The Douglas-Peucker Algorithm: Sufficiency Conditions for Non-Self-Intersections[1]

Wu, Shin - Ting   **da Silva, Adler C. G.**   **Márquez, Mercedes R. G.**

Universidade Estadual de Campinas (UNICAMP)
Faculdade de Engenharia Elétrica e Computação (FEEC)
Departamento de Engenharia de Computação e Automação Industrial (DCA)
{ting,acardoso,meche}@dca.fee.unicamp.br

30th October 2004

Abstract

The classic Douglas-Peucker line-simplification algorithm is recognized as the one that delivers the best perceptual representations of the original lines. It may, however, produce simplified polyline that is not topologically equivalent to the original one consisting of all vertex samples. On the basis of properties of the polyline hulls, Saalfeld devised a simple rule for detecting topological inconsistencies and proposed to solve them by carrying additional refinements. In this paper, we present an alternative form for the classic Douglas-Peucker to produce a simplified polyline which is homeomorphic to the original one. Our modified Douglas-Peucker algorithm is based on two propositions: (1) when an original polyline is star-shaped, its simplification from the Douglas-Peucker procedure cannot self-intersect; and (2) for any polyline, two of its star-shaped sub-polylines may only intersect if there is a vertex of one simplified sub-polyline inside the other's corresponding region.

Keywords: Topological Consistency; Line Simplification; Douglas-Peucker Algorithm; GIS.

1   Introduction

Often the geometric resolution of a polyline is much higher than the resolution supported by the application, such as visualization of geographic map boundaries or visualization of curves approximated by sampling a parametric curve at regular small intervals in a raster display. For the sake of efficiency, we search for algorithms that can extract essential features from detailed data of the original polyline and represent them on a simple one having fewer vertices, sufficient for the specified resolution. Many of them have been pursued by the researchers in different contexts [5, 6, 8, 11, 12, 14].

A simple line-simplification algorithm consists in constructing a polyline with line segments larger than an accepted tolerance. It may be achieved by discarding recursively the subsequent vertices whose distance from a prior initial vertex is less than a specified maximum distance. The

---

[1] Based on "A non-self-intersection Douglas-Peucker Algorithm", by Wu, Shin-Ting and Márquez, Mercedes R. G. which appeared in Proceedings of Sibgrapi 2003. © 2004 IEEE.

vertex that is farther away than that maximum distance is accepted as part of the new simplified polyline, and it becomes the new initial vertex for further simplification [10].

From detailed study of mathematical similarity and discrepancy measures, the Douglas-Peucker algorithm is pointed out as the most visually effective line simplification algorithm [1, 7]. Whereas vertex reduction uses closeness of vertices as a rejection criterion, the Douglas-Peucker algorithm uses the closeness of a vertex to the simplified polyline. It is a recursive procedure that starts with a line segment whose extreme vertices coincide with the extreme vertices v_1 and v_n of the polyline given as a list of n vertices in sorted order. Each segment $v_k v_j$ is split at the farthest vertex v_i (where $k < i < j$) to it until the distance between the sequence of vertices $v_k \cdots v_i$ and $v_k v_i$ and the sequence of vertices $v_i \cdots v_j$ and $v_i v_j$ are less than a fixed tolerance.

Saalfeld performed a thorough analysis of the Douglas-Peucker algorithm and listed in [9] a set of its key properties. Besides, on the basis of the hull property of the simplified polyline obtained from the Douglas-Peucker algorithm, he proposed, with proof, to use a point-on-convex hull test and the sidedness concept for detecting possible topological conflicts. He used the dynamic convex hull algorithm presented by Hershberger and Snoeyink [3] to efficiently maintain and access the current convex hull at each refinement stage.

This paper presents yet three contributions to deal with the self-intersection problem in the Douglas-Peucker algorithm. The first is a proof for the fact that the simplification of a star-shaped polyline by the Douglas-Peucker method will never result in a self-intersecting polyline. The second is a procedure for trivially discarding segments of a simplified polyline that do not intersect. And finally, on basis of first and second contributions we propose a non-self-intersecting Douglas-Peucker algorithm for any polyline.

In Section 2, the Douglas-Peucker algorithm is briefly described for completeness. The proof of the sufficiency conditions for non-self-intersections is presented in Section 3. Section 4 gives a strategy for eliminating possible conflicts between simplified sub-polylines. In Section 5, we describe our polyline simplification algorithm that integrates these two properties into the classic Douglas-Peucker to ensure topological equivalence between the original and the simplified polylines for any specified tolerance. Section 6 details a complexity analysis of the algorithm. Afterwards, some results are shown in Section 7. Finally, in Section 8, our future research directions are presented.

2   The Douglas-Peucker Algorithm

Besides its good visual results, the Douglas-Peucker algorithm is very simple to program and works for any dimension, once it only relies on the distance between points and lines. Several implementations are available at sites of the Internet [2, 10]. Its basic rule is that the approximation must contain (a subset of) the original data points and all the original data points must lie within a certain predefined distance to the approximation.

Given a polyline $\mathcal{P}$ and a tolerance ϵ as depicted in Figure 1.a. The Douglas-Peucker algorithm has a hierarchical structure starting with the single line segment e joining the first v_1 and last v_n vertices of the original polyline (Figure 1.b). Then the remaining vertices are tested for closeness to its approximating segment. If there are vertices further than a specified tolerance away from the segment, then the vertex v_i farthest from it is added to the previously simplified polyline. This creates a new approximation for the original polyline (Figure 1.c). Recursively this process continues for each approximating line segment (Figures 1.d,e) until all vertices of the original polyline satisfy the closeness condition (Figure 1.f).
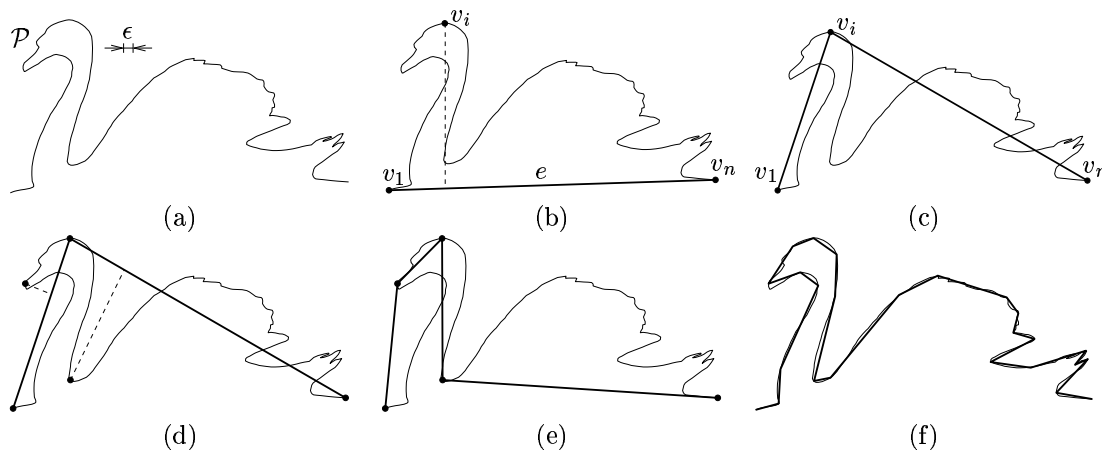

Figure 1: The basic Douglas-Peucker algorithm.

This algorithm has $O(mn)$ worst-case time complexity and $O(n \log n)$ expected time, where n is the number of input vertices and m is the number of the vertices of the simplified polyline. This is an output dependent algorithm and it will be very fast when m is small, that is when the approximation is coarser.

On the other hand, if the tolerance has a larger value, then the simplified polyline may intersect itself. Figure 2 illustrates a case for which three splittings on the initial segment e were sufficient for satisfying the tolerance condition, but could not avoid self-intersection. The trivial solution is to reduce the value of the tolerance, which may lead to a unnecessarily finer approximation.
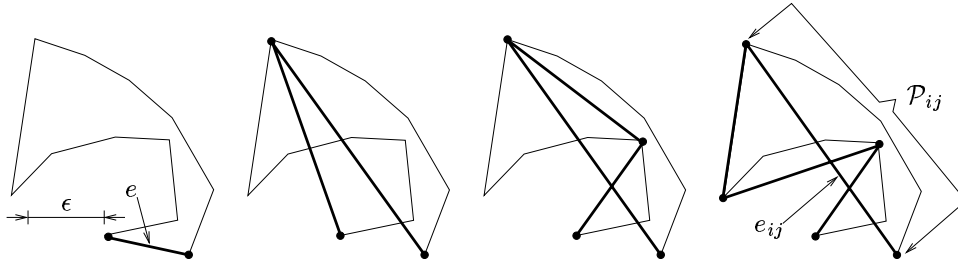
3

Figure 2: Self-intersection.

An alternative for solving this problem is to keep on applying the Douglas-Peucker procedure only on the part of the simplified polyline that presents topological conflicts. More specifically, Saalfeld [9] proposed the following procedure for detecting and solving topological conflicts in relation to each segment e_{ij} approximating a sub-polyline $\mathcal{P}_{ij}$ which is, in its turn, a subset of the original polyline $\mathcal{P}$:

1. Identify all vertices v_k (Figure 3.a), where $k < i$ or $k > j$, of the simplified polyline that lie in the region $\mathcal{R}_{ij}$ bordered by the sequence of vertices belonging to the sub-polyline $\mathcal{P}_{ij}$ and the segment e_{ij} (Figure 3.b).

2. Recursively split e_{ij} until every vertex, such as v_k, is out the region limited by the simplified and the original polylines, as shown in Figure 3.c.
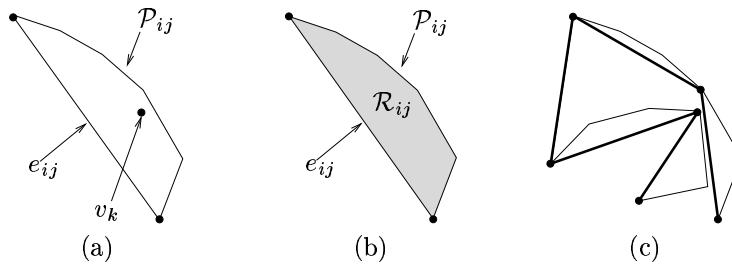

Figure 3: Further splitting until v_k becomes out.

3   Sufficiency for Non-Self-Intersections

Before demonstrating the sufficiency conditions for a simplified polyline generated by the Douglas-Peucker algorithm to be non-self-intersecting, we introduce some definitions.

Definition 3.1 *The **convex hull** $\mathcal{H}$ of a set of points $\mathcal{S}$ is the smallest set containing $\mathcal{S}$ that satisfies the following convexity property: for any pair of points $p, q \in \mathcal{H}$ the line segment pq is completely contained in $\mathcal{H}$.*

4

Definition 3.2 *Let $\mathcal{P}$ be an open polyline as a list of n vertices $v_1, v_2, \ldots, v_n$ in sorted order, and r a point on the segment $v_1 v_n$, except for the extreme vertices. We say that $\mathcal{P}$ is **star-shaped** with respect to r, if, for any point $p \in \mathcal{P}$, the ray $\overrightarrow{rp}$ intersects $\mathcal{P}$ only at p (Figure 4).*
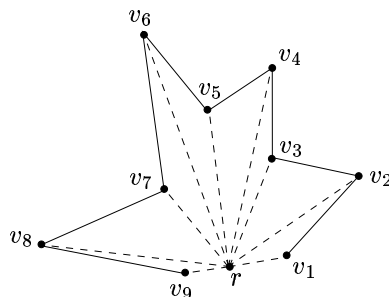

Figure 4: A star-shaped polyline.

Observe that the line segments rv_i, where $i \in \{1, 2, \ldots, n\}$, build with $(n-1)$ segments of a star-shaped polyline a set of edge-adjacent, but non-overlapping, triangles. Based on this, we are able to present the following proposition.

Proposition 3.1 *Given a star-shaped polyline $\mathcal{P}$ as a list of n vertices $v_1, v_2, \ldots, v_n$ in sorted order, its simplification from the Douglas-Peucker procedure cannot self-intersect.*

> **Proof:** *We prove the proposition by induction on m output vertices in the simplified polyline. When $m = 3$, the simplified polyline consists of the two extreme points, v_1 and v_n, and a vertex v_k, where $1 < k < n$, farthest from the line segment $v_1 v_n$. Trivially, the simplified polyline $v_1 v_k v_n$ does not self-intersect.*
>
> *When $m > 3$ and assume that the proposition is true for all values less than m. Let v_b be the newly inserted vertex and is farthest from the line segment $v_a v_c$, where $a < b < c$; the line segments $v_a v_b$ and $v_b v_c$ only intersect at v_b and cannot intersect the simplified sub-polylines $v_1 \cdots v_a$ and $v_c \cdots v_n$, because such an intersection implies that the triangle $rv_a v_b$ and/or $rv_c v_b$ overlap another triangle $rv_k v_l$, where $k \neq l$ and $k, l = 1, \ldots, a, c, \ldots, n$, contradicting the star-shapeness property. Hence, the simplified polyline consisting of m vertices cannot self-intersect.*

4 Sufficiency for Non-Intersection between Sub-Polylines

The sidedness of a vertex with respect to the simplified and original polylines may not be preserved by the Douglas-Peucker algorithm. In his work, Saalfeld concluded that topological conflicts always occur when vertices of the simplified polyline change their sidedness. He also suggested to use the data structure presented by Hershberger and Snoeyink [3] for reducing the search space of potential conflicts at each refinement recursion as well.

5

We showed in Section 3 that, by applying the Douglas-Peucker algorithm on a star-shaped polyline, the simplified polyline never self-intersects. So a possible approach for simplifying a non-star-shaped polyline would be to first decompose it into star-shaped sub-polylines and then apply the Douglas-Peucker algorithm on each sub-polyline. Nevertheless, this approach cannot ensure that non-self-intersecting piecewise simplified sub-polylines do not cross. In this section, we present a sufficient condition for non-intersections between two sub-polylines. Before this, let us introduce the following definition, which is useful to distinguish the two sides of a polyline on a plane.

Definition 4.1 *A polyline* $\mathcal{P} = \{v_1, v_2, \ldots, v_n\}$ *is* **orientable** *if there exists a simply-connected region* $\mathcal{R}$ *whose boundary is the closed sequence* $v_1, v_2, \ldots, v_n, v_1$. *If the sorted order of the vertices is in the clockwise sense along the boundary of* $\mathcal{R}$, $\mathcal{P}$ *is said to be clockwise oriented with respect to* $\mathcal{R}$ *(Figure 5.a); otherwise, it is counter-clockwise oriented (Figure 5.b). The region* $\mathcal{R}$ *is called* **polyline region**.

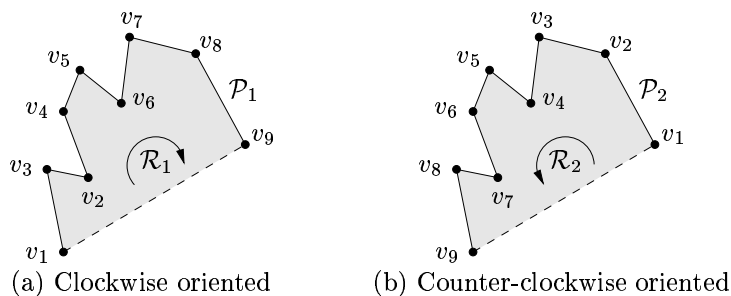(a) Clockwise oriented   (b) Counter-clockwise oriented

Figure 5: Orientable polylines.

From the spatial relationship between the corresponding regions of two orientable polylines, we may derive a condition for trivially discarding pairs of sub-polylines that do not self-intersect. It is worth noting that the results are applicable to the star-shaped polylines, once a star-shaped polyline is an orientable polyline. This is because that the union of a set of edge-adjacent and non-overlapping triangles with a common vertex is a simply-connected region and a star-shaped polyline is the boundary of this set of triangles.

Proposition 4.1 *Let* $\mathcal{P}$ *and* $\mathcal{Q}$ *be two orientable polylines that have at most some points in common. Let* $\mathcal{R}$ *and* $\mathcal{S}$ *be the respective polyline regions that have* **only** *these points in common. Let* $\mathcal{P}_s$ *and* $\mathcal{Q}_s$ *be the corresponding simplifications of* $\mathcal{P}$ *and* $\mathcal{Q}$. *If* $\mathcal{P}_s \in \mathcal{R}$, *where* $\mathcal{Q}_s \in \mathcal{S}$, *then* $\mathcal{P}_s$ *and* $\mathcal{Q}_s$ *have at most those points in common.*

6

Proof: *If $\mathcal{P}_s$ and $\mathcal{Q}_s$ have other points in common, then $\mathcal{R}$ and $\mathcal{S}$ must have more common points, because $\mathcal{P}_s \in \mathcal{R}$ and $\mathcal{Q}_s \in \mathcal{S}$. This contradicts the supposition. Hence, $\mathcal{P}_s$ and $\mathcal{Q}_s$ can share at most the common points of $\mathcal{P}$ and $\mathcal{Q}$.*

Proposition 4.1 tells us that if a polyline is partitioned into a set of orientable sub-polylines, such that their regions have at most the extreme or the boundary points in common, then we may restrict the search space for potential topological conflicts to the simplified sub-polylines that do not lie entirely in the region of their corresponding original sub-polylines. Figure 6 presents two cases: (a) $\mathcal{P}_s \in \mathcal{R}$, $\mathcal{Q}_s \in \mathcal{S}$; and (b) $\mathcal{P}_s \notin \mathcal{R}$, $\mathcal{Q}_s \notin \mathcal{S}$. Conflicts may only occur in the second case.
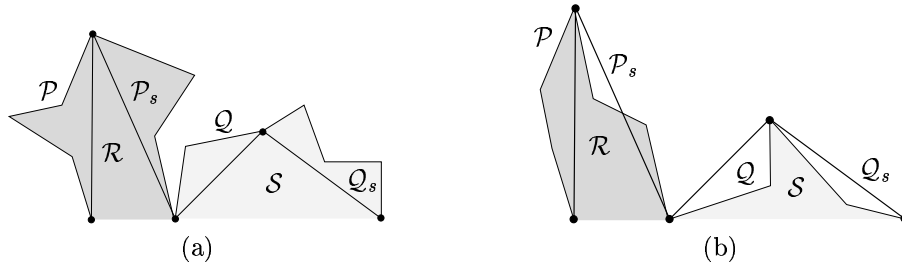

Figure 6: Potential topological conflicts.

Let us introduce one more definition before presenting a practical corollary.

Definition 4.2 *Two orientable polylines are called **separable**, if their polyline regions have at most the extreme or boundary points in common (Figure 7).*
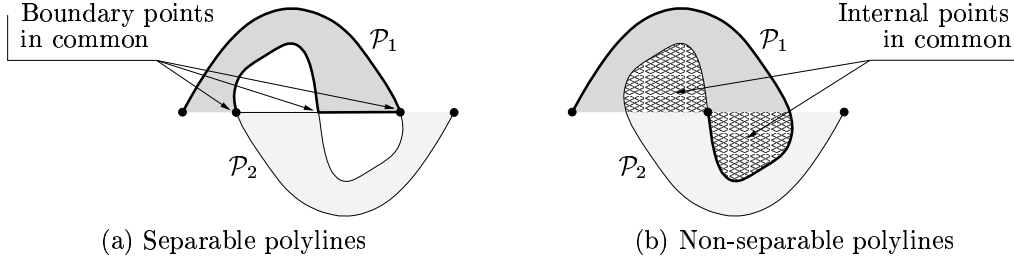

Figure 7: Separability of two polylines.

Corollary 4.1 *The simplified polylines of two separable polylines cannot intersect at their interior points if the simplified polylines lie entirely in the corresponding polyline regions.*

Proof: *If the simplified polylines intersect at their interior points, the corresponding polyline regions must intersect at some interior points, which violates the definition of separable polylines.*

5 A Non-Self-Intersecting Douglas-Peucker Algorithm

Based on the properties presented in Sections 3 and 4, we present a modified Douglas-Peucker simplification method that may avoid self-intersections along the recursive refinements of a polyline $\mathcal{P}$ with n input vertices $v_1, v_2, \ldots, v_n$ and that can detect the potential topological conflicts with a simple sidedness test as follows.

The algorithm comprises three steps:

1. Partition $\mathcal{P}$ into a set $\mathcal{C}$ of separable star-shaped sub-polylines.

2. Apply the Douglas-Peucker algorithm for every sub-polyline $\mathcal{C}_i \in \mathcal{C}$.

3. Beside topological conflicts between the sub-polylines.

5.1 Partition into separable star-shaped polylines

According to Corollary 4.1, the simplified polylines of two separable star-shaped polylines cannot intersect except at the boundary vertices of their polyline regions, if they lie entirely in the corresponding polyline regions. Moreover, Proposition 3.1 tells us that the simplified polyline of any star-shaped polyline cannot self-intersect. This motivates us to decompose an input polyline $\mathcal{P}$ into a set of separable star-shaped polylines before carrying out the Douglas-Peucker procedure at each one.

We devised a two-step procedure for partitioning any open polyline $\mathcal{P}$. In the first step, $\mathcal{P}$ is partitioned into a set of separable sub-polylines; and, in the second step, we apply a visibility algorithm to decompose each orientable sub-polyline into non-overlapping star-shaped pieces.

5.1.1 First Step

The support line of the vector $\overrightarrow{v_1 v_n}$ divides $\mathcal{P}$ in two parts: its left side and its right side (Figure 8.a). We determine the intersection points $u_1, u_2, \ldots, u_r$. Then, they are sorted along the direction of the vector $\overrightarrow{v_1 v_n}$, and $\mathcal{P}$ is split at them into subsequences of vertices, as shown in Figure 8.b. For instance, $u_1 \cdots u_2$ and $u_5 \cdots u_6$ are two distinct subsequences of $\mathcal{P}$.

We can determine all regions, $\mathcal{R}_1, \mathcal{R}_2 \ldots \mathcal{R}_l$, by simply tracing along the support line twice: one turn in the direction $\overrightarrow{v_1 v_n}$ and another turn in the direction $\overrightarrow{v_n v_1}$. This is because the side of a region alternates between in (I) and out (O) at each intersection point, as depicted in Figure 8.c. Hence, we may link the intersection points in sorted order to build the separable polylines on the left side, in the case $\mathcal{P}_2$, and, analogously on the right side, to obtain $\mathcal{P}_1$ and $\mathcal{P}_3$ (Figure 8.d).
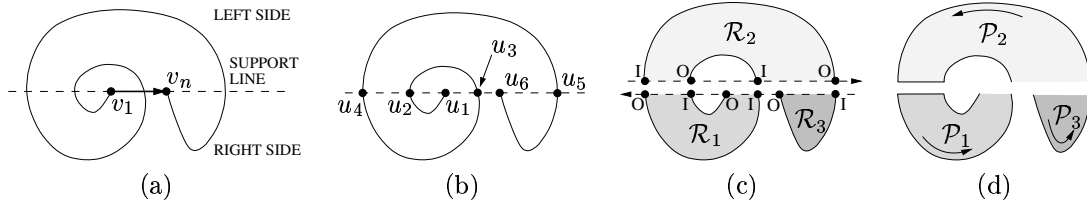
8

Figure 8: Partition into a set of separable sub-polylines.

5.1.2 Second Step

To determine a star-shaped sub-polyline of a polyline with respect to a point r is similar to the classic problem of computing a visibility polygon from r. Our implementation is based on the Hipke's linear time algorithm [4]. For decomposing a polyline with n vertices $\mathcal{P} = \{v_1, v_2, \ldots, v_n\}$ into a set of star-shaped polylines, we apply the algorithm on $\mathcal{P}$ to obtain a visibility polygon $\mathcal{V}$; then we replace $\mathcal{P}$ by $\mathcal{P}$-$\mathcal{V}$ and apply the algorithm recursively until no vertex remains.

For the sake of completeness, an outline of Hipke's algorithm is given in this section. The algorithm scans the n vertices of the polyline $\mathcal{P}$ in sorted order and chooses the visible ones on the basis of the tracking sense at each vertex and on the mode of operation, which depends only on the current and previous tracking senses.

According to the signal of the turn angle ϕ, that is the angle between the segments rv_{i-1} and rv_i, two senses are distinguished: forward or positive (Figure 9.a), and backward or negative (Figure 9.b).

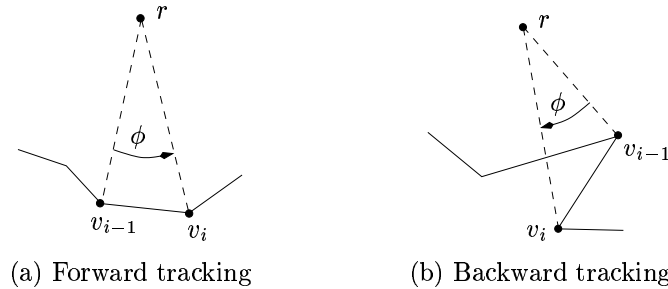(a) Forward tracking   (b) Backward tracking

Figure 9: Tracking senses.

Backward trackings are further subdivided into inward (the segment $v_{i-1}v_i$ lies between the segment $v_{i-2}v_{i-1}$ and the point r, as shows Figure 10.a) and outward (the segment $v_{i-2}v_{i-1}$ lies between the segment $v_{i-1}v_i$ and the point r, as illustrates Figure 10.b).

Three modes are distinguished: normal, skip and curl. In the normal mode, the current vertex is inside the scanned star-shaped polyline $\mathcal{V}$. In both skip and curl modes, the current vertex is not visible and the algorithm waits to come back into the area of $\mathcal{V}$. Switching from the normal
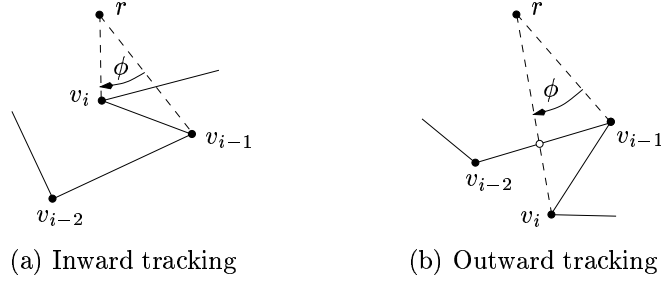
(a) Inward tracking   (b) Outward tracking

Figure 10: Backward trackings subdivision.

mode to one of these two modes depends on the way how the tracking tends to progress when it passes from a visible to a non-visible vertex: when it seems to advance spirally, as shown in Figure 11.a, the vertices will be further scanned in the curl mode; otherwise, we enter the skip mode (Figure 11.b).
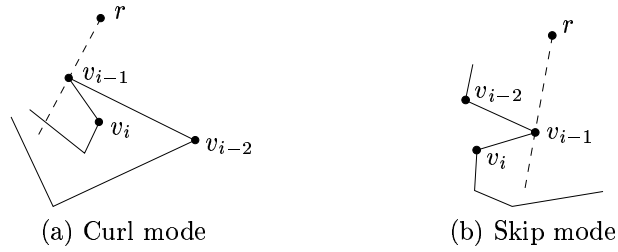

(a) Curl mode   (b) Skip mode

Figure 11: Modes for handling non-visible vertices.

For distinguishing correctly the visible from non-visible vertices, the algorithm demands a stack of *skip segments* that grows, whenever the skip mode is activated, or shrinks, if any skip segment becomes obscured. Each *skip segment* is represented by its first point s and its end point q. The points at which the polyline starts curling are also represented as curl points.

The procedure starts at $v_i = v_2$ in the normal operation mode. It also considers that the previous tracking sense was forward. The vertices are scanned sequentially from $i = 2$ to n. We may be in one of the three modes of operation at each vertex v_i:

1. normal mode: if the tracking sense is

   - forward: if the previous sense was in-backward and rv_i intersects the support line of $v_{i-2}v_{i-1}$, set v_{i-1} as the curl point, skip v_i and switch to the curl mode; otherwise, v_i is inserted in the output list $\mathcal{V}$.

   - out-backward: pile v_{i-1} as the initial point of a *skip segment*, skip v_i, and switch to the skip mode.

10

- in-backward: if the segment $v_{i-1}v_i$ crosses any *skip segment*, remove from the output list $\mathcal{V}$ all vertices v_j, with $j < i$, until v_k (the first visible vertex after the *skip segment*), pop out all *skip segments* with end points v_l, where $l > k$, and switch to the skip mode; otherwise, remove from the output list $\mathcal{V}$ all vertices v_j, with $j < i$, until a segment in $\mathcal{V}$ intersects the ray beginning at r and passing through v_i, and insert vertex v_i in $\mathcal{V}$.

2. skip mode: if the segment $v_{i-1}v_i$ intersects the ray beginning at r and passing through s (the first point of the current *skip segment*), then determine the intersection point q, set it as the end point of the current *skip segment*, add v_i in the output list $\mathcal{V}$, and switch to the normal mode; otherwise, skip vertex v_i and keep on the skip mode.

3. curl mode: if the segment $v_{i-1}v_i$ intersects the ray beginning at r and passing through c (the curl point), then add v_i in the output list $\mathcal{V}$ and switch to the normal mode; otherwise, skip v_i, and keep on the curl mode.

Figure 12 illustrates an application of this procedure to the polyline $\mathcal{P}$ in Figure 12.a. In the figure, the current vertex is highlighted with a black square, the intersection points are indicated by the white circles, and the current tracking sense is pointed out by a curved arrow.

The algorithm starts in the normal mode at the vertex v_2. The (current) tracking sense from v_1 to v_2 is forward (Figure 12.b). It continues moving forward, until it finds the out-backward tracking from v_6 to v_7 (Figure 12.c). It then sets v_6 as the current skip point s and enters the skip mode. It begins skipping vertices, until it reaches the end point q – the intersection between the ray rs and the segment $v_{11}v_{12}$ – of the current *skip segment* (Figure 12.d). In consequence, it returns to the normal mode.

The algorithm moves further forward until the in-backward tracking is detected at v_{15} (Figure 12.e). It begins removing vertices previously inserted in the output list, until it finds the intersection between the segment $v_{12}v_{13}$ and the ray rv_{15} (Figure 12.f). As the movement from v_{15} to v_{16} is in-backward (Figure 12.g), it pops out the skip segment sq and discards all vertices from the output list, until it finds an intersection between the ray rv_{16} and the segment v_5v_6 (Figure 12.h). At v_{17} the spiral movement is identified, then v_{16} is assigned as the curl point c and the procedure enters the curl mode (Figure 12.i). In consequence, all subsequent vertices are ignored, until the intersection between rc and $v_{18}v_{19}$ is reached. The normal mode is then restored.

The procedure reevaluates the state of the output list with regard to v_{19}. As the tracking from v_{18} to v_{19} is in-backward, it have to remove all the vertices from the output list till the

11

intersection between rv_{19} and v_4v_5 (Figure 12.k). Further forward progress until the end vertex v_{21} results in the star-shaped polyline $\mathcal{V}$ (Figure 12.l).
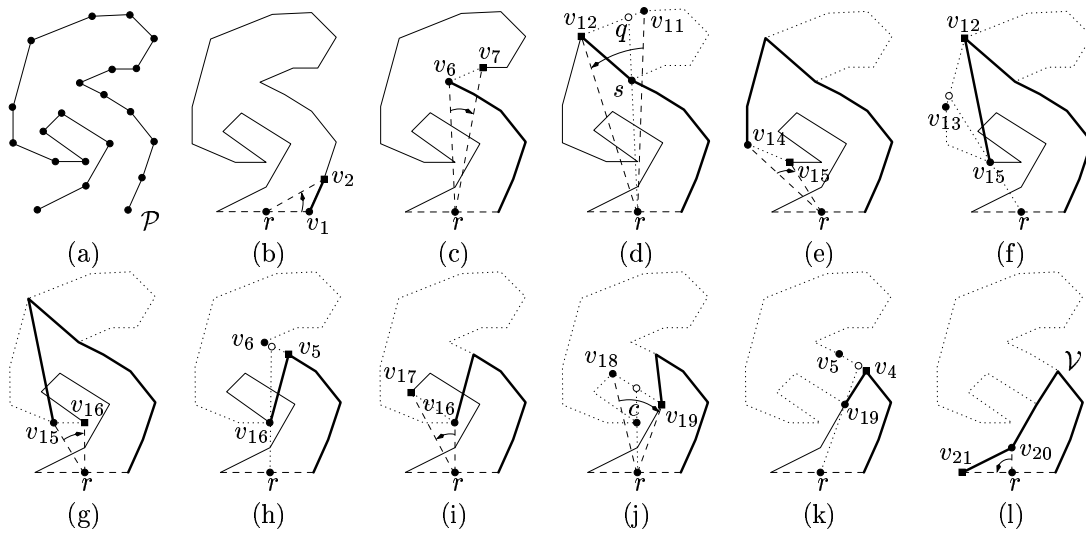

Figure 12: Determination of a star-shaped polyline.

5.2   Beside Topological Conflicts

As already stated, the intersections may only occur among the simplified sub-polylines belonging to the distinct polyline regions. Hence, we propose to include in the Douglas-Peucker algorithm the test whether a simplified polyline is outside of the corresponding polyline regions.

If we consider that each star-shaped polyline is clockwise oriented, this sidedness test is equivalent to the condition that there is a sample v_j between v_i and v_k, where $i < j < k$, that violates the expression

$$v_kv_j \times v_jv_i \geq 0. \tag{1}$$

This allows us to reduce the search space for the topological conflicts and to carry out the intersection test only between the star-shaped polylines that violate Eq. 1.

Figure 13 illustrates the proposed procedure. Figure 13.a presents three segments (d_1, d_2 and d_3) that do not lie in the corresponding polyline region. There is only a vertex that violates Eq. 1 with regard to d_1. Hence, d_1 is further refined until Eq. 1 is satisfied (Figure 13.b).
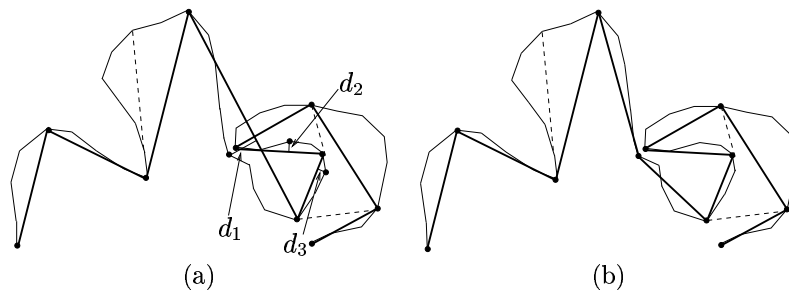
Figure 13: Besiding topological conflicts.

6 Complexity Analysis

There is a variety of ways to implement our algorithm, since a number of algorithms is available for solving each subproblem. In this section we present a time complexity analysis of our algorithm as we implemented it.

For partitioning an input polyline with n vertices into a set of separable polylines, we first find all points that intersect the support line. This is performed in $O(n)$ time. After then, we sort r intersection points in $O(r \log r)$ time. We finally join the pieces to build a set of separable simple polylines with $O(r)$ time. Therefore, the worst-case time complexity of this step is $O(n + r \log r + r)$, or simply $O(n + r \log r)$.

The worst case for partitioning any polyline with n vertices into a set of l star-shaped ones occurs when it is already separable, that is $r = 1$. Since the algorithm for determining a star-shaped polyline we used is linear, the worst-case complexity of the whole partitioning is $O(ln)$. However, the number of star-shaped polylines l is always less than the number of output vertices m. So, we can upperbound its complexity to $O(mn)$.

We know as well that the algorithm for finding the farthest vertex of a sequence, used exhaustively in Douglas-Peucker simplification, is linear. Then, the Douglas-Peucker algorithm runs for each star-shaped polyline in $O(kj)$ time, where k and j are, respectively, the number of input and output vertices. Since the worst case happens whenever we have a unique star-shaped polyline with n vertices, the worst-case complexity of this step is also $O(mn)$.

For handling the topological conflicts between the star-shaped polylines, we have to verify every external segment of a simplified polyline against all segments of the others. In the worst case, this check is performed in $O(m^2)$ time.

Finally, for regrouping the separable polylines, built in preprocessing, into one sequence as the output, we have just to pass through every output vertex sequentially, what takes $O(m)$ time.

For computing the whole complexity, we add together the worst-case complexity of each step, that results in

$$O(n + r \log r + mn + m^2 + m) \tag{2}$$

Since the number of input vertices n is always greater than or equal to the number of output vertices m, the three last terms of the sum in Equation 2 fulfills the inequality

$$mn + m^2 + m \leq mn + mn + mn.$$

In addition, once the preceding vertex and the subsequent vertex of the intersection points are always included in the simplified polyline, the number of output vertices m is greater than the number of intersection points r. Thus, for the first two terms of the sum in Equation 2, we have

$$n + r \log r < n + m \log m < n + mn \leq mn + mn.$$

Summarizing, the worst-case complexity of our algorithm is $O(mn)$, which is equivalent to the time complexity of the original Douglas-Peucker algorithm we used in our implementation.

7   Results

To evaluate the algorithm we implemented, we present in this section some results we obtained on the simplification of the outlines of the continents from the atlas data available at [13]. Comparisons with the results from the original Douglas-Peucker algorithm [1] are also provided.

Figure 14.a illustrates a Europe's outline with 53,626 vertices. Its Douglas-Peucker simplification with a tolerance of 4.0 contains 18 vertices and two self-intersections, as depicted in Figure 14.b. The reduced outline from our algorithm (Figure 14.c) has no self-intersections, although it possesses much more vertices (46), 22 of which were inserted during the partition into separable polylines and 24 were added for splitting them into star-shaped ones. It is worth noting that in Figure 14.c the shape of Scandinavian Peninsula is much more distorted than in Figure 14.b. It is due to the partition of the original outline into star-shaped polylines before performing the Douglas-Peucker algorithm.

Figure 15.a shows an Asia's outline with 87,337 vertices. Figures 15.b and 15.c are respectively the simplified outlines, with tolerance of 8.6, from the Douglas-Peucker and our algorithm. Because of high tolerance and wavy outline, the original Douglas-Peucker algorithm delivers self-

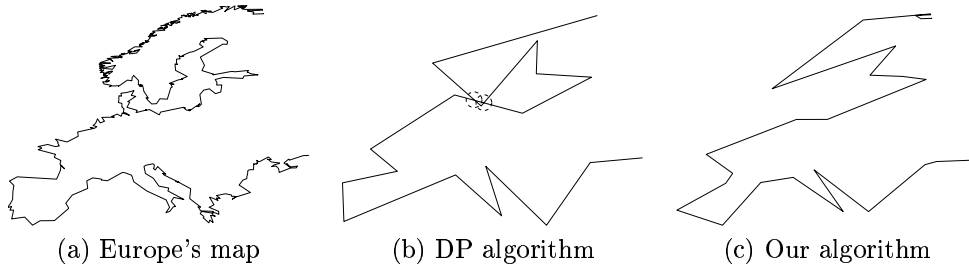(a) Europe's map   (b) DP algorithm   (c) Our algorithm

Figure 14: Comparison between Douglas-Peucker and our approach simplifications of Europe's map with tolerance of 4.0.

intersections in a very reduced outline, with only 13 vertices. As expected, the result of our algorithm is self-intersection free, at cost of more vertices (35) in the output: 15 of them were inserted while the original polyline was splitted into star-shaped sub-polylines.

(a) Asia's map

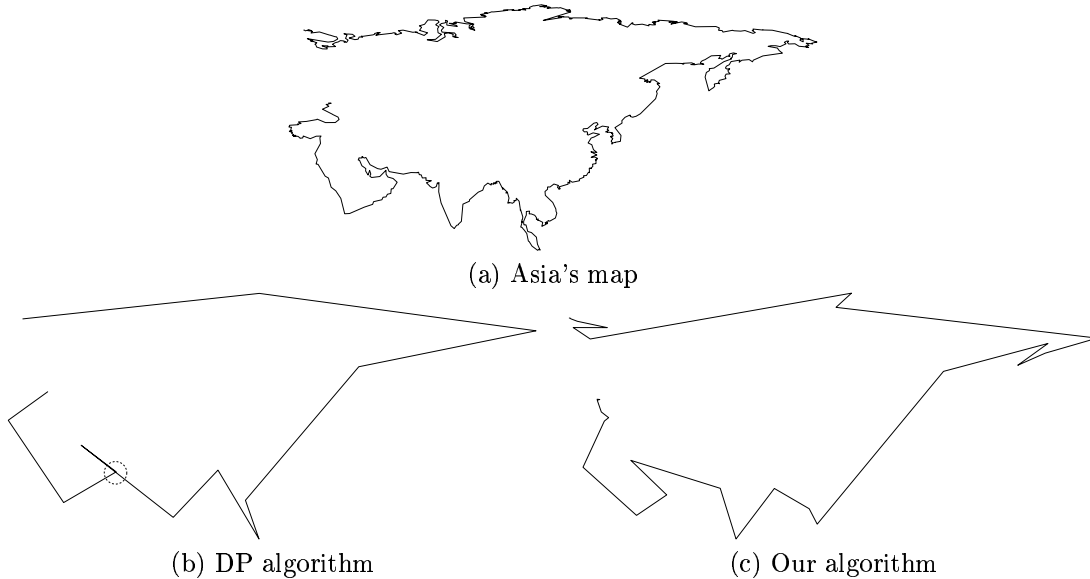(b) DP algorithm   (c) Our algorithm

Figure 15: Comparison between Douglas-Peucker and our approach simplifications of Asia's map with tolerance of 8.6.

Figures 16.a and 16.b present, respectively, the simplifications with tolerance of 0.3 of the North America's outline from the Douglas-Peucker and our proposal. The number of vertices in the original outline is 105,499. The Douglas-Peucker algorithm reduces it to 993 vertices, but because of highly wavy borderline 54 self-intersections were resulted. Although our algorithm delivers much more number of vertices in the output (1,717), it contains no self-intersections. Visually, the results are much alike.

Figure 17.a illustrates an Oceania's outline with 27,577 vertices. The tolerance of 4.2 was used for simplifications. From the Douglas-Peucker algorithm 9 vertices were yielded, two of which are

(a) DP algorithm   (b) Our algorithm

Figure 16: Simplifications of the North America's outline with tolerance of 0.3.

very closely laid in the position pointed in Figure 17.b. Figure 17.c shows our simplification with 27 vertices, where 16 vertices introduced in preprocessing are laid closely in the position indicated by the arrow in Figure 17.c. As expected, for avoiding self-intersections, low tolerance may lead to an excessive number of star-shaped sub-polylines in the protruding polyline.

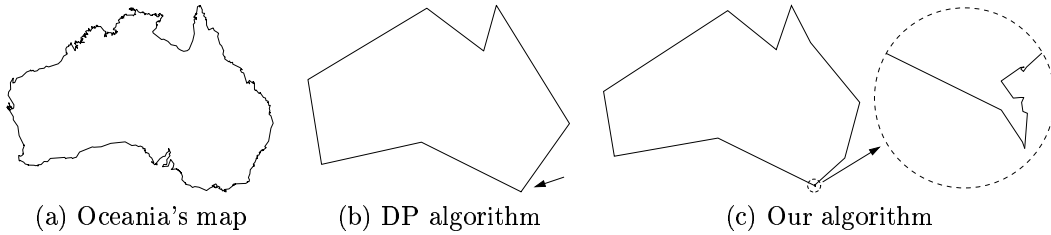(a) Oceania's map   (b) DP algorithm   (c) Our algorithm

Figure 17: Simplifications of the Oceania's outline with tolerance of 4.2.

Figure 18.a illustrates an Africa's outline with 28,653 vertices. Its Douglas-Peucker simplification, with tolerance of 4.5, contains 11 vertices, as depicted in Figure 18.b. Because its undulations are relatively larger with respect to the specified tolerance, the number of vertices in the output of our algorithm is almost the same: 19. Moreover, the reduced outlines in Figure 18.b and Figure 18.c are very similar.

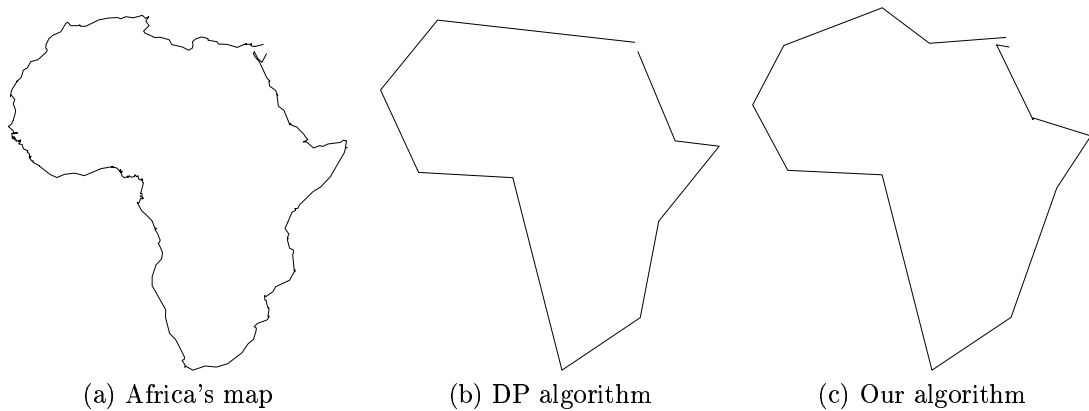(a) Africa's map   (b) DP algorithm   (c) Our algorithm

Figure 18: Simplifications of the Africa's outline with tolerance of 4.5.

Table 1 summarizes the results obtained with the use of the Douglas-Peucker algorithm and our algorithm. The column "Input" contains the number of input vertices. The simplification tolerance is given in the column "ϵ". The output is recorded in the column "Output". The number of occurrences of self-intersections is in the column "S-I". We also show the number of inserted vertices in the preprocessing ("Pre") and during the partition into the star-shaped polylines and the elimination of topological conflicts ("Post"). Additionally, the processing time for each simplification is given in seconds in the column "t(s)".

Data			DP algorithm			Our algorithm			
Map	Input	ϵ	Output	S-I	t(s)	Output	Pre	Post	t(s)
Oceania	27,577	4.2	9	0	1.010	27	16	11	1.390
Africa	28,653	4.5	11	0	1.070	19	6	13	1.450
Europe	53,626	4.0	18	2	1.990	46	22	24	2.690
Asia	87,337	8.6	13	1	3.260	35	15	20	4.400
North America	105,499	0.3	993	54	4.170	1,717	103	1,614	7.510

Table 1: Summary of maps simplification results.

Observe that our procedure has a simplification ratio comparable to the Douglas-Peucker procedure with the advantage that no self-intersection appears. We did not perform detailed measurements for analyzing the visual effect of our algorithm. However, the simplified polylines we obtained are fair from our subjective judgment, once most of extra vertices that our procedure introduced are in the portion of the simplified polyline that has a large variation in the curvature.

8 Concluding Remarks

In this paper, we present yet an improvement to the classic Douglas-Peucker line simplification algorithm in terms of preventing self-intersections. We derive, with proof, two sufficient conditions. On the basis of these conditions, we also propose a way to integrate them into the Douglas-Peucker algorithm and to reduce the search space for the topological violations. The concept of polyline region was not only useful for trivially discarding star-shaped sub-polylines that cannot cause any topological conflicts, but also for controlling further local refinements as well.

Another contribution of our work is the decomposition of the problem into a set of sub-problems whose solutions are well-known. This facilitated the implementation of our algorithm and the validation of our idea. From the experiments we carried out, the results of our algorithm are similar to the ones produced by the classic Douglas-Peucker algorithm, except at the vicinity of the connecting vertices of the star-shaped polylines.

Our algorithm tends to refine much more. The focus of this work is the robustness of simplification, and not the efficiency. Hence, we did not investigate its performance thoroughly. From a quick analysis, our implemented algorithm has, like the classic Douglas-Peucker algorithm, $O(mn)$ worst-case time, where m is the number of output vertices, and n the number of input vertices. This performance is output dependent.

As further work, we may improve the performance of our algorithm by optimizing the decomposition of the input polyline into a set of separable polylines. It is also interesting to compare the performance of the optimized algorithm with the existing ones.

Another point of our interest is to reduce the unnecessary refinements in the vicinity of connecting vertices of the star-shaped polylines. Finally, we would like to extend our algorithm to a multiresolution version that enables a fast description of an object at different levels of details.

Acknowledgments

References

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

[2] J. De Halleux. An C++ implementation of Douglas-Peucker line approximation algorithm, 2003. http://www.codeproject.com/useritems/dphull.asp.

[3] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *The 5th International Symposium on Spatial Data Handling*, pages 134–143, 1992.

[4] C. A. Hipke. Computing visibility polygons with LEDA, 1996.

[5] G. F. Jenks. Lines, computers and human frailties. In *Annals of the Association of American Geographers*, pages 1–10, 1981.

[6] T. Lang. Rules for robot draughtsmen. *Geographical Magazine*, 22:50–51, 1969.

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

[8] K. Reumann and A. P. M. Witkam. Optimizing curve segmentation in computer graphics. In *Proceedings of the International Computing Symposium*, pages 467–472, 1974.

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

[10] Dan Sunday. Geometric algorithms. `http://geometryalgorithms.com/Archive/algorithm_0205/algorithm_0205.htm`.

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

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

[13] P. Wessel and W. H. F. Smith. A global self-consistent, hierarchical, high-resolution shoreline database. *Journal of Geophysical Research*, 101:8741–8743, 1996.

[14] Z. Zhao and A. Saalfeld. Linear-time sleeve-fitting polyline simplification algorithms. In *Proceedings of AutoCarto 13*, pages 214–223, 1997.