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, namely the spatial placement of a polyline with respect to itself and to its neighbouring features. To overcome this problem, some approaches based on the consistency of a point feature have been proposed. For the sake of simplicity, they 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 it to fail and then present a robust strategy for remedying the remaining problems without abandoning the basic principle of reducing a linear feature to a sequence of point 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 in order to represent them at a smaller scale without 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 and to homogenize different data sets in the process of data integration. A variety of techniques has been presented by researchers in different contexts [Tobler 1964, Lang 1969, Reumann and Witkam 1974, Jenks 1981].

In automated cartography, the most used algorithms are the classical 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 maintains the spatial relationship among features and, hence, cannot preserve the original topology of most maps (Figure 1). This is because they take the polyline in isolation, without considering the features in its vicinity. Many ideas [Müller 1990, Edwardes et al. 1998, McKeown et al. 1999] have been published in attempt 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.

There exists a second class of algorithms which takes into consideration the whole map throughout the course of the simplification [van der Poorten and Jones 1999,

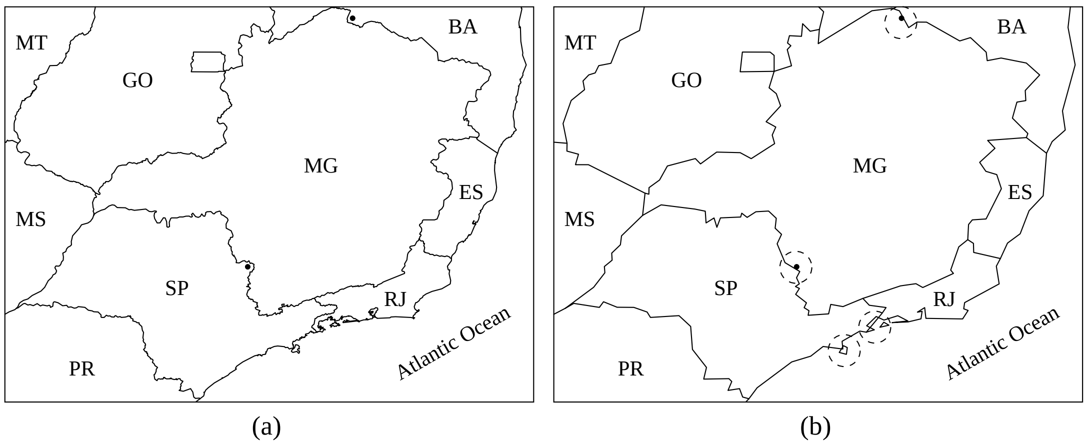


Figure 1. (a) Original map and (b) its inconsistent simplification outcome.

Ai et al. 2000, van der Poorten and Jones 2002]. In these techniques, a constrained Delaunay triangulation is performed on the whole collection of map features in a pre-processing stage. The triangulation-based approach permits these algorithms to implicitly preserve the topology of the map while removing vertices from the polylines. Due to its great capability to store spatial relationships and to detect topological conflicts, the Delaunay triangulation is also used to implement other generalization operators, such as exaggeration, collapse and amalgamation [Jones et al. 1995]. However, when the concern is only on the simplification, this approach may be expensive, since the vicinity of any feature or subfeature must be retriangulated whenever it is removed from the map.

A third class of algorithms modifies a polyline in context, taking into consideration only the relationship between the polyline with nearby features, instead of the complete map features. In these techniques, there is no need for a pre-processing or post-processing stage. The topology of the polyline is conserved along the simplification procedure by preserving the sidedness of the features that are inside its convex hull. Many of these techniques are based on an isolated simplification procedure and simply include the sidedness topological constraint when selecting a vertex to be inserted in or removed from the polyline. This simplification approach may be an alternative to the triangulation-based ones for efficiently solving topological conflicts.

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 two polygons 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 on 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, the algorithms on simplification in context 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 $\mathcal{P}'$.

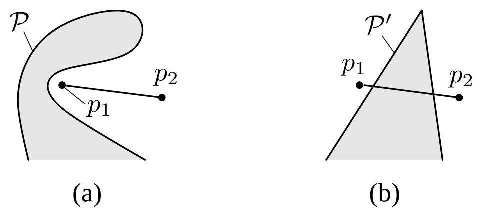


Figure 2. A case where point feature consistency fails in handling linear features: (a) the original polyline $\mathcal{P}$ and the line segment $\overline{p_1p_2}$, and (b) the simplified polyline $\mathcal{P}'$ that preserves the sidedness of p_1 and p_2, but still intersects $\overline{p_1p_2}$.

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 the 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 to Saalfeld's algorithm. After then, in Section 5., we give some basic results of our strategy and compare them to Saalfeld's solution. 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.

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

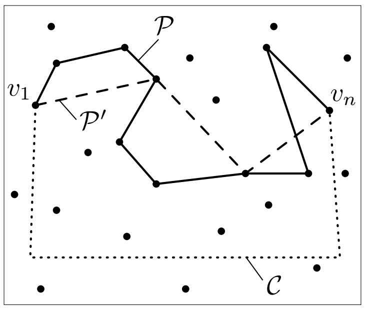


Figure 3. Consistency of polylines $\mathcal{P}$ and $\mathcal{P}'$ with respect to a set of point features.

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 $\mathfrak{P}'$.

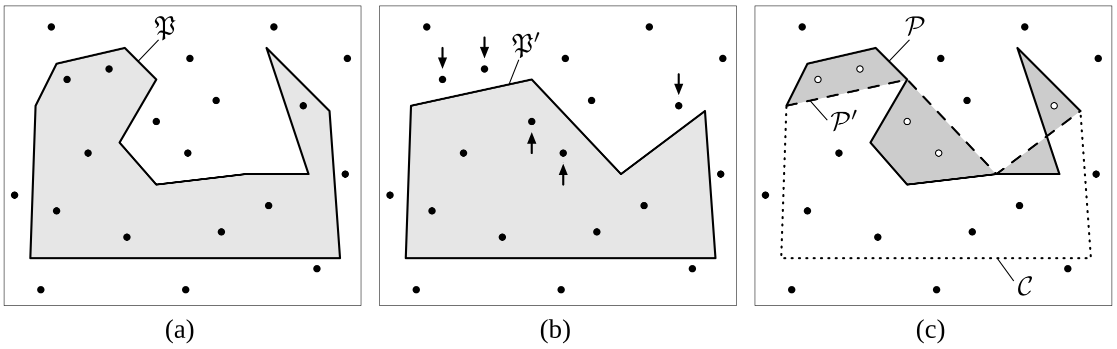


Figure 4. (a, b) Consistency of the polygons $\mathfrak{P}$ and $\mathfrak{P}'$ with respect to a set of point features. (c) The only features that are on the wrong side lie between the polylines $\mathcal{P}$ and $\mathcal{P}'$.

The definition of consistency given by de Berg *et al.* is valid only for point features. Without additional constraints, the point feature consistency cannot be used for handling linear features. Applying this definition on the cases illustrated in Figure 2, we may close the polyline $\mathcal{P}$ and $\mathcal{P}'$ and build the polygons $\mathfrak{P}$ and $\mathfrak{P}'$, respectively, as shown in Figure 5. Even with all points on the correct side with respect to $\mathfrak{P}$ and $\mathfrak{P}'$, intersections still occur. This is because although the extremes of the segment are on the correct side, its intermediate points lie on the wrong side.

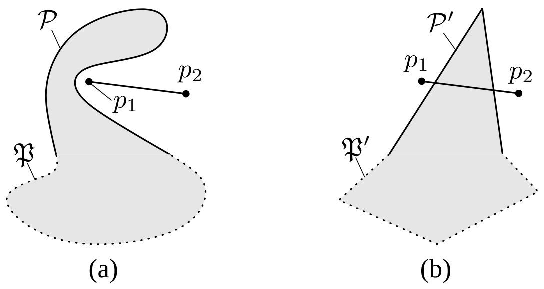


Figure 5. Case of inconsistency with areal features: (a) original polygon of a subdivision and (b) inconsistent simplified polygon.

In his work, Saalfeld concentrates not only on features that are 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.

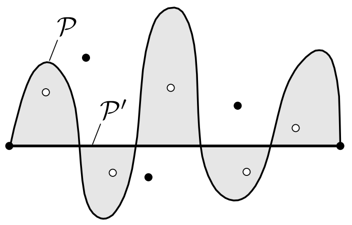


Figure 6. The sidedness of polylines for detecting inconsistent point features.

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

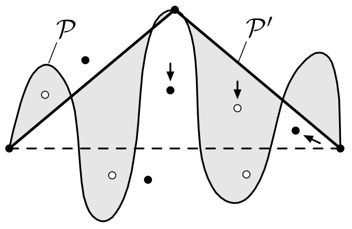


Figure 7. Triangle inversion property.

So far, as we know, works on simplification in context, that are based on reducing linear to point features, are not able to correctly handle linear 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 $\overline{v_i v_k}$ of the subpolyline $\mathcal{P}_{ik}$ and forms the region depicted in Figure 8(b) where p_1 lies. One can show that inconsistencies may 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

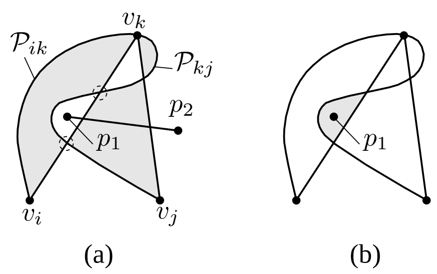


Figure 8. (a) Case of inconsistency with (b) the incorrect sidedness classification in the bounded region.

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 $\overline{v_iv_j}$ contain no element of F. Figure 9(c) illustrates an example of a consistent simplification.

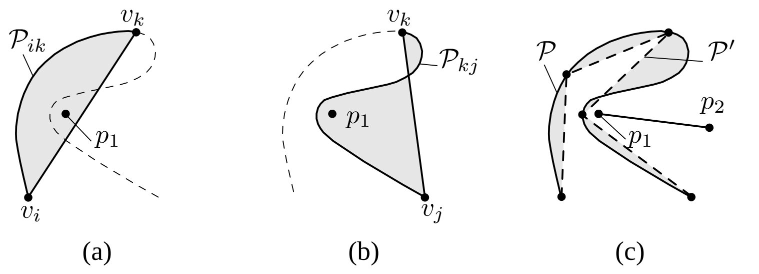


Figure 9. Our strategy to handle linear features.

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 as many as the number of segments intersect on it.

We proceed to formalize a sufficient point feature-based condition for ensuring consistent linear simplification.

Proposition 1. *Let $\mathcal{P}_{ij}$ be a subpolyline of the polyline $\mathcal{P}$ and let $\overline{v_iv_j}$ be its correspondent simplifying segment in $\mathcal{P}'$. If a line segment $\overline{p_1p_2}$ 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 $\overline{v_iv_j}$, then $\overline{p_1p_2}$ does not intersect $\overline{v_iv_j}$.*

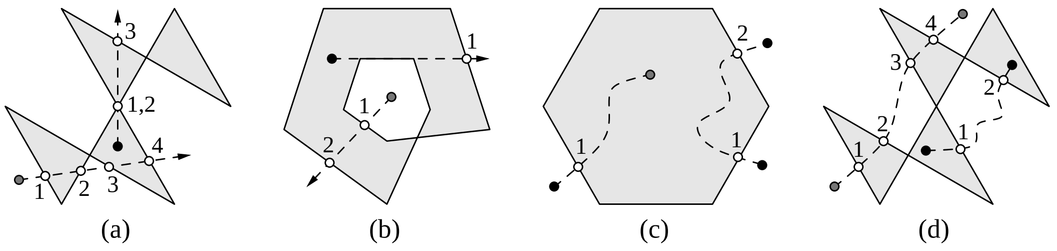


Figure 10. (a, b) Computing sidedness with the parity rule and (c, d) the application of the parity property.

Proof. From the fact that $\overline{p_1p_2}$ does not intersect $\mathcal{P}$ (and consequently $\mathcal{P}_{ij}$) and 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 $\overline{v_iv_j}$, and, consequently, $\overline{p_1p_2}$ and $\overline{v_iv_j}$ do not overlap. Therefore, they can intersect at most in one single point. From the parity property and from the fact that p_1 and p_2 are on the same side of the polygon $\mathfrak{P}_{ij}$, we have that the line segment $\overline{p_1p_2}$ crosses $\mathfrak{P}_{ij}$ an even number of times. Since $\overline{p_1p_2}$ does not intersect $\mathcal{P}_{ij}$, if there is any crossings, it must be between the line segments $\overline{v_iv_j}$ and $\overline{p_1p_2}$. However, since they can intersect in no more than a single point, the number of crossings between $\overline{p_1p_2}$ and $\mathfrak{P}_{ij}$ to be even must be zero. Hence, $\overline{p_1p_2}$ does not intersect $\overline{v_iv_j}$. □

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_1p_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 $\mathcal{P}'$. Since it is more restrictive than the point feature consistency, we can use it to uniformly handle both point and linear feature, without making any distinction between them. In the next section we present an algorithmic solution that can correctly replace the triangle inversion test devised in Saalfeld's algorithm.

4. The Algorithm

In this section we present an algorithmic solution for correctly handling linear features. We replace the triangle inversion one by 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 described.

4.1. Saalfeld's Algorithm

Saalfeld [Saalfeld 1999] proposes some modifications to the RDP algorithm that give it the capability of recovering the topology of the original polyline. His strategy is to successively add vertices to the "inconsistent" segments of the polyline until all errors are removed. His algorithm is convergent, because, in the worst case, it adds all vertices of the original polyline, recovers the original geometry and, consequently, the original topology of the map. Naturally, in real data sets, the worst case almost never occur. His algorithm is divided in two steps, as illustrates the flowchart of Figure 11. The first step consists only in the application of the RDP in the input polyline $\mathcal{P}$ while a given tolerance ϵ is not achieved and the second step is comprised of the topological correction procedures.

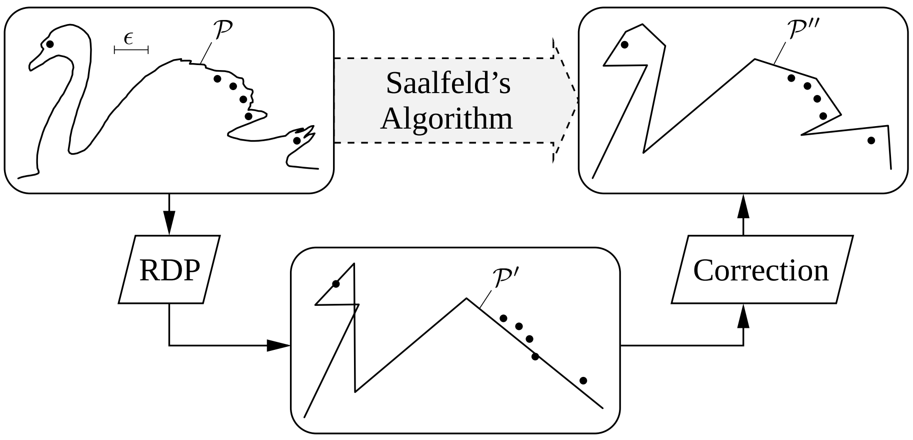


Figure 11. Flowchart of Saalfeld's algorithm.

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. These features represent potential topological conflicts. This list may include point features, vertices of neighbouring polylines, and the remaining vertices of the polyline itself (namely the vertices v_k such that $k < i$ or $k > j$). The sidedness of these features are computed with the parity rule.

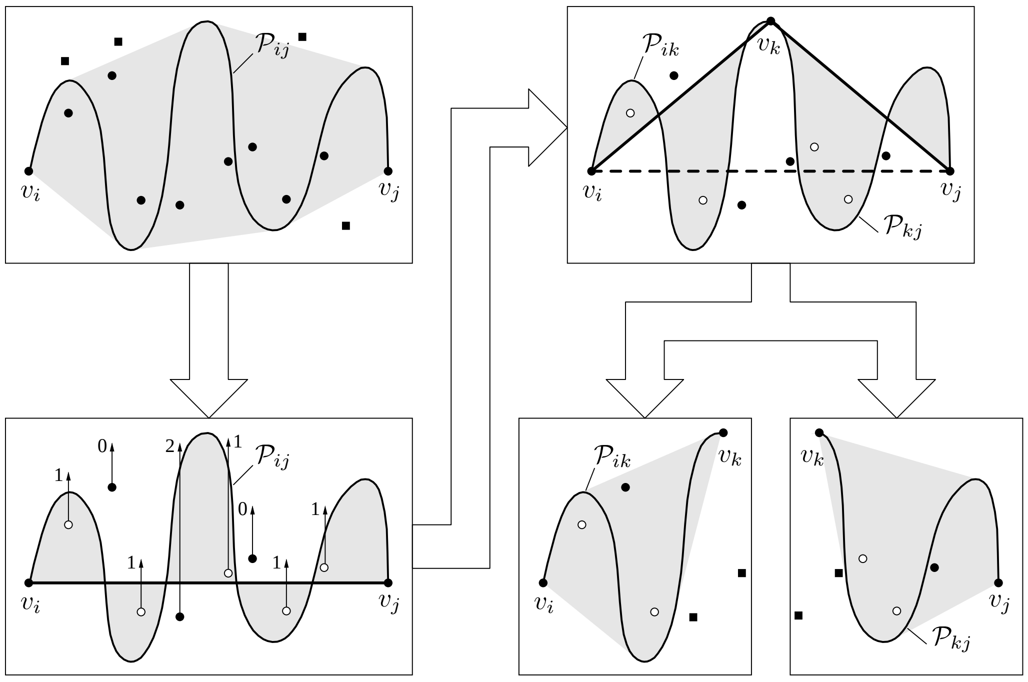


Figure 12. Correction step of Saalfeld's algorithm: selecting features inside the convex hull, and computing and updating their sidedness classification.

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 classification 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 classification 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. As all the features are on the correct side with respect to $\mathcal{P}_{4,8}$, no further splitting should be applied on it. Regarding to $\mathcal{P}_{1,4}$, the vertices v_5 and v_6 are inserted in its list of features (Figure 13(d)). Both vertices are considered to be on the wrong side. The algorithm adds the vertex v_3 and updates the sidedness classification 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 $\mathcal{P}_{3,4}$.

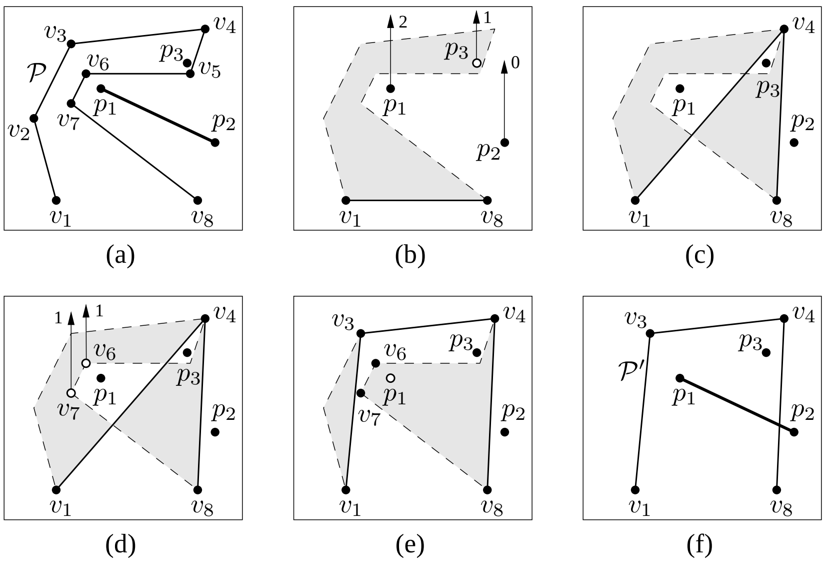


Figure 13. Saalfeld's algorithm.

Observe, from Figure 13(c), that the triangle inversion property is equivalent 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 distinct subpolylines. Nevertheless, his algorithms is not yet able to remove all intersections.

4.2. Update of Sidedness Classification

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. This permits us to associate the feature to precomputed data that 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 them in a 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 classification 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.

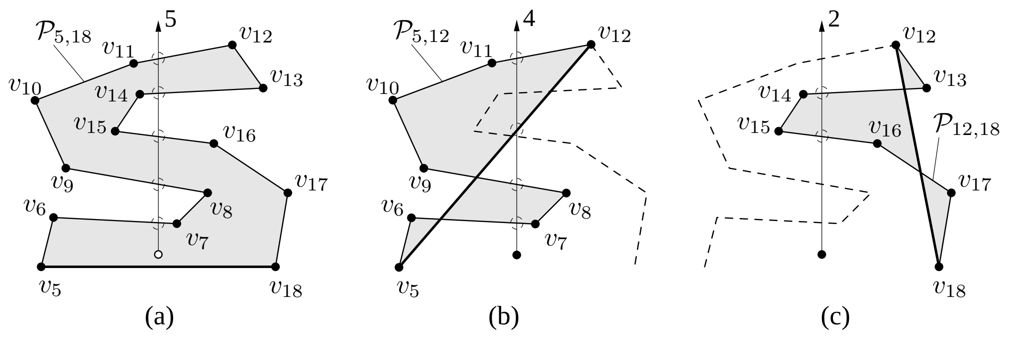
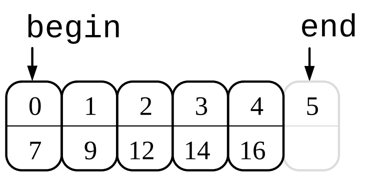


Figure 14. Stages to overcome the triangle inversion property: (a) computation of crossings between the upward ray from feature f and (b, c) update of its sidedness classification with respect to the resulting subpolylines.

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 to the number of line segments being processed, one expects the array `crossings` to be very small too. The variables `begin` and `end` store initially 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) The data structure used in our strategy and (b) its visual representation for the polyline 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 initializes 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 below the current line segment. Observe, in line `09`, that the algorithm firstly stores the indices in a linked list. This 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 crossed line segments are stored in ascending order. The ordered array has a specific purpose in the update stage.

```
Procedure computeCrossings(P: Polyline; i, j: Integer; var p: Feature)
01 Var
02    k: Integer;
02    list: IntegerList;
03    prevSide, currSide: Side;
04 Begin
05    prevSide ← (P[i].x ≤ p.x) ? left : right;
06    For k ← i+1 to j do
07       currSide ← (P[k].x ≤ p.x) ? left : right;
08       If currSide ≠ prevSide .and. p is below line segment $\overline{P[k]P[k-1]}$ then
09          Push k in list;
10       End if
11       prevSide ← currSide;
12    End for
13    Allocate memory for p.crossings and copy the content of list to it;
14    p.begin ← 0
15    p.end ← list.size
16 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`.

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 has 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, and finds the element 14 of index 3. After then, it updates the index `begin` and `end` of 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.

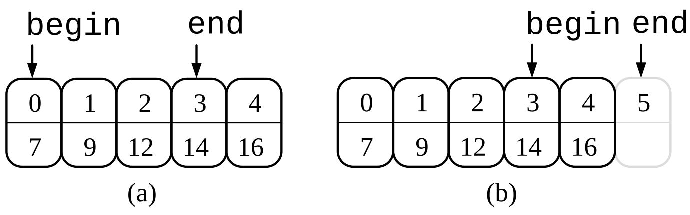


Figure 16. Results of the update stage of the first strategy for the subpolylines of (a) Figure 14(b) and (b) Figure 14(c).

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 classification 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 time complexity of the triangle inversion test, because the array of crossings is usually very small. In Section 5., we present some results that validate this statement.

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.

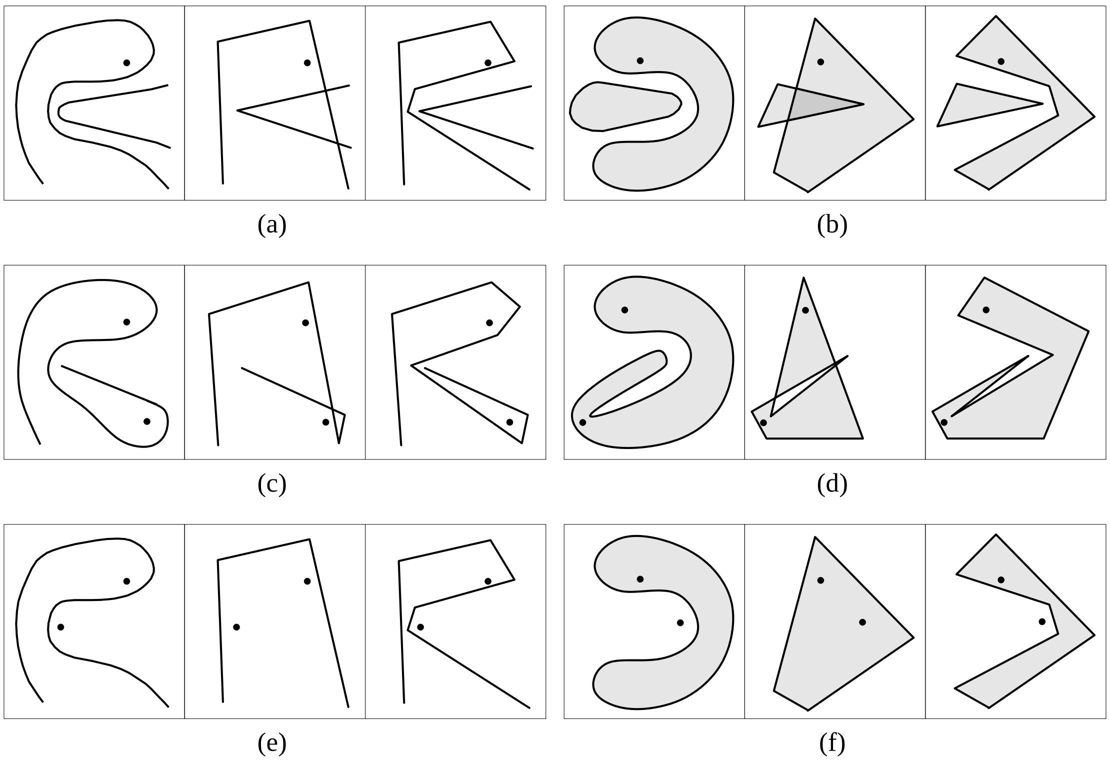


Figure 17. Comparison between the point feature consistency and linear feature consistency in Saalfeld's algorithm.

To validate our algorithmic solution for ensuring the topological correctness of both point and linear features, we compare it with the triangle inversion test in terms of time performance. Table 1 gives the processing time of both strategies for maps with different number of points. Notice that the processing time of the array of crossings is very close to the one of the triangle inversion test, even for the map with more vertices. This is because the number of crossings and, consequently, the size of the array are very small when compared to the number of vertices of the polylines. Thus, the search for crossings in the array can be considered of constant time complexity and the performance of the algorithms can be considered equivalent. Another important result that we achieved with our proposal is that the additional number of vertices that is required to preserve the topological consistency is insignificant. In all cases that we tested, it is less than one vertex in 4,000 inserted by Saalfeld's algorithm (approximately 0.025%).

Original map	Simplified map			
	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.

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. We observed that, if no pre-processing is carried out in order to satisfy some conditions, 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 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 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 needs. 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 our procedure and compared it to the ones of triangle inversion test, showing that the procedures have equivalent performances, but our technique always preserves the topology of the original map.

Our future researches point mainly to the development of a topologically consistent simplification procedure that would treat all the polylines together in a global approach. This strategy has the advantage of testing only the current vertices on the simplified polylines, instead of checking all the vertices of the nearby polylines, resulting in better generalizations and faster processing. We intend to separate the simplification procedure from the topological control, so that it may be possible to ensure topological consistency in distinct isolated simplification algorithms. We also plan to place this new algorithm as a faster alternative to the triangulation-based approach for preserving topological consistency in simplification.

Acknowledgments

We would like to acknowledge the Coordination for the Improvement of Higher Education Personnel Foundation (CAPES) and the State of São Paulo Research Foundation (FAPESP, Grant N^o 2003/13090-6) for financial support.

References

Ai, T., Guo, R., and Liu, Y. (2000). Safe sets for line simplification. In *The 9th International Symposium on Spatial Data Handling*, pages 30–43.

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.

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

Jones, C.-B., Bundy, G.-L., and Ware, J.-M. (1995). Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22(4):317–331.

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.

Müller, J. C. (1990). The removal of spatial conflicts in line generalisation. *Cartography and Geographic Information Systems*, 17(2):141–149.

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.

van der Poorten, P. and Jones, C. (1999). Customisable line generalisation using Delaunay triangulation. In *The 19th International Cartographic Association Conference*.

van der Poorten, P. and Jones, C. (2002). Characterisation and generalisation of cartographic lines using Delaunay triangulation. *International Journal of Geographical Information Science*, 16(8):773–795.

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.