification (the algorithm in this paper also deals with an unconnected graph, in case of islands in the polygonal areas, e.g. face 4 in figure 1). Furthermore, in their description of the algorithm they show that it is necessary to check after every simplification step whether points that could not be removed before are now allowed to be simplified. It appears that their algorithm in this case can lead to quadratic running times. Also, it is not clear in their description how near points that might influence the simplification can be obtained efficiently in an implementation.

Dyken et al. (2009) also present a method for simultaneous simplification of a collection of polylines in the plane (simplifying them *en mass*). The method is based on building a triangulation. Although this approach seems promising, building a triangulation after every generalization operation will be expensive from a computational point of view, mainly because we already have a topological graph at hand.

It must be noted that none of the methods described above discuss line generalization in a stepwise generalization process, thus intermingled with other generalization operations, such as merging and splitting of polygonal areas (aggregation) in a planar partition for a variable-scale context.

3 A valid planar partition at a fixed and at a variable map scale

A planar partition can be fully described by storing a topological structure. Polylines form the boundaries of the polygonal areas; each polyline has two endpoints, which we call the *exterior* points, and the rest of the points of the polyline are called *interior* points. Following Gröger and Plümer (1997), we give a set of requirements for this boundary representation, so that the resulting model is a *valid* planar partition of polygonal areas. The mathematical concept of a graph G(N,E) consists of a set of nodes N and a set of edges E, where the edges define a relationship between the nodes. Let us consider the set of instances $n \in N$ and $e \in E$. If a relationship between a node n_0 and an edge e_0 exists, the two instances are called *incident*. The *degree* of a node is the number of its incident edges. If a node n_1 is also incident to an edge e_0, the nodes n_0 and n_1 are said to be *adjacent*. Using the graph concept, we can specify a set of requirements for the boundaries (as illustrated in figure 1):

1. With a graph $G(N,E)$, we model the geometric relationship of the endpoints of the polylines: when two endpoints have the exact same coordinates, they become a node in the graph; thus N is the set of geometrically unique endpoints and E the set of polylines. We embed G in the 2D plane by the location of the points of the polylines and we specify that G is a planar drawing. This implies that polylines are only allowed to touch at their endpoints, no intersections or overlaps are present and each polyline must be simple (also no self-intersections are allowed).
2. All nodes in G must have a *degree*>1. This prevents having dangling polylines as a boundary.
3. As a result of the fixed embedding of the graph, we can define each face f of G as the maximal connected region of the 2D plane where every two points x and y in f can be connected to each other without intersecting or touching any edge $e \in G$. The edges that delimit f form its boundary b. The edges $e \in b$ form one (or more) cycle(s). For each cycle there exists a path $n_0, e_0, n_1, e_1, ..., n_{i-1}$, in which endpoints and polylines alternate and where endpoint $n_0 = n_i$.
4. Each face is delimited by at least 1 cycle (holes in the interior of a face are thus allowed). If a face has more than 1 cycle, these cycles have to be nested properly geometrically (if this is the case, one of these cycles should contain all other cycles). This nesting can only be

1 level deep on account of the previous requirement ('connected interior').

5. Each polygonal area in the planar partition corresponds to exactly one face (thus no multi-part polygonal areas are allowed) and each face corresponds to exactly one polygonal area. This symmetry enforces that all polygonal areas form a complete covering of the domain, without overlaps.
6. In *G*, there exists one unbounded face (universe), which has exactly one cycle (geometrically containing the cycles of all other faces). Furthermore, the set of faces *f* is *F*.

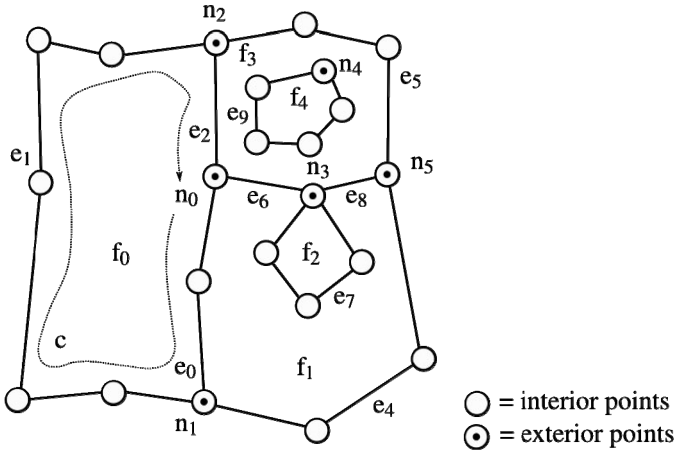


Fig. 1: Face f_0 is delimited by the boundary cycle *c* (i.e. the path $n_0, e_0, n_1, e_1, n_2, e_2, n_0$). The boundary cycle that delimits f_1 ($n_0, e_6, n_3, e_7, n_3, e_8, n_5, e_4, n_1, e_0, n_0$) is not simple (it passes through n_3 twice). However, f_1 forms 1 maximal connected region. Face f_3 is delimited by two cycles (one starting at n_5 and one starting at n_4), which are properly nested. Node n_4 has a *degree*=2.

From *G*(*N*,*E*) we can derive its dual *G**(*F'*,*E'*). This dual graph *G** models the *adjacency* relationships of the polygonal areas, i.e. in this graph *G** the faces *F'* form the nodes and the edges *E*' are the polylines one has to cross for neighbouring faces $f \in F'$.

So far, we were only concerned about the planar partition at one fixed map scale. We now extend the approach to a tessellation of the 2D plane at a variable (i.e. not pre-defined) map scale. For this, we need generalized

versions of the planar partition P. These versions with a lower level of detail can be seen as a third dimension. To obtain all versions, we iteratively apply an operation O on the planar partition P, which selects a polygonal area that should be generalized and modifies P accordingly, outputting P'. Symbolically, we can describe this as: $O(P) \rightarrow P'$. The generalization operator must ensure that the output it delivers (P') is again a valid planar partition that fulfils all requirements given above, e.g. plain removal of a polygonal area is thus not allowed, as this would violate the requirement of complete coverage of the domain (and would create a gap).

We allow two types of generalization operators to modify the number of faces in P: for a merge operation, we remove the boundary polyline(s) between a candidate and its most compatible neighbour and form a new boundary cycle for this neighbour; and for a split operation, we remove all boundaries associated with the polygonal area that is split, we introduce new boundaries between the neighbours of this area, and we form new boundary cycles for these neighbours. With both operations the dual graph G^* can be used to find the neighbours.

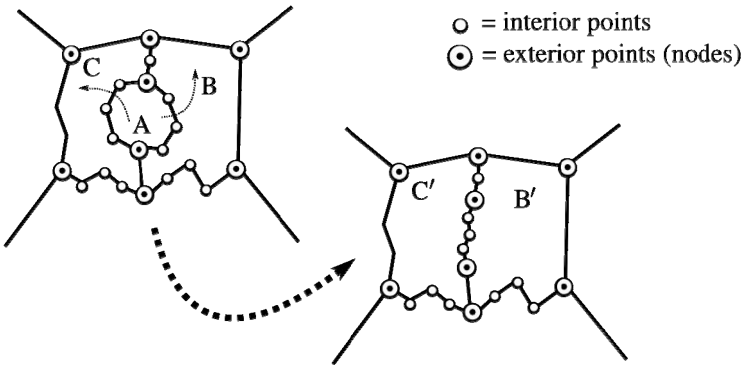
Based on the observations above, to have a variable-scale tessellation of the 2D plane, we add two more requirements to our list:

1. Every generalization operator O applied to P must output a *valid* planar partition P'.
2. Hierarchically speaking the new polygonal areas and boundaries from P' must be adjacent to and must not overlap with the remaining and unchanged areas and boundaries from P (in the map scale dimension).

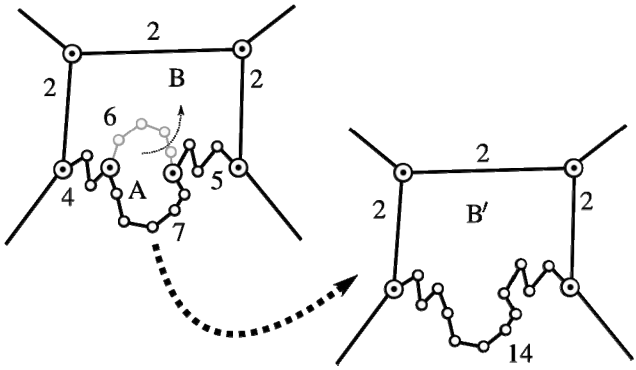
4 Line simplification

4.1 The need for line simplification

We use a stepwise map generalization process. This process records all states of the planar partition after applying a generalization operator in a tree structure. With the obtained hierarchy the average number of polygonal areas shown on a user's screen can be kept roughly equal, independent from the size of the user's view port (by varying the level of detail when a user zooms in or out, thus showing objects closer to or further from the top of the hierarchical structure). The removal of area objects (by merging or splitting them) leads to less objects per region. A user zooming out leads to an enlarged view port and ascending the hierarchy can supply an equal number of more generalized (thus larger) area objects to an end user, similar to the number before the zoom action.



(a) Splitting of polygonal areas leads to unwanted nodes



(b) Unwanted nodes also result from merging two polygonal areas. Furthermore, the average number of coordinates per boundary increases

Fig 2. Both (a) and (b) show that unwanted nodes can exist after a split or a merge operation. Furthermore, it is illustrated that not simplifying merged edges leads to an increased average number of coordinates per boundary.

However, the related boundaries of the polygonal objects will get more coordinates (per object) if the original boundary geometry is kept and not simplified. As can be observed in Figure 2, a split operation, e.g. implemented using triangulation, like in Bader and Weibel (1997), can lead to unnecessary nodes in the topology graph (nodes where degree = 2). This also happens when a merge operation is performed (see Figure 2(b)). Therefore, we merge the boundaries that are incident to those nodes. However, this merging leads to boundaries with more coordinates.

The increase in the number of coordinates is illustrated by the example shown in Figure 2(b). Polygonal areas A and B are merged. This leads to

two nodes with *degree*=2. On average, the number of coordinates before the area merge operation in the boundaries is (2+2+2+4+7+5+6)/7=28/7=4. After the merge, we can remove the two *degree*=2 nodes and thus merge the boundaries which leads to: 4+7+5–2=14 coordinates for this new boundary. On average the number of coordinates of all the boundaries is: (14+2+2+2)/4=20/4=5, which is more than before the merge operation. According to our rule that we want to keep the number of vertices per polyline equal, the polylines have to be simplified.

4.2 An overview of the simplification procedure

We employ a decimation approach for simplifying the selected boundary polylines. The order of removing points is determined by a weight value *w*, which we calculate for each interior point of the polylines to be simplified. For calculating the weight values, we get 3 consecutive points, p_{i-1}, p_i, p_{i+1} from a polyline forming a triangle τ. In our implementation the weight is calculated based on the area of the associated triangle τ, i.e. $\Box(p_{i-1}, p_i, p_{i+1})$, and therefore completely based on geometry (cf. Visvalingam and Whyatt 1993). There could be more geometrical criteria, like sharpness of turn angle, length of sides, ratio of sides, etcetera (alternatives are discussed in Barkowsky et al. 2000). Note that Kulik et al. (2005) also assign a 'semantic' weight per point (next to the 'geometric' weight), which they base on the feature class of the polyline, where the point belongs to and is also dependent on the user's application domain.

The exterior points of the polylines (forming a node in the planar partition) cannot be removed. At each step, the interior point p_i having the overall lowest weight value will be removed, leading to a 'collapse' of triangle τ into a *short cut* $\overline{p_{i-1,} p_{i+1}}$. Our simplification strategy has to obey the requirements that we have given for a planar partition, thus not all short cuts will be allowed. We observed that a spatial configuration that leads to a problem at first might be changed later, because another point has been removed (that was preventing a collapse). The algorithm re-tries removal of the blocked point in this case.

4.3 Dynamic and static polylines and how to select those

Two types of polylines that play a role in the simplification can be distinguished: *dynamic* polylines that will be simplified, i.e. interior points can be removed as long as no intersections or degeneracies in the planar parti-

tion requirements are caused by this removal; and *static* polylines that will not be simplified and for which all points are fixed (these points can forbid certain short cuts in lines that are simplified). Points of the first type are termed dynamic points and points of the second type are termed static points. Points that eventually will be removed by the simplification algorithm have to be interior and dynamic points.

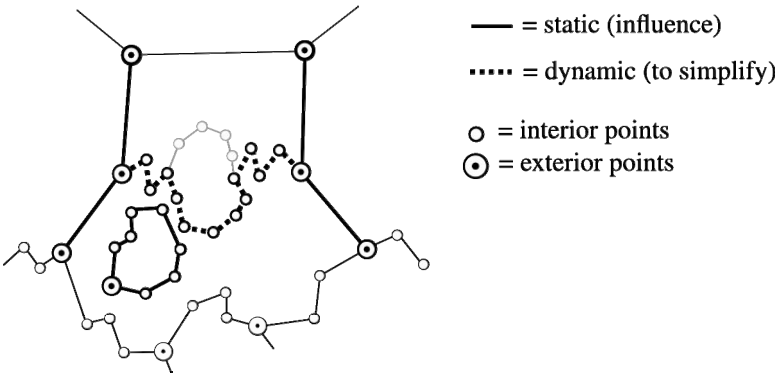


Fig. 3. Dynamic polylines will be simplified (only one in this figure), static polylines can have an influence on the simplification. Note that the alternative is illustrated in which only the polylines that are incident to a merge boundary will be simplified.

After a merge or split generalization operation is finished we must chose which lines to simplify (thus select the *dynamic* polylines). Two viable alternative approaches are:

1. Simplify the polylines that are (in case of an area merge operation) incident to the common merge boundaries or (in case of an area split operation) simplify the new boundaries that stem from the split operation; and
2. Simplify all polylines of the resulting new area(s).

As the simplification should be topology-aware, the *static* polylines in the neighbourhood also have to be selected as input for our algorithm as these can influence the outcome of the simplification. For this purpose, we can use the topology structure to select the lines that are in the vicinity of the lines that we want to simplify. We use the topology structure as an initial spatial filter (going from neighbouring areas to their related boundaries); then with a second pass we can select the related boundaries based on bounding box overlap with the union of the bounding box of all dynamic polylines. An alternative approach is to keep an auxiliary data structure (such as an R-tree or quad-tree) for fast selection of the polylines in

the vicinity. Downside of this approach is that an auxiliary structure needs to be kept, while the topology structure is already present. However, the initial filtering step using the topology structure can be expensive if the new polygonal area is at the border of the domain (leading to a selection of *all* edges at the border of the domain that have to be filtered on their bounding box).

4.4 A stop criterion for the simplification

We iteratively remove points from all dynamic input polylines, until a certain optimal goal is reached. We have two main choices for defining this optimal goal (to stop the simplification):

eps-stop Use a geometric measure as a threshold ε (all points having their weight $w<\varepsilon$ should be removed, where w is based on the size of the triangle of 3 consecutive points in the polyline).

Using this approach, we could use a fixed ε throughout the whole process of building the variable-scale hierarchy. This is not a very realistic option as the number of polygonal areas (and thus the level of detail) decreases when more generalization operators have been applied (when more polygonal areas have been merged or split, the remaining boundaries should also be simplified more). A better option is to determine dynamically the value of ε with every generalization step. For this we can:

- take the average or median value of all weight values as ε (all points having a weight value smaller than this have to be removed);
- set an ε based on other criteria, like the smallest segment length of all polylines taking part in the simplification. Such an alternative choice for ε also means that the weight values w for all interior points have to be calculated accordingly.

count-stop Use a fixed number of points that we want to see removed.

Using a fixed number of output points as the optimal goal, we can count the number of points in the input and try to remove a certain percentage. Two similar, but somewhat different options in this respect are:

- take a local approach: e.g. per input polyline try to remove half of the points (but do not remove more points from a polyline than half of its original points); or
- take a regional approach: for all polylines being simplified, count the total number of points and keep removing points, until in total half of these points have been removed.

Note that both approaches can leave more points as a result than wished for, because some of the points can be blocked by others (because topo-

logical errors must be prevented), although they fulfil the condition for removal (e.g. $w<\varepsilon$). Note also that with both approaches we can vary the percentage of points that we want to remove (instead of half of the points) depending on how far we want to 'push' the generalization. In an extreme case, we could set the percentage to such a value that the algorithm will try to remove all points leading to straight lines as much as possible (only topological 'problematic' points are remaining).

4.5 To prevent topological errors

The algorithm

An outline of the procedure is depicted in Algorithm 1. For all dynamic polylines, a doubly-linked list is created (storing the points in the order in which they are present in the original polyline, cf. Algorithm 1, line 1). Further, for all interior points of these polylines, a weight w is calculated. Important points get a higher weight than less important ones.

All dynamic and interior points are inserted in a priority queue Q, ordered by their weight values w (Algorithm 1, line 3). In our implementation we use a red-black tree (Guibas and Sedgewick 1978) for the priority queue. Points with equal weights are dealt with in the order of insertion. In-order traversal of the red-black tree Q allows now to find the point with the smallest weight value, which is then removed from Q. For point p_i, its neighbours, p_{i-1} and p_{i+1} can be retrieved from the polyline doubly-linked list. The three points together form the triangle τ (see Figure 4).

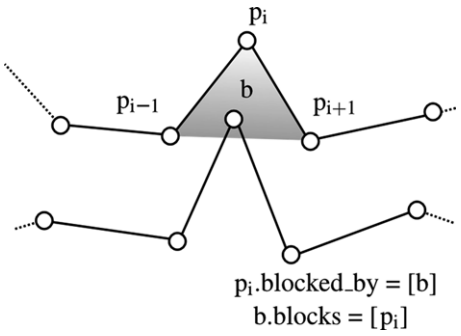


Fig. 4. As b blocks the removal of p_i, the blocks and blocked by lists are filled accordingly.

Algorithm 1 Simplification, while keeping a planar partition valid
Input: A set of dynamic polylines and a set of static polylines
Output: A set of simplified polylines

```
  {pre-processing}
1: Create doubly-linked list for each dynamic polyline
2: Compute weights w for all interior points of dynamic polylines
3: Add dynamic, interior points to priority queue Q based on weights
4: Create pointers between points of static polylines with only 2 points
5: Create kd-tree of all points of both dynamic and static polylines
  {simplifying}
6: while Q not empty do
7:      Pop least important p_i from Q
  {stop criterion, see section 4.4}
8:      if stop criterion met for p_i then
9:           break
10:     allowed ← True
11:     if p_i part of loop edge with 4 points then
12:          allowed ← False {no more 'tries' for this point}
13:     Retrieve τ (using p_i-1 and p_i+1 from linked list)
14:     vicinity ← search kd-tree for points near p_i using box of τ
15:     for all b ∈ vicinity do
16:          if b ∉ (p_i-1, p_i , p_i+1) and b part of segment p_i−1, p_i+1 then
17:               allowed ← False {no more 'tries' for this point}
18:     if allowed then
19:          for all b ∈ vicinity do
20:               if b ∉ (p_i-1, p_i , p_i+1) and b on τ then
21:                    allowed ← False
22:                    Append b to p_i.blocked_by list
23:                    Append p_i to b.blocks list
24:     if allowed then
25:          Remove p_i from linked list
26:          Adjust weights for p_i-1 and p_i+1
27:          Check whether p_i-1 and p_i+1 are still blocked, otherwise add to Q
28:          Mark p_i as removed in kd-tree
29:          for all u ∈ p_i.blocks do
30:               Remove p_i from u.blocked_by list
31:               if u.blocked by list empty then
32:                    Add u to Q
  {output}
33: return Simplified polylines by traversing doubly-linked lists
```

The short cut that will be taken is $\overline{p_{i-1}, p_{i+1}}$. Such a short cut is only allowed if it does not lead to an *invalid* planar partition, i.e. violates one of the requirements, as described in section 3. Any intersections of the new short cut with other polylines or another segment of the polyline itself (i.e. a line between two consecutive points of the polyline) have to be prevented. As the partition is valid to begin with (which can be ensured by us-

ing a constrained triangulation, see Ledoux and Meijers 2010; Ohori, 2010), the polylines of the planar partition do not contain any (self-) intersections. An intersection of the short cut can only be created when a segment σ 'enters' τ via the open side $\overline{p_{i-1,}p_{i+1}}$ (as it is not allowed to enter or leave the area of τ via either $\overline{p_{i-1,}p_i}$ or $\overline{p_{i,}p_{i+1}}$; this immediately would lead to an intersection). A point of σ must thus be interacting with τ for an intersection to happen and it is sufficient to check whether such a point exists, to prevent this. Points that can influence the collapse are termed *blockers*. These blockers stem from:

1. the polyline itself (self-intersection); or
2. other polylines in the vicinity of τ (both static and dynamic).

To efficiently find those points, we use a kd-tree (not just a regular kd-tree, but one following Bentley (1990) for which the tree does not need to be re-organised after removal of points, but to which no extra points can be added after initial organisation of the tree). All (interior as well as exterior) points of all polylines taking part in the simplification are inserted in this kd-tree (algorithm 1, line 5). The bounding rectangle around the triangle τ is used to query the kd-tree to find all points in the neighbourhood of this triangle to see if there are any blockers for creating the shortcut $\overline{p_{i-1,}p_{i+1}}$. If a blocker is found, the short cut is not taken and as p_i was removed from Q it will not turn up in the next iteration.

As the kd-tree contains the points of all dynamic polylines, a potential blocker b can be a point that forms the triangle τ. If this happens we do not check whether b blocks p_i (i.e. p_{i-1}, p_{i+1}, nor p_i itself can block removal of p_i) or no simplification could take place at all. Since a blocker b can be removed itself later on and then a short cut for this vertex p_i might be allowed, a cross reference is set up between p_i and b (b is registered in the 'blocked by'-list of p_i and p_i is registered in the 'blocks'-list of b).

If no blockers were found, p_i can be removed from the doubly-linked polyline list it belongs to (creating a short cut in this polyline). The point is also marked as removed in the kd-tree. If the removed point p_i was a blocker itself (having one or more points in its 'blocks' list), it removes itself from the 'blocked by' list of these particular points. If for a point u its 'blocked by' list becomes empty (because of the removal of p_i), u is placed back again in Q, so it has a chance of being a short cut in the next iteration

(if then not blocked by any other point and still not having fulfilled the condition for removal, e.g. having a weight $w<\varepsilon$). If one of the two neighbouring points p_{i-1} or p_{i+1} was blocked, it is also checked whether this is still the case (the shape of their related triangle also has changed, because of the short cut operation).

The algorithm ends when the chosen criterion has been met, i.e. there are no more points that can fulfil the criterion to reach the optimal goal, and the new polylines are returned.

More cases for validity

Apart from intersection prevention by testing near points, more specific situations have to be taken into account, because of the validity requirements of the planar partition. Two other conditions also have to be checked (illustrated in Figure 5) to prevent occurrence of zero-sized polygonal areas:

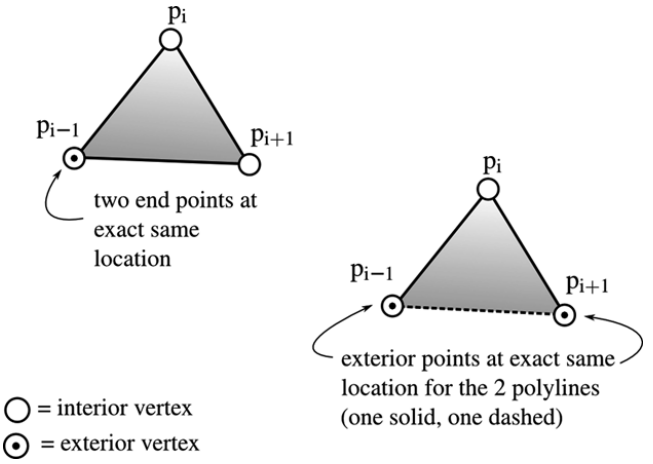


Fig. 5. If taking a short cut leads to a polygonal area that has no area, we put the end points as blockers for p_i. The result is that p_i is not removed, as the endpoints of the polylines will never be removed.

1. Same polyline (see Algorithm 1, line 11): A special check is performed when p_{i-1} or p_{i+1} is the endpoint of a so-called 'loop' polyline (a special case where the 2 exterior points of the polyline are at the exact same location, cf. Figure 5-left). We now have to check whether there will still be enough points in the polyline when we take away p_i (because no zero-size area is allowed). We can do this by travers-

ing the linked list and check when p_{i-1} is a loop endpoint, whether p_{i+2} is also such an endpoint (similar with p_{i-2} for p_{i+1}). Note that this is a rare case (only the two last interior points for a triangular face).

2. Different polyline (see Algorithm 1, line 16): Another check is performed on whether $\overline{p_{i-1}, p_{i+1}}$ is already connected by another polyline (by allowing twice such a polyline, a zero-size area would be created, Figure 5-right). To prevent this, it is necessary to check if a potential blocking point *b* returned by the kd-tree is part of such a polyline between p_{i-1} and p_{i+1}: a. for a dynamic point returned by the kd-tree, it is possible to use the doubly-linked list to navigate to the next vertex and check whether p_{i-1} and p_{i+1} are fixed; and b. for a static point returned by the kd-tree, we put an extra pointer to the other endpoint of the static polyline if the line only consists of two points. This allows checking whether a static point blocks the collapse of τ.

5 Experiments

We implemented the line simplification algorithm in our tGAP test environment. This environment is using the PostgreSQL[1] database system with PostGIS[2] extension. The algorithm is implemented in the Python[3] programming language. With the implementation we tried different alternatives.

Table 1. Symbolic names of the alternatives that were tested.

	Simplify which edges?	
Stop criterion?	Merged edges only at nodes with degree = 2	All edges of new area (including merged ones)
No simplification at all	none	none
Count stop (regional, half # of interior points)	m_ct	ct
Eps stop (median of all weights)	m_eps	eps
As far as possible	m_full	full

In total, we tested 7 alternatives — with only merge operations applied to the polygonal areas — for which the symbolic names are shown in Ta-

[1] www.postgresl.org
[2] postgis.refractions.net
[3] www.python.org

ble 1. The first alternative (labelled 'none' in Table 1) we tested, was merging edges at nodes with *degree*=2, but not applying any simplification. This was meant as a reference test as we already knew that this would lead to too many coordinates per boundary. The remaining strategies come from varying two alternatives: which lines to simplify (only the merged boundaries, prefixed with 'm_' in Table 1, or all boundaries of the new area); and when to stop the generalization (based on the median ε-value for all boundaries being simplified – dynamic eps-stop –based on the number of points – the regional count-stop approach, or simplify as far as possible, respectively labelled 'eps', 'ct' or 'full').

Table 2 shows the number of polylines and their average number of coordinates for the datasets we used in our experiment. We tested with three datasets representing different types of geographic data. We used a topographic, urban dataset; a topographic, rural dataset; and a land use dataset. Both topographic datasets represented infrastructure objects, which were not present in the land use dataset.

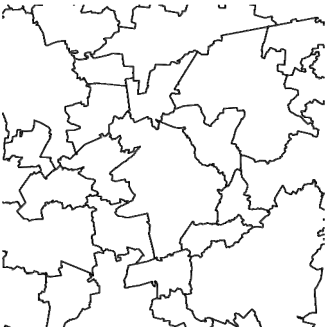
Table 2. For the datasets used in our experiment, the number of polygonal areas, polylines and average number of coordinates per polyline at start.

Dataset	# of areas at start	# of polylines at start	avg # coords per polyline	total # coords
Topographic, urban	9,381	24,528	4.6	112,828
Topographic, rural	3,286	8,212	10.6	87,047
Land use	5,537	16,592	7.2	119,462

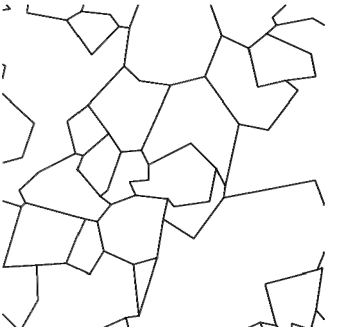
Figure 6 graphically shows some results of a few of the alternatives tested for the land use dataset. Figure 6(a) shows the result of keeping all original coordinates of the boundaries, thus not simplifying them. Tiny details and too many coordinates in the boundaries are the result. It can be seen in Figure 6(b) that the count-stop approach applied on *all* boundaries of the new area leads to a very simplified and coarse version. Both alternatives in which only the merged boundaries are simplified leave more details (see Figure (c) and (d)), where the count-stop approach is a bit more 'aggressive' than the eps-stop approach.

This is also illustrated by the graphs in Figure 7. In each graph, it is shown how many coordinates there are left for the total map, after every generalization step. As expected, the line at the top of the graph is the reference situation, where no coordinates are weeded. As already visually illustrated in Figure 6, it is also clear that the approach, where only the merged boundaries play a role in the simplification, is gentler in removing coordinates compared to when all edges of the new area will be simplified.

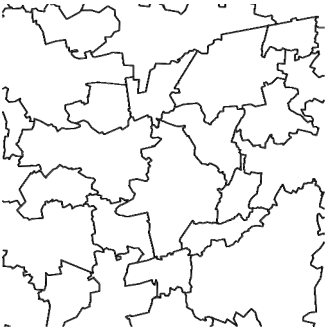
The main cause for this is that if all edges of the new area are simplified, they will be simplified more often, compared to the situation where only the merged edges are simplified (i.e. for every generalization step in which a polygonal area is the area to which a neighbour is merged, its boundary edges will again be simplified).



(a) No simplification (none)



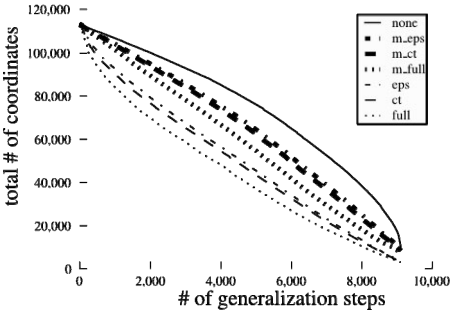
(b) Count stop for all edges of new area (ct)



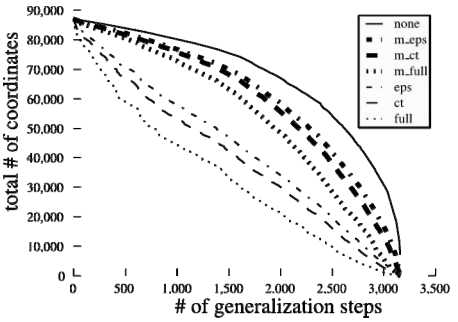
(c) Epsilon stop, using only merged edges (m_eps)

(d) Count stop, using only merged edges (m_ct)

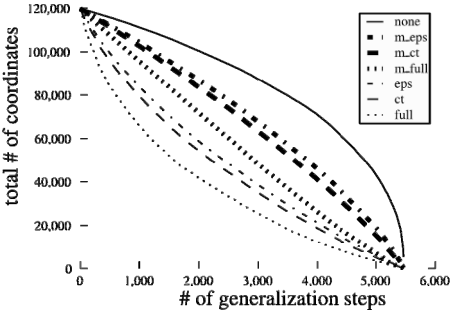
Fig. 6. From the land use dataset: 'Slices' of variable-scale data that show the result of the different alternatives for the line simplification, plotted at the same map scale (within brackets the symbolic name of the tested alternative). Note that the simplification of the boundaries changes the size of the areas and influences the order in which the areas are merged; therefore, the boundaries on the 4 maps do not exactly correspond to each other.



(a) Topographic, urban



(b) Topographic, rural



(c) Land use

Fig. 7. For each dataset, the graph shows the total number of coordinates for the complete map, in each generalization step (i.e. the number of coordinates in a 'slice' of variable-scale data).

Table 3 illustrates the fact that simplifying the boundaries over-and-over again also has a negative effect on the contents of the hierarchy. Although the graphs from Figure 7 show that there are less coordinates on average on every 'slice' derived from the variable-scale structures when all boundaries of a new face are simplified, the opposite is true for the contents of

the data structures. More coordinates need to be stored, because for every line that is simplified, a new version with the simplified geometry also has to be stored in the data structures (e.g. compare alternative 'm_ct' with 'ct'—in all cases more coordinates are stored for the 'ct' alternative). Therefore, simplifying only the merged edges is to be preferred over simplifying all the edges of a new area.

Table 3. Resulting number of polylines in the tGAP hierarchy with their average number of coordinates per polyline and the sum of coordinates in the total hierarchy.

(a) topographic, urban dataset

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
None	36,447	7.1	256,969
Ct	60,390	4.3	260,777
Eps	62,006	4.6	284,289
Full	55,084	3.7	205,870
m_ct	36,449	4.6	167,431
m_eps	36,438	4.8	176,350
m_full	36,403	3.8	139,187

(b) topographic, rural dataset

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
None	12,347	22.4	276,335
Ct	23,553	8.5	200,860
Eps	24,539	10.1	247,767
Full	19,640	6.7	131,538
M_ct	12,345	11.1	136,940
M_eps	12,343	13.0	160,066
M_full	12,349	7.8	96,665

(c) land use dataset

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
None	26,771	15.4	413,250
Ct	54,166	5.8	312,394
Eps	55,603	6.3	348,118
Full	45,040	4.8	216,174
M_ct	26,770	7.5	200,132
M_eps	26,768	8.4	223,623
M_full	26,769	5.3	141,019

6 Conclusion and future work

We described an algorithm to simplify simultaneously a subset of polylines in a planar partition in a variable-scale context. For this, we formalized what we consider a valid variable-scale planar partition. The algorithm is aware not to introduce any topological errors. Furthermore, we gave a theoretical description of the options that we have when employing this algorithm in practice. Another contribution is that we analysed how much the average number of points in the boundaries of the polygonal areas would grow without simplification to choose the best simplification strategy, also from the perspective of the amount of data to be stored in the data structures. Further we showed some visual results.

Some notes on future work:

- We think that an integrated way of formalizing 2D maps plus 1D for scale in a 3D space (leading to 3D volume objects, but where not all axes have the same geometric meaning) could lead to a better axiomatic description of what we consider to be a valid variable-scale hierarchy. This could also lead to an even more continuous variable-scale structure (opposed to our current solution, in which discrete 'jumps' still exist) in which it is possible to gradually morph polylines from the state before applying an aggregation or split operation to the state afterwards (for a technical implementation it might be sufficient to store only the beginning and end states in such a model). As such, it could enable smooth zooming of vector data for an end user.
- We plan to implement the requirements for valid planar partition and vario-scale hierarchy as check constraints in a DBMS (as technical implementation of the conceptual model).

References

Bader M. and Weibel R. (1997) Detecting and resolving size and proximity conflicts in the generalization of polygonal maps. pages 1525–1532.

Barkowsky T., Latecki L. J., and Richter K. F. (2000) Schematizing Maps: Simplification of Geographic Shape by Discrete Curve Evolution. In *Spatial Cognition II*, volume 1849 of *Lecture Notes in Computer Science*, pages 41–53. Springer Berlin / Heidelberg.

Bentley J. L. (1990) K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197. ACM, New York, NY, USA.

Da Silva A. C. G. and Wu S. T. (2006) A Robust Strategy for Handling Linear Features in Topologically Consistent Polyline Simplification. In AMV Mon-

teiro and CA Davis, editors, *GeoInfo*, VIII Brazilian Symposium on Geoinformatics, 19-22 November, Campos do Jordão, São Paulo, Brazil, pages 19–34.

De Berg M., Van Kreveld M., and Schirra S. (1998) Topologically Correct Subdivision Simplification Using the Bandwidth Criterion. Cartography and Geographic Information Science, 25:243–257.

Douglas D. H. and Peucker T. K. (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: The International Journal for Geographic Information and Geovisualization, 10(2):112–122.

Dyken C., Dæhlen M., and Sevaldrud T. (2009) Simultaneous curve simplification. Journal of Geographical Systems, 11(3):273–289.

Gröger G. and Plümer L. (1997) Provably correct and complete transaction rules for GIS. In *GIS '97: Proceedings of the 5th ACM international workshop on Advances in geographic information systems*, pages 40–43. ACM, New York, NY, USA.

Guibas L. J. and Sedgewick R. (1978) A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, 1978*, pages 8–21.

Kulik L., Duckham M., and Egenhofer M. (2005) Ontology-driven map generalization. Journal of Visual Languages & Computing, 16(3):245–267.

Ledoux H. and Meijers M. (2010) Validation of Planar Partitions Using Constrained Triangulations. In *Proceedings Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science*, pages 51–55. Hong Kong.

Meijers M., Van Oosterom P., and Quak W. (2009) A Storage and Transfer Efficient Data Structure for Variable Scale Vector Data. In *Advances in GIScience*, Lecture Notes in Geoinformation and Cartography, pages 345–367. Springer Berlin Heidelberg.

Ohori K. A. (2010) Validation and automatic repair of planar partitions using a constrained triangulation. Master's thesis, Delft University of Technology.

Plümer L. and Gröger G. (1997) Achieving integrity in geographic information systems—maps and nested maps. Geoinformatica, 1(4):345–367.

Ramer U. (1972) An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing, 1(3):244–256.

Saalfeld A. (1999) Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. Cartography and Geographic Information Science, 26:7–18.

Van Oosterom P. (1990) Reactive Data Structures for Geographic Information Systems. Ph.D. thesis, Leiden University.

Van Oosterom P. (2005) Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest. Cartography and Geographic Information Science, 32:331–346.

Visvalingam M. and Whyatt J. D. (1993) Line generalisation by repeated elimination of points. The Cartographic Journal, 30(1):46–51.