A Robust Strategy for Handling Linear Features in Topologically Consistent Polyline Simplification

Adler C. G. da Silva and Shin-Ting Wu

Department of Computer Engineering and Industrial Automation (DCA)
School of Electrical and Computer Engineering (FEEC)
State University of Campinas (UNICAMP)
P.O. Box 6101, 13083-970 – Campinas, SP, Brazil

{acardoso,ting}@dca.fee.unicamp.br

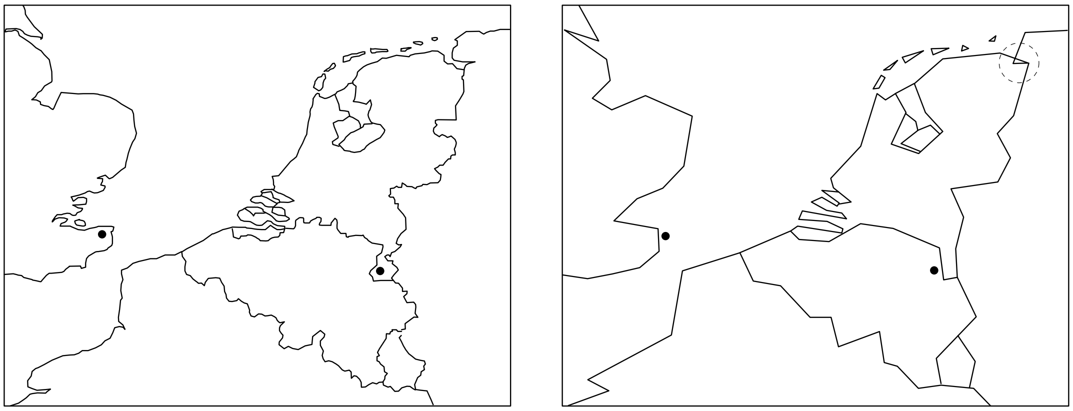
Abstract. *Polyline simplification is a technique that reduces the number of vertices of a polygonal chain for the purpose of map generalization and for speeding up processing and visualization in GIS. Unfortunately, the majority of simplification algorithms does not preserve the topological consistency of the map. To overcome this problem, some works proposed different approaches based on the consistency of a point feature for recovering the relationship of a polyline with respect to itself and to its neighbouring features. For the sake of simplicity, most algorithms unify the handling of linear and point features by considering a linear feature as a sequence of point features. This solution, however, fails in a few particular cases. In this paper, we firstly examine the reason for this solution to fail and then present a robust strategy for handling linear features.*

1. Introduction

Polyline simplification is one of most thoroughly studied subjects in map generalization. It consists in reducing the number of vertices of a polygonal chain for the purpose of removing unnecessary details. Besides its main application in generalization, it is also considerably employed in Geographic Information Systems (GIS) to reduce digital map data for speeding up processing and visualization. A variety of techniques has been presented by researchers in different contexts [Tobler 1964, Lang 1969, Reumann and Witkam 1974, Jenks 1981].

In the context of automated cartography, the most used algorithms are the Ramer-Douglas-Peucker (RDP) algorithm [Ramer 1972, Douglas and Peucker 1973], Visvalingam's algorithm [Visvalingam and Whyatt 1993] and Wang and Müller's algorithm [Wang and Müller 1998]. Unfortunately, like the majority of algorithms, none of them maintain the spatial relationship among features and, hence, cannot preserve the original topology of most maps (Figure 1). That is because they take the polyline in isolation, without regard to the features in its vicinity. Many attempts [Edwardes et al. 1998, McKeown et al. 1999] have been proposed to remove the topological conflicts in a post-processing stage, but, in the cases where the original data is not present, some inconsistencies cannot be avoided.

Other algorithms modify a polyline in context, taking into consideration the relationship between the polyline with nearby features. In these techniques, there is no need for a post-processing stage, since the topology is preserved along the simplification procedure. Many of them are based on an isolated simplification procedure and simply



(a) (b)

Figure 1. (a) Original map and (b) its inconsistent simplification outcome.

include a new topological criterion for selecting a vertex to be inserted in or removed from the polyline. In general, algorithms on simplification in context guarantees that: (i) the features remain on the same side with respect to the original and simplified polylines; (ii) the simplified polyline has no self-intersections; and (iii) the simplified polyline does not intersect any other feature.

Well known algorithms on simplification in context are presented in the papers of de Berg *et al.* [de Berg et al. 1998] and Saalfeld [Saalfeld 1999]. The former works on subdivision simplification, where the polylines are always part of some polygon in the map. It succeeds in generating a topologically consistent polyline that is at a maximum error ϵ from the original one and has as few vertices as possible. The latter works with a more general polyline simplification, involving linear features that may not be part of a polygon, as, for example, rivers and roads. It improves the classical RDP algorithm for recovering the topology of the original polyline. Saalfeld's algorithm is more popular than de Berg *et al.*'s, because of the popularity of the RDP algorithm, and because of its simpler implementation and faster processing.

For the sake of simplicity, these algorithms unify the handling of linear and point features by considering the vertices of a linear feature as point features. However, in some particular cases, even if the vertices of a line segment (handled as point features) lie on the correct side, the line segment can still intersect the the simplified polyline. The example of Figure 2 illustrates this situation. Before the simplification (Figure 2(a)), the points p_1 and p_2 lie outside the shaded region. After the simplification (Figure 2(b)), although the sidedness of the points is preserved, the line segment $\overline{p_1p_2}$ intersects the simplified polyline $\mathcal{P}'$.

Our motivation for this work is twofold. Firstly, we would like to remedy the inconsistent outcomes when handling linear features as point features. Secondly, we would like to devise an incremental sidedness test that is appropriate for handling linear features and can be easily integrated to Saalfeld's algorithm. The remainder of this paper is organized as follows. We present, in Section 2., a brief analysis of how consistency has been studied in previous works. Then, in Section 3., we explain how to overcome the sidedness problem with linear features. Afterwards, in Section 4., we give an algorithmic solution

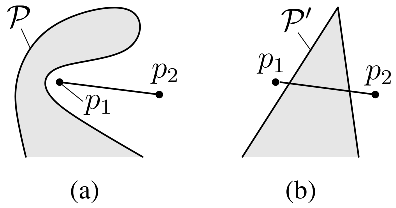


Figure 2. Case where consistency of point features fails in handling linear features: (a) the original polyline $\mathcal{P}$ and the line segment $\overline{p_1p_2}$, and (b) the simplified polyline $\mathcal{P}'$ that preserves the sidedness of p_1 and p_2, but still intersects $\overline{p_1p_2}$.

to Saalfeld's algorithm. After then, in Section 5., we give some basic results of our strategy and compare them to Saalfeld's. Finally, in Section 6., we present some concluding remarks and our future research directions.

2. Related Work

As previously stated, de Berg *et al.*'s work is on subdivision simplification. They assume that every polyline of a map is part of two polygons of the subdivision. For the purpose of validating their procedure, they formalized the definition of consistency of polylines with respect to point features as follows. Let $\mathcal{P}$ and $\mathcal{P}'$ be two simple polylines oriented from vertex v_1 to vertex v_n, and let F be a set of point features. The polylines $\mathcal{P}$ and $\mathcal{P}'$ are said to be consistent with respect to F, if there exists a simple polyline $\mathcal{C}$ oriented from vertex v_n to vertex v_1 that closes both $\mathcal{P}$ and $\mathcal{P}'$ to simple polygons which have the same subset of points of F in their interior as depicted in Figure 3. One can show that if there exists such a polyline $\mathcal{C}$, then any other simple polyline that closes both $\mathcal{P}$ and $\mathcal{P}'$ in simple polygons will give the same result for the consistency of $\mathcal{P}$ and $\mathcal{P}'$. In other words, the polylines $\mathcal{P}$ and $\mathcal{P}'$ are consistent with respect to F no matter what polygons of the subdivision they are part of.

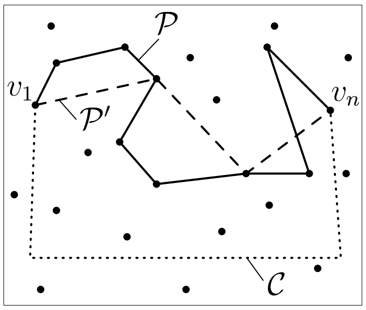
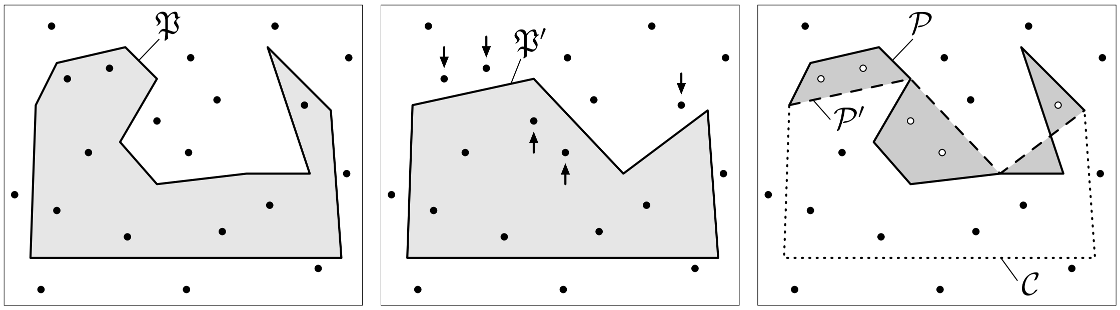


Figure 3. Consistency of polylines $\mathcal{P}$ and $\mathcal{P}'$ with respect to a set of point features.

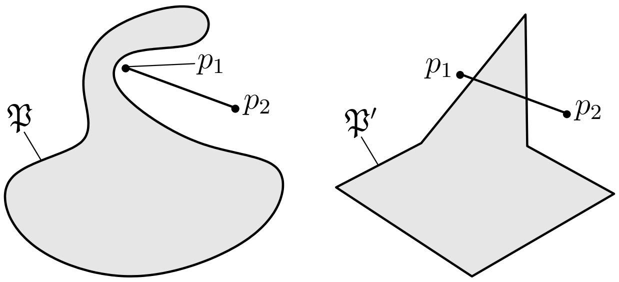
The reasoning of this definition is quite simple. Let us consider that, after the simplification of the configuration depicted in Figure 3, the polygon $\mathfrak{P}$, formed by $\mathcal{P}$ and $\mathcal{C}$ (Figure 4(a)), was replaced by the polygon $\mathfrak{P}'$, formed by $\mathcal{P}'$ and $\mathcal{C}$ (Figure 4(b)). A point feature is consistently placed with respect to $\mathfrak{P}$ and $\mathfrak{P}'$, if it lies inside or outside both polygons. In Figure 4(b), the point features lying inside $\mathfrak{P}$ and outside $\mathfrak{P}'$ are indicated with downward arrows and those lying outside $\mathfrak{P}$ and inside $\mathfrak{P}'$ are indicated with upward arrows. From Figure 4(c), we may see that the point features lying between $\mathcal{P}$ and $\mathcal{P}'$ are the only points that have different sidedness classification with respect to $\mathfrak{P}$ and $\mathfrak{P}'$.



(a) (b) (c)

Figure 4. (a, b) Consistency of the polygons $\mathfrak{P}$ and $\mathfrak{P}'$ with respect to a set of point features. (b) The only features that are on the wrong side lie between the polylines $\mathcal{P}$ and $\mathcal{P}'$.

The definition of consistency given by de Berg *et al.* is valid only for point features. Without any other constraints, the point feature consistency cannot be used for handling linear features. Figure 5 illustrates a particular case where intersections may still appear in the simplified map. Even with all points on the correct side with respect to $\mathfrak{P}$ and $\mathfrak{P}'$, intersections still occur. That is because, although the extremes of the segment are on the correct side, its intermediate points may still change their sidedness.



(a) (b)

Figure 5. Case of inconsistency with areal features: (a) original polygon of a subdivision and (b) inconsistent simplified polygon.

In his work, Saalfeld concentrates not only on features that lie in the wrong polygon, but also on features that lie on the wrong side of a linear feature. According to him, point features always change their sidedness, if they are trapped between $\mathcal{P}$ and $\mathcal{P}'$. Figure 6 gives an example of this situation. Among the point features of the figure, only the white ones lie between $\mathcal{P}$ and $\mathcal{P}'$. Three of them are below $\mathcal{P}$ and above $\mathcal{P}'$ and two of them are above $\mathcal{P}$ and below $\mathcal{P}'$. Actually, this is a generalization of the point feature consistency, defined by de Berg *et al.*, for polylines that are not part of polygons. As de Berg *et al.*, Saalfeld uses the point feature consistency for handling linear features.

Another important contribution of Saalfeld's work is the triangle inversion property, stated as follows. When two segments replace one segment in $\mathcal{P}'$ (or vice-versa), the only point features that invert their sidedness are those inside the triangle formed by the replaced segment and the two replacing segments. Figure 7 indicates the three points inside the triangle that inverted their sidedness in comparison to Figure 6. Saalfeld uses this

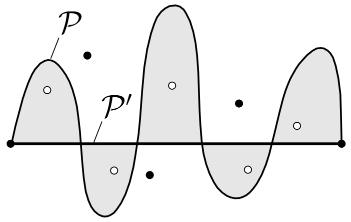


Figure 6. The sidedness of polylines for detecting inconsistent point features.

property in his algorithm to efficiently update the sidedness of features after the insertion of a vertex in the simplified polyline.

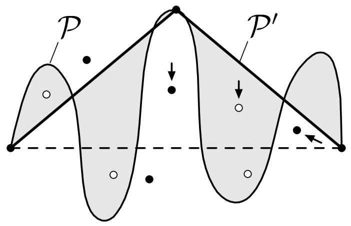


Figure 7. Triangle inversion property.

As we have shown, the point feature consistency fails for some cases of linear features. So far, most works on consistent simplification are not able to correctly handle linear features. Our motivation for this work is twofold. Firstly, we would like to remedy the inconsistent outcomes when handling linear features as point features. Secondly, we would like to devise an incremental sidedness test that is appropriate for handling linear features. This test would be equivalent to the triangle inversion test for point features.

3. Handling Linear Features

First of all, we have to identify the cases where the consistency for point features fails with linear features. Let us consider the configuration given in Figure 8(a). The line segments $\overline{v_i v_k}$ and $\overline{v_k v_j}$ replace respectively the subpolylines $\mathcal{P}_{ik}$ and $\mathcal{P}_{kj}$. Notice that p_1 is considered to be on the correct side, even if some intermediate points of the line segment $\overline{p_1 p_2}$ are not. That is because the subpolyline $\mathcal{P}_{kj}$ crosses the simplifying line segment $\overline{v_k v_j}$ of the subpolyline $\mathcal{P}_{ik}$ and forms the region depicted in Figure 8(b) where p_1 lies. One can show that problematic cases occur whenever a subpolyline $\mathcal{P}_{ab}$ crosses the simplifying segment $\overline{v_c v_d}$ of another subpolyline $\mathcal{P}_{cd}$.

The solution we adopted is very simple. We apply separately the sidedness criterion to each subpolyline and its correspondent simplifying segment. Figures 9(a) and 9(b) show the application of this criterion for the case of Figure 8. Notice that p_1 is on the wrong side with respect to both subpolylines, eliminating the problematic region. We formalize the consistency for linear features as follows. Let $\mathcal{P}$ be a polyline, $\mathcal{P}'$ be a simplified version of $\mathcal{P}$, and F be a set of vertices of linear features. The polylines $\mathcal{P}$ and $\mathcal{P}'$ are said to be consistent with respect to F, if the polygons formed by each subpolyline $\mathcal{P}_{ij}$ and its correspondent line segment $\overline{v_i v_j}$ contain no element of F. Figure 9(c) illustrates an example of a consistent simplification.

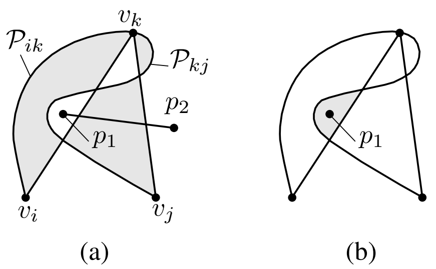


Figure 8. (a) Problematic case with (a) incorrect sidedness classification in the bounded region.

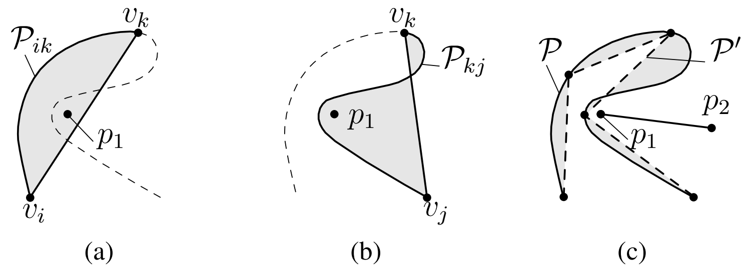


Figure 9. Our strategy to handle linear features.

We consider that the interior of the polygons (represented by the shaded regions in Figure 9) are determined with the parity (or odd-even) rule. We compute the number crossings between a ray from the feature and the polygon formed by a subpolyline and its correspondent simplifying line segment. If the number of crossings is odd, the feature is on the wrong side; otherwise, it is on the correct side (Figure 10(a) and 10(b)). For elucidating how the linear feature consistency works, we introduce the parity property as follows. Two points are considered to be on the same side of a polygon, if a line connecting them crosses the polygon an even number of times. Otherwise, they are considered to be on opposite sides. Figures 10(c) and 10(d) illustrate two examples of the parity property. Notice that the crossings on self-intersecting points of a polyline are counted separately.

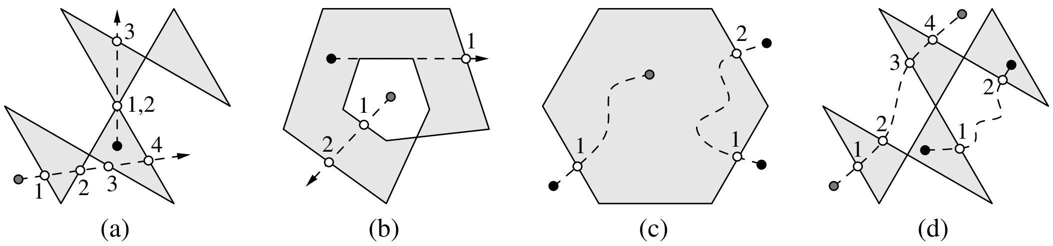


Figure 10. Cases of parity property .

Proposition 1. *Let $\mathcal{P}_{ij}$ be a subpolyline of the polyline $\mathcal{P}$ and let $\overline{v_i v_j}$ be its correspondent simplifying segment in $\mathcal{P}'$. If a line segment $\overline{p_1 p_2}$ does not intersect the original polyline $\mathcal{P}$, and the points p_1 and p_2 are both outside the polygon $\mathfrak{P}_{ij}$ formed by $\mathcal{P}_{ij}$ and $\overline{v_i v_j}$, then $\overline{p_1 p_2}$ does not intersect $\overline{v_i v_j}$.*

Proof. From the fact that p_1 and p_2 are both outside $\mathfrak{P}_{ij}$, we have that p_1 and p_2 do not coincide with the line segment $\overline{v_i v_j}$, and, consequently, $\overline{p_1 p_2}$ and $\overline{v_i v_j}$ do not overlap. Since $\overline{p_1 p_2}$ and $\overline{v_i v_j}$ do not overlap, they can intersect at most in one single point. From the parity property, we have that the line segment $\overline{p_1 p_2}$ crosses the polygon $\mathfrak{P}_{ij}$ an even number of times. Since $\overline{p_1 p_2}$ does not intersect $\mathcal{P}$ (and consequently $\mathcal{P}_{ij}$), and $\overline{v_i v_j}$ and $\overline{p_1 p_2}$ can only intersect in a single point, the number of crossings between $\overline{p_1 p_2}$ and $\mathfrak{P}$ must be zero. Hence, $\overline{p_1 p_2}$ does not intersect $\overline{v_i v_j}$. □

When applying the conditions of Proposition 1 to each subpolyline of $\mathcal{P}$ and its correspondent line segment in $\mathcal{P}'$, we ensure that $p_1 p_2$ will not intersect $\mathcal{P}'$. Hence, our approach guarantees that any linear feature that does not intersect the original polyline $\mathcal{P}$ will not intersect the segments of the simplified polylines $\mathcal{P}'$. It permits us to handle linear features just by taking into consideration its vertices. Since it is more restrictive than the point feature consistency, we can use it to handle both point and linear feature in a uniform manner, without making any distinction between them. In the next section we present an algorithmic solution that can correctly replace the triangle inversion test used in Saalfeld's algorithm.

4. Algorithm

In this section we present an algorithmic solution for correctly handling linear features. We substitute the triangle inversion one with our proposed strategy in Saalfeld's algorithm. To be self-contained, Saalfeld's algorithm is briefly presented in Section 4.1.. After then, in Section 4.2., our solution is introduced.

4.1. Saalfeld's Algorithm

Saalfeld [Saalfeld 1999] proposed some modifications to the RDP algorithm that give it the capability of recovering the topology of the original polyline. His strategy is to continue adding vertices to the "inconsistent" segments of the polyline until all errors are removed. His algorithm is convergent, because it always adds vertices. In the worst case, it adds all vertices of the original polyline and recovers the original geometry and, consequently, the original topology of the map. His algorithm is divided in two steps, as illustrates the flowchart of Figure 11.

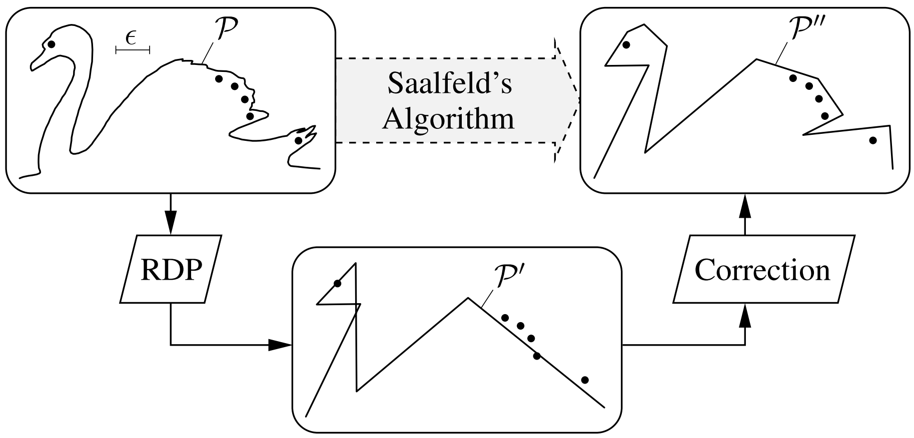


Figure 11. Flowchart of Saalfeld's algorithm.

The first step consists only in the application of the RDP in the input polyline $\mathcal{P}$ under a given tolerance ϵ. The correction step is depicted in Figure 12 and works as follows. For each subpolyline $\mathcal{P}_{ij}$ replaced by the polyline segment $\overline{v_i v_j}$ in the simplified polyline $\mathcal{P}'$, the algorithm determines its convex hull. Each subpolyline is then associated to the list of features that are inside its convex hull and have to be checked. This list may include point features, vertices of neighbouring polylines, and the remaining vertices of the polyline (i.e., the vertices v_k such that $k < i$ or $k > j$). The sidedness of these features are computed with the parity rule.

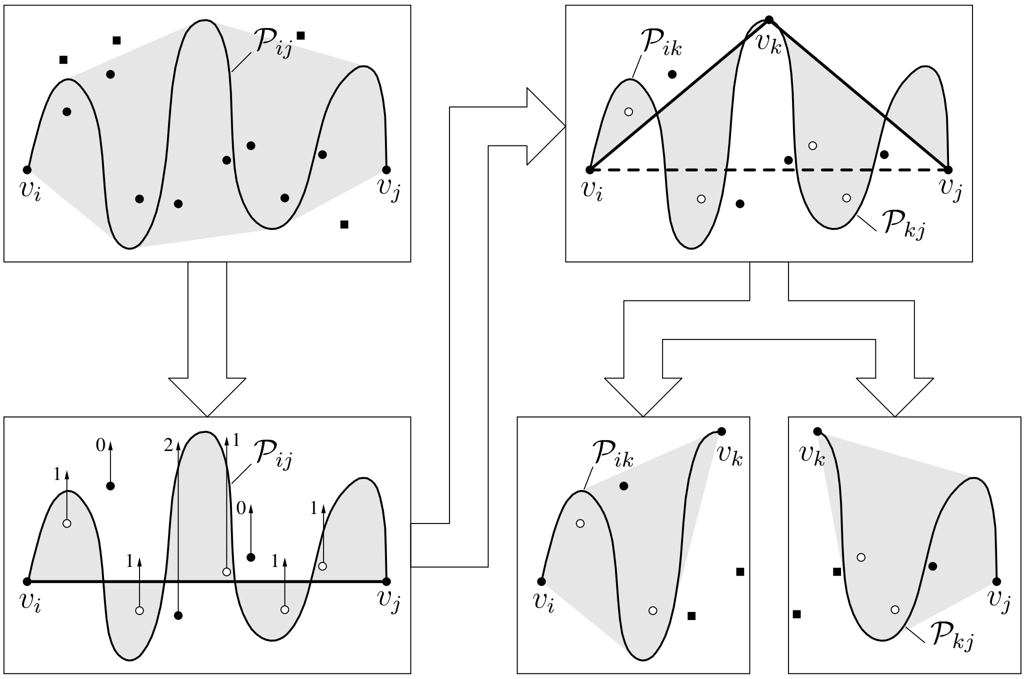


Figure 12. Correction step of Saalfeld's algorithm: selecting features inside the convex hull, and computing and updating their sidedness.

After the initialization, for each subpolyline $\mathcal{P}_{ij}$ that has features on the wrong side, the algorithm breaks its correspondent line segment $\overline{v_i v_j}$ by adding the farthest vertex v_k. It updates the sidedness of the external points in the current convex hull, using the triangle inversion property. Then, it splits the convex hull in two and selects the external points of the resulting convex hulls. After that, it calls the correcting procedure for the subpolylines $\mathcal{P}_{ik}$ and $\mathcal{P}_{kj}$. Since the whole process is restarted independently for the two subpolylines, the vertices of $\mathcal{P}_{ik}$ must be checked with respect to the convex hull of $\mathcal{P}_{kj}$ and vice-versa. If some vertices of one subpolyline interfere in the other subpolyline convex hull, their sidedness is computed with the parity rule.

To understand how intersections may occur, let us consider the application of Saalfeld's algorithm under tolerance ∞ to the polyline $\mathcal{P}$ of Figure 13(a). Because of the ∞-tolerance, the first step (RDP algorithm) adds no vertices to $\mathcal{P}'$. In the second step the algorithm first calculates the number of crossings of the points p_1, p_2, and p_3 and classifies their sidedness (Figure 13(b)). Since p_3 is on the wrong side, it adds the farthest vertex v_4 and updates the sidedness of p_3, that is inside the triangle $\triangle v_1 v_4 v_8$ (Figure 13(c)). Then, it handles independently the subpolylines $\mathcal{P}_{1,4}$ and $\mathcal{P}_{4,8}$, after evaluating the dependency of their vertices. In this evaluation, v_5 and v_6 are inserted in the list of features of $\mathcal{P}_{1,4}$ (Figure 13(d)). Both vertices are considered to be on the wrong side. The algorithm adds the vertex v_3 and updates the sidedness of p_1, v_6 and v_7 (Figure 13(e)). Observe that p_1 is on the wrong side, but the algorithms stops. That is because there is no more changes for $\mathcal{P}_{1,3}$ and $\mathcal{P}_{3,4}$ and the treatment for $\mathcal{P}_{4,8}$ finished a time ago (remember that it is a independent call to the correction procedure).

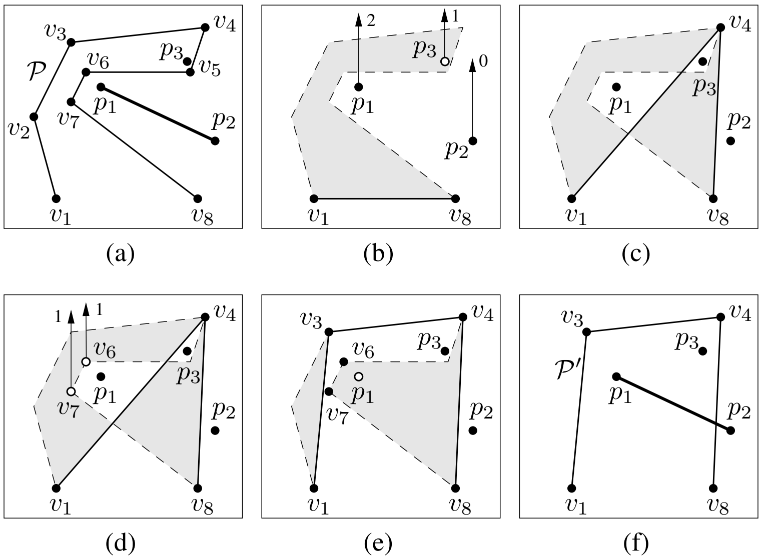


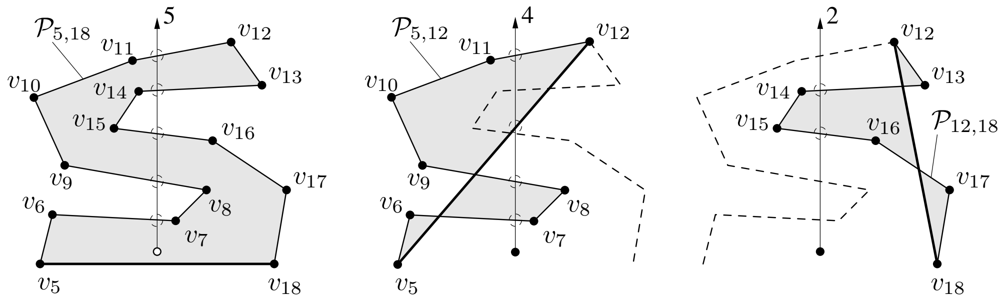
Figure 13. Step by step of Saalfeld's algorithm.

Observe, from Figure 13(c), that the triangle inversion property leads to the point consistency strategy of de Berg *et al.*. However, Saalfeld's algorithm adds more vertices to the simplified polyline, due to its independent treatment of different subpolylines. Nevertheless, his algorithms is not yet able to remove all intersections. In the next section, we present an algorithmic solution for overcoming the triangle inversion test and correctly handling the linear features.

4.2. Update of Sidedness for Linear Features

Our strategy is based on the fact that the relationship between a feature and the original polyline never changes throughout the course of the simplification. That permits us to associate to the feature a precomputed data to store this relationship. We divide our strategy in two stages. In the first stage, we compute the crossings between the upward ray from a feature f and a subpolyline $\mathcal{P}_{ij}$, as depicted in Figure 14(a), and fill the data structure associated to f. In the second stage, after breaking the line segment $\overline{v_i v_j}$ in two new line segments $\overline{v_i v_k}$ and $\overline{v_k v_j}$, we update the sidedness of f with respect to the subpolylines $\mathcal{P}_{ik}$ and $\mathcal{P}_{kj}$, as shown in Figures 14(b) and Figure 14(c), respectively. We present a pseudocode of our algorithm that can be easily integrated to Saalfeld's.

Besides its point coordinates `x` and `y`, we associate to the feature f the array `crossings` and the indices `begin` and `end`, as depicted in the structure of Figure 15(a). The array `crossings` is used to store the indices of the line segments of $\mathcal{P}_{ij}$ that cross the upward ray from f. Since in real maps the number of crossings is usually very small in comparison with the number of line segments being processed, one expects `crossings` to be very small too. The variables `begin` and `end` store initially



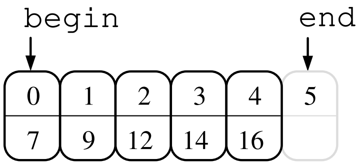
(a) (b) (c)

Figure 14. Stages to overcome the triangle inversion property: (a) computation of crossings between the upward ray from feature f and (b, c) update of its sidedness with respect to the resulting subpolylines.

the first and last indices of `crossings`. Figure 15(b) illustrates the initial state of the array `crossings` and the indices `begin` and `end`. (The element 5 of the array is illustrated just for the purpose of explanation.) Observe that the number of crossings can be directly obtained by the subtraction (`end` − `begin`).

```
Type Feature
Begin
    x, y: Float;
    crossings: Integer[];
    begin, end: Integer;
End
```

(a)



(b)

Figure 15. (a) Data structure used in our strategy and (b) its visual representation for the example of Figure 14(a).

The procedure computeCrossings outlined in Algorithm 1 performs the computation of the crossings between the ray and the original subpolyline, stores them in the array and initialize the indices. From line `08`, a crossing is found when (1) the subpolyline changes its side with respect to the ray (`currSide` and `prevSide` are different) and (2) f is under the current line segment. Observe, in line `09`, that the algorithm firstly stores the indices in a linked list. That is because, before processing, it does not know the number of crossings. After the computation, the content of the list is finally copied to the array, for which enough memory has been allocated (line `13`). Since the search for crossings is done from `i+1` to `j` (line `06`), the indices of the crossing line segments are stored in ascending order. The ordered array has a specific purpose in the update stage.

In the update stage, to determine the number of crossings of the upward ray from f with the subpolylines $\mathcal{P}_{ik}$ and $\mathcal{P}_{kj}$, the algorithm adopts the following strategy. Since the algorithm already computed the crossings with $\mathcal{P}_{ij}$ and stored them, it just looks for the first element after the index k (of the breaking vertex v_k) in the array `crossings`. Let us reconsider the example of Figure 14, where $k = 12$. The algorithm allocates two distinct copies of f for $\mathcal{P}_{ik}$ and $\mathcal{P}_{kj}$. Then, it looks for the first element in `crossings` greater than k, the element 14 of index 3. After then, it updates the index `begin` and `end` of

```
Procedure computeCrossings(P: Polyline; i, j: Integer; var p: Feature)
Var
   k: Integer;
   list: IntegerList;
   prevSide, currSide: Side;
Begin
   prevSide ← (P[i].x ≤ p.x) ? left : right;
   For k ← i+1 to j do
      currSide ← (P[k].x ≤ p.x) ? left : right;
      If currSide ≠ prevSide .and. p is under line segment P[k]P[k-1] then
         Push k in list;
      End if
      prevSide ← currSide;
   End for
   Allocate memory for p.crossings and copy the content of list to it;
   p.begin ← 0
   p.end ← list.size
End
```

Algorithm 1. Compute the crossings between the upward ray from feature `f` and the line segments of subpolyline P_{ij}, and store the indices of the intersecting segments in the array `crossings` associated to `f`.

the copies of f as depicted in Figure 16. The ascending order of the array `crossings` permits the algorithm to perform a binary search. To avoid the overhead of copying `crossings`, the copies of f just keep a reference to it. After the update process, we can obtain the number of crossings for each subpolyline just by subtracting the new indices.

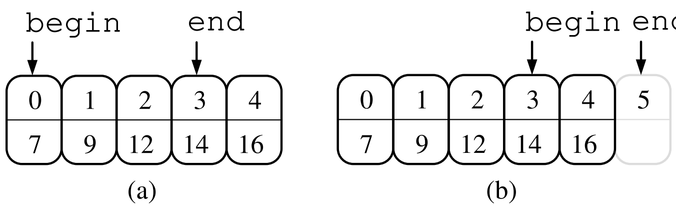


Figure 16. Results of the update stage of the first strategy for the subpolylines of (a) Figure 14(b) and (b) Figure 14(c).

For computing the sidedness of a given feature f, the algorithm has time complexity $O(n)$, due to the search for crossings, and memory complexity $O(n)$, due to the array `crossings`. For updating the sidedness of f, the binary search gives time complexity $O(\log n)$. The new copies of f keep just a reference to `crossings`, so there is no overhead for copying the array. The processing time of this algorithm is comparable to the the triangle inversion approach, because the array of crossings is usually very small. In Section 5., we present some results that confirm this hypothesis.

5. Results

To validate our theoretical study of consistency for linear features, we present some results of our approach and compare them to those of the point feature consistency used with Saalfeld's algorithm. We examine the basic cases where Saalfeld's strategy fails in preserving the original polyline topology. For each image, the first square presents the original polyline, the second square shows the outline of Saalfeld's algorithm, and the third square exhibits the outline of our approach. The results represent typical cases of intersections (Figures 17(a) and 17(b)), self-intersections (Figures 17(c) and 17(d)), and misplaced point features (Figures 17(e) and 17(f)) that occurs in Saalfeld's algorithm with point feature consistency, but are correctly handled with the linear feature consistency.

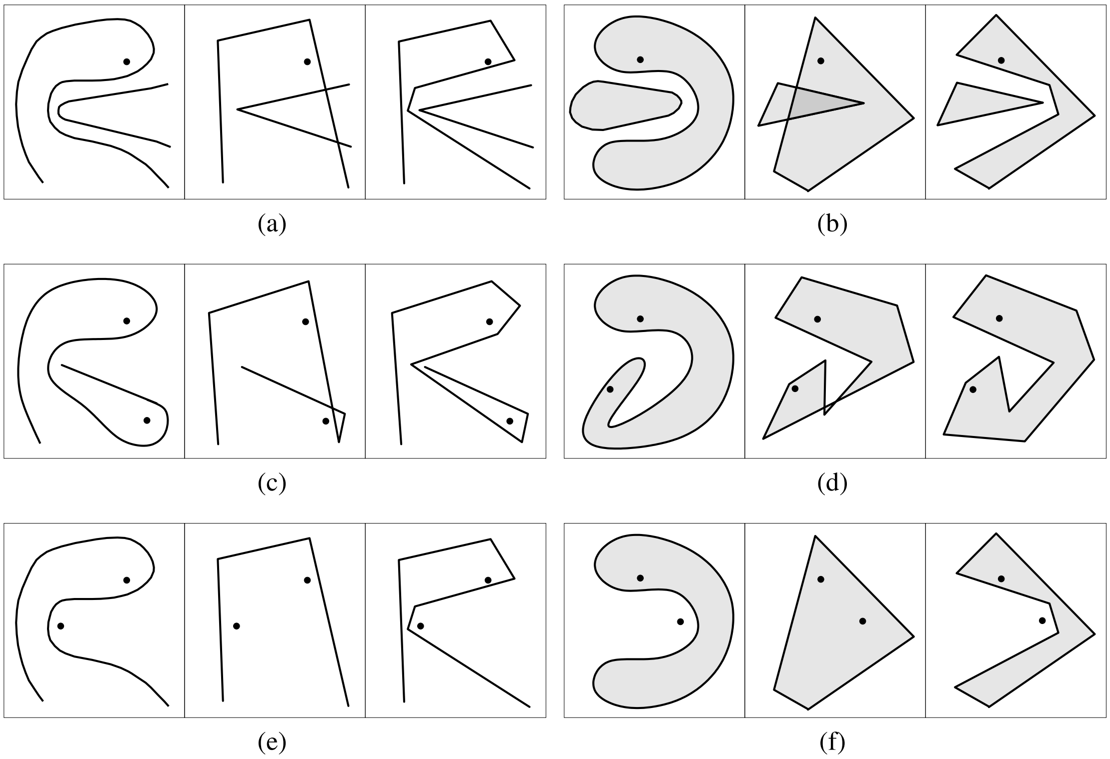


Figure 17. Comparison between the point feature consistency and linear feature consistency in Saalfeld's algorithm.

To validate our algorithmic solution, we compare it with the triangle inversion test in terms of time performance. Table 1 gives the processing time of both strategies with six different data sets. Notice that the processing time of the array of crossings is very close to the one of the inversion triangle test, even for the biggest data set. The array of crossings inserts in some cases more vertices than the triangle test. The performance of both algorithms can be considered equivalent.

6. Concluding Remarks

In this paper, we firstly studied the common problem of using the point feature consistency for handling linear features in topologically consistent polyline simplification algorithms [de Berg et al. 1998, Saalfeld 1999]. We observed that, in the particular case of Saalfeld's algorithm [Saalfeld 1999], a few arrangements of linear features can still lead to intersections. To overcome this problem, we presented a more restrictive consistency constraint for avoiding both changes of sidedness of point and linear features and intersections between linear features. We consider that the sidedness of a point or a vertex of

Original	Triangle inversion		Array of crossings	
Points	Time	Points	Time	Points
26,536	0.930s	7,288	0.935s	7,288
52,639	1.310s	12,165	1.320s	12,167
68,506	2.312s	15,683	2.330s	15,685
103,450	3.010s	29,713	3.110s	29,713
126,404	3.080s	25,769	3.140s	25,775
166,157	5.650s	24,954	5.750s	24,987

Table 1. Comparison of time performance between the triangle inversion test and the strategy with array of crossings.

a line segment must be individually checked against each subpolyline $\mathcal{P}_{ij}$ of the original polyline and its correspondent simplifying line segment $\overline{v_i v_j}$ in the output polyline. This simple theoretical solution permits us to uniformly handle both point and linear features.

In the practical context, the main contribution of this paper lies on an algorithmic strategy that can finely replace the triangle inversion test employed in Saalfeld's algorithm. Our strategy is based on the fact that, once the crossings are computed, they can be stored in a data structure and recovered whenever one need it. We discussed the ins and outs of the presented strategy and showed that its time complexity is comparable to the triangle inversion test. We also gave a pseudocode of the procedure that may be directly integrated to Saalfeld's algorithm. Finally, we presented some results of both procedures and compared them to the ones of Saalfeld's original algorithm.

Our future researches point mainly in two directions. The first direction is the development of a new strategy that handles differently point and linear features. This would have the advantage of being less restrictive for point features, which could be handled with the triangle inversion test. Besides, we want to remedy only the intersections that are really happening. For that, we have consider the line segments explicitly, instead of handling just their end-points. The second direction is the development of a new topologically consistent iterative simplification that would treat all the polylines of a map in a global approach. This strategy has the advantage of testing only the current vertices on the simplified polylines, resulting in better simplifications and faster processing.

References

de Berg, M., van Kreveld, M., and Schirra, S. (1998). Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required for represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122.

Edwardes, A., Mackaness, W., and Urvin, T. (1998). Self evaluating generalization algorithms to automatically derive multi scale boundary sets. In *The 8th International Symposium on Spatial Data Handling*, pages 361–372, Vancouver, Canada.

Imai, H. and Iri, M. (1988). *Computational morphology*, chapter Polygonal approximations of a curve-formulations and algorithms, pages 71–86. Elsevier.

Jenks, G. F. (1981). Lines, computers and human frailties. In *Annals of the Association of American Geographers*, volume 71, pages 1–10.

Lang, T. (1969). Rules for the robot draughtsmen. *The Geographical Magazine*, 42(1):50–51.

McKeown, D., McMahill, J., and Caldwell, D. (1999). The use of spatial context in linear feature simplification. In *GeoComputation 99*, Mary Washington College, Fredericksburg, Virginia.

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:224–256.

Reumann, K. and Witkam, A. P. M. (1974). Optimizing curve segmentation in computer graphics. In Gunther, A., Levrat, B., and Lipps, H., editors, *Proceedings of the International Computing Symposium*, pages 467–472. American Elsevier.

Saalfeld, A. (1999). Topologically consistent line simplification with the Douglas-Peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Tobler, W. R. (1964). An experiment in the computer generalization of map. Technical report, Office of Naval Research, Geography Branch.

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

Wang, Z. and Müller, J. C. (1998). Line generalization based on analysis of shape characteristics. *Cartography and Geographic Information Systems*, 22(4):264–275.

Zhang, Y.-Y. (1996). A modified Douglas-Peucker simplification algorithm. Master's thesis, The Ohio State University.