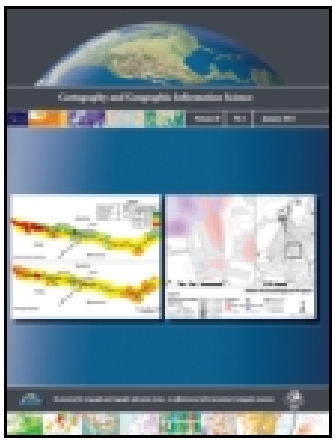

Cartography and Geographic Information Science
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tcag20

Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm

Alan Saalfeld

Published online: 14 Mar 2013.

To cite this article: Alan Saalfeld (1999) Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm, Cartography and Geographic Information Science, 26:1, 7-18, DOI: 10.1559/152304099782424901

To link to this article: http://dx.doi.org/10.1559/152304099782424901

Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm

Alan Saalfeld

ABSTRACT: We examine key properties of the Douglas-Peucker polyline simplification algorithm which are shared with many similar "vertex sub-sampling" algorithms. We examine how the Douglas-Peucker algorithm and similar algorithms can fail to maintain consistent or correct topological relations among features. We then prove that a simple test added to the stopping condition of Douglas-Peucker-like algorithms can guarantee that the resulting simplified polyline is topologically consistent with itself and with all of its neighboring features, and is correctly situated topologically with respect to all other features. We describe how a dynamically updated convex hull data structure may be used to efficiently detect and remove potential topological conflicts of the polyline with itself and with other features in that polyline's neighborhood.

KEYWORDS: Generalization, sub-sampling algorithms, approximation, convex hull

Introduction

Polyline simplification is a map generalization technique that replaces a linear map feature with a less complex representation of the same feature. The classic Douglas-Peucker polyline simplification algorithm (Douglas and Peucker 1973), also known among computer scientists as the Ramer simplification algorithm (Ramer 1972), selects a subset of a polyline's vertices and, from those fewer vertices, builds a new, simplified polyline which lies within a predefined distance ε of the original polyline. Because the simplified polyline may be displaced by a distance as large as ε from the original polyline, the simplified polyline may lose its proper topological relationship to other point or linear features lying at a distance less than ε from the original polyline. By the same token, however, other point and linear features that lie at a distance greater than ε from the original polyline cannot cause any topological problems whatsoever with the simplified polyline.

These observations give us two new simple criteria for detecting and correcting topological inconsistencies.

Alan Saalfeld is assistant professor in the Geodetic Science Program of the Department of Civil and Environmental Engineering and Geodetic Science, Ohio State University, Hitchcock Hall 470, 2070 Neil Avenue, Columbus, OH 43210-1275. E-mail: <saalfeld.1@ osu.edu>.

1. The ε-nearness of potentially conflicting features permits us to drastically limit our search space for potential conflicts. We will see that we can further limit the collection of potentially conflicting features that we must examine to *point* features, including polyline end- and shape-points. We will also see that a necessary (but not sufficient) condition for those point features to conflict with the simplified line is that the point features must lie within the closed convex hull of the original polyline, as well as being within ε of the original polyline.
2. We assume that our input maps have been topologically edited and are free from topological inconsistencies. This assumption guarantees that the nearest external point, shape-point, or end-point to a given linear feature is some distance $\delta > 0$ from the linear feature itself. If we choose our threshold, ε, to be less than δ, then we remove all possibility of any other feature conflicting with the feature undergoing simplification.

We describe efficient local tests for selecting additional vertices to add to the simplified polyline. These new vertices will guarantee that the simplified polyline preserves the left-sidedness or right-sidedness of all other nearby features. Our techniques build upon and extend the use of dynamic convex hull algorithms, neighborhood screening, and efficient measures of topological consistency between features.

Topological Correctness and Consistency

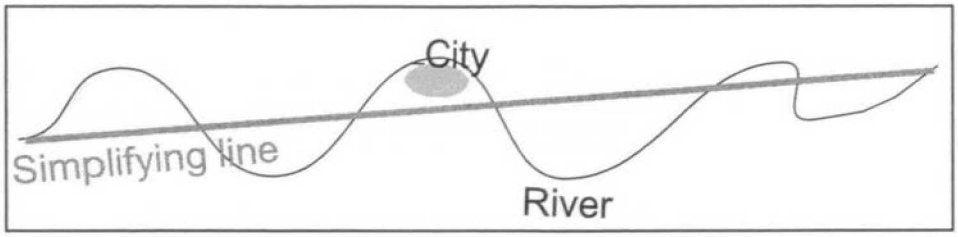

Figure 1. A simplification algorithm may locate a point on the wrong side of a line feature.

Let us begin by illustrating some things that can go wrong with simplification algorithms. Our first observation is that polyline simplification *always* alters the topological character or sidedness of some points, namely, those points that (1) are trapped between the original line and the simplifying line, or (2) lie either on the simplifying line or on the original line, but not on both. If the points in question are not feature points or parts of other feature lines, then their wrong-sidedness or overlap is of no consequence to the topological structure of features. Conversely, when features (point or line) lose their proper sidedness with respect to other features, then topological correctness is lost. When features (point or line) lose their disjointness from other features, then topological/geometric consistency is lost.

We concentrate on changes in feature topology, i.e., point features that lie in the wrong polygon or on the wrong side of a linear feature (see Figure 1), and segments that intersect after simplification when the original lines did not intersect prior to simplification. Intersecting segments may come from within the same polyline (Figure 2) or from different polylines (Figure 3).

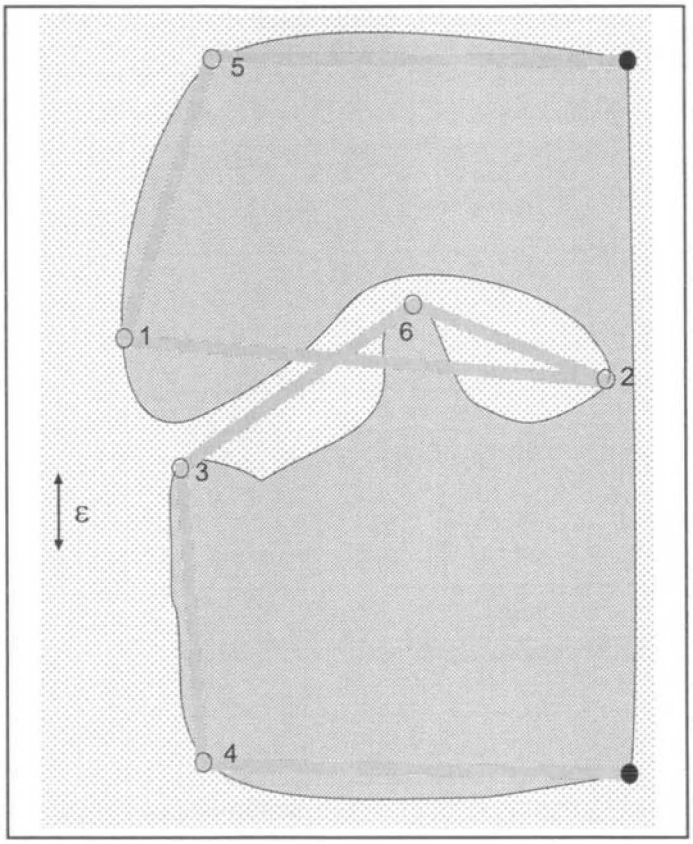

Figure 2. The Douglas-Peucker algorithm may produce a self-intersecting simplified polyline.

Polyline Simplification

To place our results in a broader context of polyline simplification algorithms, we distinguish three approaches to polyline simplification. A fully comprehensive approach modifies the complete collection of polylines and other features *en masse* or as a whole, taking the topological relationships among all features into consideration during the adjustment process. A locally aware approach modifies a single polyline *en suite* or in context, taking the topological relationships of the polyline with other nearby features into consideration during the adjustment process. A third approach modifies the polyline *in vacuo* or in isolation, producing potential topological conflicts that must be addressed and resolved at some later processing stage.

The comprehensive approach is necessarily more complex because more data must be processed simultaneously to guarantee topological consistency of the entire map. The last approach, modification in isolation, is obviously easiest to implement, but it then places a considerable burden on the overall generalization system for detecting and correcting any resulting inconsistencies. In fact, polyline simplification done in isolation may produce situations that cannot be corrected with post-processing. The Douglas-Peucker algorithm is a simplification algorithm which processes a single linear feature in isolation from other neighboring features. The

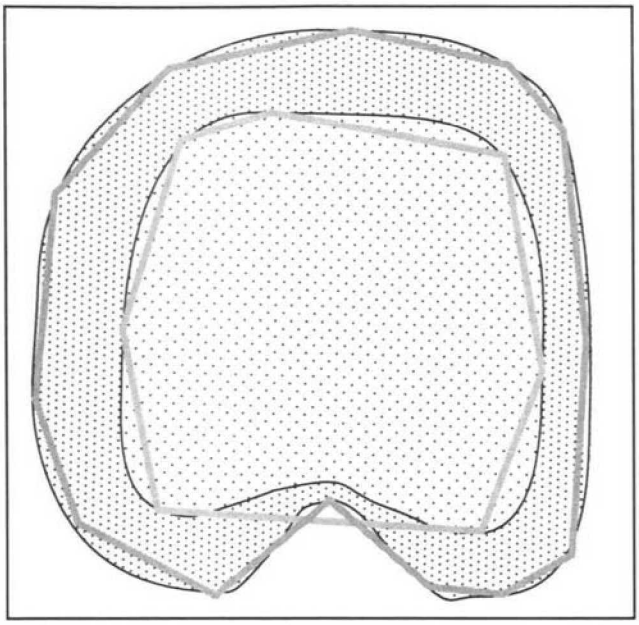

Figure 3. A simplification algorithm may produce intersecting polylines.

National Ocean Service, however, rejected the use of the Douglas-Peucker algorithm and similar algorithms for simplifying coastlines in 1985 because they all produced topological inconsistencies (see Figures 1, 2, and 3) that could not be easily detected or resolved.

Unfortunately, until very recently, nearly all work on polyline simplification has dealt with the more tractable "single polyline sans context" problem. Two recent papers offer contextual (de Berg and Kreveld 1995) and global or holistic (Jones et al. 1995) approaches to generalization. The contextual approach in the cited paper is computationally expensive; and the global approach in the other paper focuses on planar subdivisions of area features.

Douglas-Peucker Algorithm

The classic Douglas-Peucker polyline simplification algorithm falls into the category of vertex subsampling algorithms that modify individual polylines in isolation. Only the polyline's sequence of vertices, not the surrounding features, enter into the simplification process. A sub-sequence of the original polyline's vertices constitutes the output of the algorithm, representing the vertices, in order, of the simplified output polyline. Below we present those elementary definitions that would enable us to formulate the pseudo-code for the recursive version of the Douglas-Peucker polyline simplification algorithm (see Boxes 1 and 2). Then we discuss some important properties of the algorithm, including areas where it fails to distinguish important polyline differences.

Let $\{v_0, v_1, v_2, \ldots, v_n\}$ be an ordered sequence of vertices. Let P_{ij} represent the polyline that links v_i to v_{i+1} to ... to v_{j-1} to v_j, in that order. Then P_{0n} represents the entire polyline. Similarly, let $e_{ij} = [v_i, v_j]$ represent the straight line segment (edge) linking v_i to v_j. For any edge e_{ij} and any vertex v_k let $\delta(v_k, e_{ij})$ represent the distance from vertex v_k to edge e_{ij}. Let $P(\{v_{i0}, v_{i1}, v_{i2}, \ldots, v_{im}\})$ represent the simplified polyline formed by linking the sub-sequence of vertices, $\{v_{i0}, v_{i1}, v_{i2}, \ldots, v_{im}\}$, in that order. Note that for this to be a sub-sequence, we must have $0 \leq i_0 < i_1 < i_2 < \ldots i_m \leq n$. Finally, let $\varepsilon > 0$ be the maximum distance that the simplified polyline may deviate from the given polyline P_{0n}.

The result of the first recursive call to simplify the polyline P_{0n} is illustrated in Figure 4. We note that the most expensive pseudo-coded step in our recursive procedure is finding the farthest intermediate vertex from the current approximating segment: Find $k \in (i, j)$ with $\delta(v_k, e_{ij}) \geq \delta(v_s, e_{ij})$, $s \in (i,j)$. Hershberger and Snoeyink (1992) have observed that the selected point v_k must lie on the convex hull of the candidate points; and they have built data

```
Simplify (P_ij, ε){
  If j > i + 1, then {
    Find k ∈ (i,j) with δ(v_k,e_ij) ≥ δ(v_s,e_ij), s ∈ (i,j);
    If δ (v_k, e_ij) > ε, then {
      Add v_k to the output polyline vertex set V;
      Simplify (P_ik,ε);        // Recursively handle left sub-polyline
      Simplify (P_kj, ε);       // Recursively handle right sub-polyline
      }
    }
  }
```

Box 1. Recursively called pseudo-code for simplifying any subpolyline P_{ij}.

Initialize output polyline vertex set V to $\{v_0, v_n\}$;
Simplify (P_{0n}, ε);
Output V;

Box 2. Pseudo-code for the main program to simplify the polyline P_{0n}.

structures to maintain and access the current convex hull efficiently. The convex hull of a set of points is a key structure in our work as well, so we provide here a few definitions and key properties of convex hulls:

- A **convex set C** is a set of points satisfying the following property (called convexity): If p_1 and p_2 are any two points in C, then all points on the line segment joining p_1 and p_2 are also in C. That line segment may be written as the set: $\{p_1 + \lambda (p_2 - p_1) | 0 \leq \lambda \leq 1\}$.
- The **convex hull of a set of points** is the boundary polygon of the smallest convex set containing those points.
- The **convex hull of a polyline** P_{0n} is the convex hull of its vertex points $\{v_0, v_1, v_2, \ldots, v_n\}$ (see Figure 5).

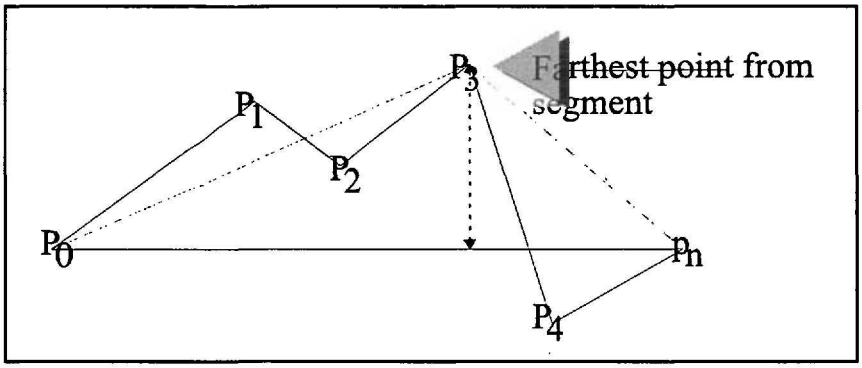

Figure 4. The Douglas-Peucker algorithm selects the farthest point from the approximating segment.

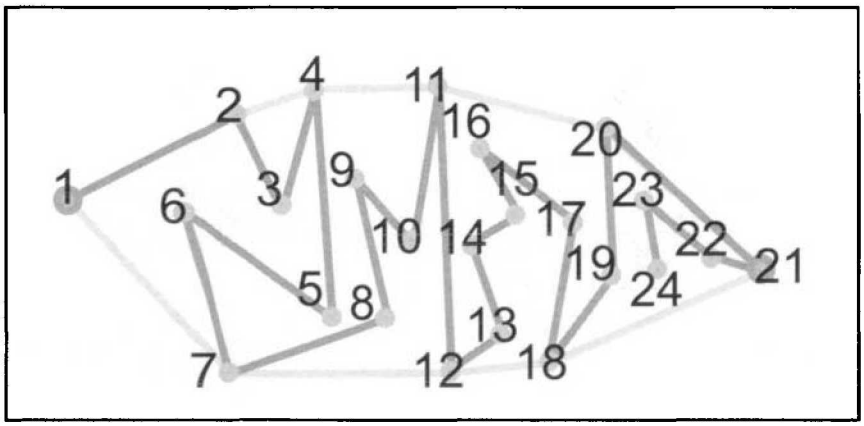

Figure 5. The convex hull of $P_{1,24}$ is P({1,2,4,11,20,21,18,12,7,1 }).

Properties of the Douglas-Peucker Algorithm

The following properties of the Douglas-Peucker algorithm are significant to our subsequent analysis and algorithm modification.

1. **Subset property**. The vertices of the simplified polyline are a subset of the vertices of the original polyline. Consequently, the simplified line cannot stray from the convex region determined by vertices of the original line.

2. **Greedy property.** The simplified polyline adds a vertex (or remains unchanged) at each step of the process and never backs up or changes its mind. Hence, the simplified line's vertex set grows bigger, never smaller, with each recursive call. Some authors refer to this type of simplification algorithm as "additive" and contrast such algorithms with "subtractive" algorithms which throw away vertex points one (or several) at a time (McMaster and Shea 1992).[1]

3. **Extreme point property.** The vertex v_k that is added to the vertex set while processing P_{ij} is a vertex on the current convex hull boundary of P_{ij}. As i and j change, the hull boundary changes; but it changes in a very predictable manner—see Theorem 1.

4. **Proximity properties.** The simplified polyline remains within a distance ε of the original polyline (because the original polyline contributes

[1] One could easily frame the Douglas-Peucker algorithm as a "subtractive" algorithm: Start with the set of all polyline vertices, then throw away all vertices between i and j when the farthest intermediate vertex from segment e_{ij} is closer than ε.

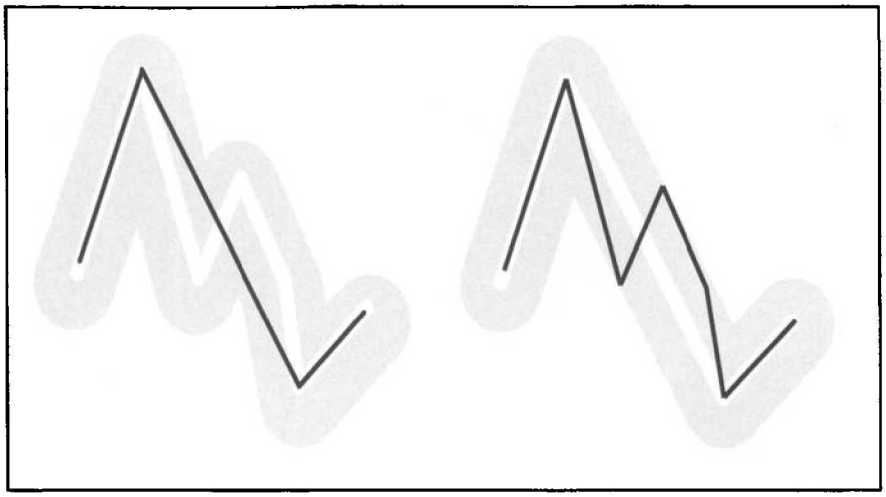

Figure 6. The new line lies within an ε-buffer of the original line and vice versa.

vertices to the simplified line until all of the remaining vertices of the original polyline lie within a distance ε of the simplified polyline—see the left side of Figure 6). Conversely, the original polyline remains within an ε-buffer of the simplified line (see the right side of Figure 6). We treat ε as a parameter that we may adjust to keep our simplified line from moving too far from the original line.

5. **Uniqueness property.** At each recursive step, "Simplify (P_{ij}, e)," the choice of the next candidate vertex v_k between v_i and v_j to add to the simplified polyline's vertex set is uniquely determined (up to ties); and the candidate is independent of the choice of ε. The magnitude of ε does determine whether or not the candidate actually is added to the vertex set. If the points are in "general position" (i.e., no ties), then the Douglas-Peucker polyline simplification algorithm produces a unique vertex set for the simplified polyline. If there are ties, then the choice of the vertex to add to the simplified polyline's vertices can make a great difference. The choice of a good representative can reduce or even eliminate subsequent added vertices. Pathological examples of ties may be constructed in which the selection of a bad representative can precipitate a long sequence of otherwise unnecessary vertex additions. We will, therefore, assume throughout the remainder of our discussion that we do not have to deal with ties in choosing our next vertex to add to the vertex set of the simplified polyline.

The insertion of a vertex v_k into the simplified polyline P_{ij} creates two (local) sub-sequences of the vertices v_i, .., v_k and v_k, ..., v_j of the original polyline; and, from each of the two sub-sequences we must examine a most distant vertex (globally largest within the local sub-sequence) from the shortcut segment (e_{ik} or e_{kj}) as a candidate for the next vertex insertion.

6. **Binary tree hierarchy property.** Each insertion is followed by exactly two recursive calls (which may result in subsequent insertions); and the algorithm itself does not specify how to order those two recursive calls. The insertion order is, thus, a partial ordering that behaves like a (directed) rooted binary tree. This tree-like behavior has been duly noted and studied in some depth by Cromley (1991). The algorithm's action on any branch of the tree is completely independent of its action on any other non-intersecting branch, since the decomposition of P_{ij} only depends on the vertices between i and j. The partial dependence of insertion order is illustrated in Figure 7. In Figure 7, the end-points of the segment being simplified are shown as two circles, and the new point being inserted at each stage is highlighted with a rectangle. The vertex corresponding to a node in the tree cannot be accepted unless all of its ancestor node vertices have already been accepted.

7. **Monotone threshold dependence property.** Reducing the threshold ε will only add nodes to the tree, not change or remove any nodes that are already there. Increasing the threshold ε will prune entire branches (consisting of a node and all of its children) from the tree. In other words, suppose that $V(\varepsilon_i)$ is the set of vertices selected by the Douglas-Peucker algorithm for a threshold ε_i. Then $\varepsilon_1 < \varepsilon_2$ implies that $V(\varepsilon_2) \subset V(\varepsilon_1)$.

It should be pointed out that the distance of the farthest vertex from the approximating line segment does not necessarily decrease as we descend the tree. For that reason, it may be impossible to increase the vertex set by a single node by lowering the threshold, ε. In the zig-zag example in Figure 8, one either gets only two vertices or all vertices in the approximating line. There are no in-between approximations.

8. **Tree pruning property.** The threshold ε may be increased or decreased at any step in the algorithm (for any particular recursive call), and the vertex selection/line simplification procedure itself will not change fundamentally. This leads to a natural extension to the Douglas-Peucker algorithm: we may choose a different ε for each test situation, and we have a "new" subroutine call, Simplify (P_{ij},ε_{ij}}). Since the identity of the next candidate vertex does not depend on the threshold (only its acceptance or rejection does), we always have the same maximal

underlying tree, as described in property 6. Total flexibility in changing thresholds at any stage allows us either to accept any node or to prune any node (and its entire sub-tree) after we have descended to that node by accepting all of its ancestors.

9. **Convergence property.** One may obtain any desired precision to the original polyline by choosing a small enough ε. In fact, by taking ε small enough, one gets the trace, or identical point set, of the original polyline.[2] This extreme, trivial simplification guarantees that a modified Douglas-Peucker process (for variable thresholds) can *always* be made to converge to a topologically correct structure because, in the extreme case, when we let ε go to zero, cell topology and geometry are fully recaptured. Note that convergence to a finite sub-sequence of a finite sequence of vertices means that the process "stabilizes" or stops after a finite number of refinement steps.

10. **Sufficiency of point conflicts property.** Any linear feature that intersects the simplified polyline (but not the original polyline) must have at least one of its vertices within the convex hull of the original polyline (Zhang 1996). This property requires proof, as demonstrated in Lemma 1. The property is key to reducing our search space for potential topological conflicts. For point-line conflicts we must examine points that lie in certain restricted convex regions; for line-line conflicts we need only examine segment end-points that lie in certain restricted convex regions.

Before we begin proving lemmas and theorems, we highlight one more key property of the Douglas-Peucker algorithm that will help us keep track of inconsistencies.

11. **Triangle inversion property.** When two segments replace one segment in the simplified polyline, the only points that change their sidedness with respect to the old simplified polyline and the new simplified polyline are in the triangle formed by the replaced segment and the two replacing segments (Figure 9). This obvious property helps us keep track of wrong-sided and right-sided point features when we compare the original polyline to two successive simplification stages in the point insertion tree.

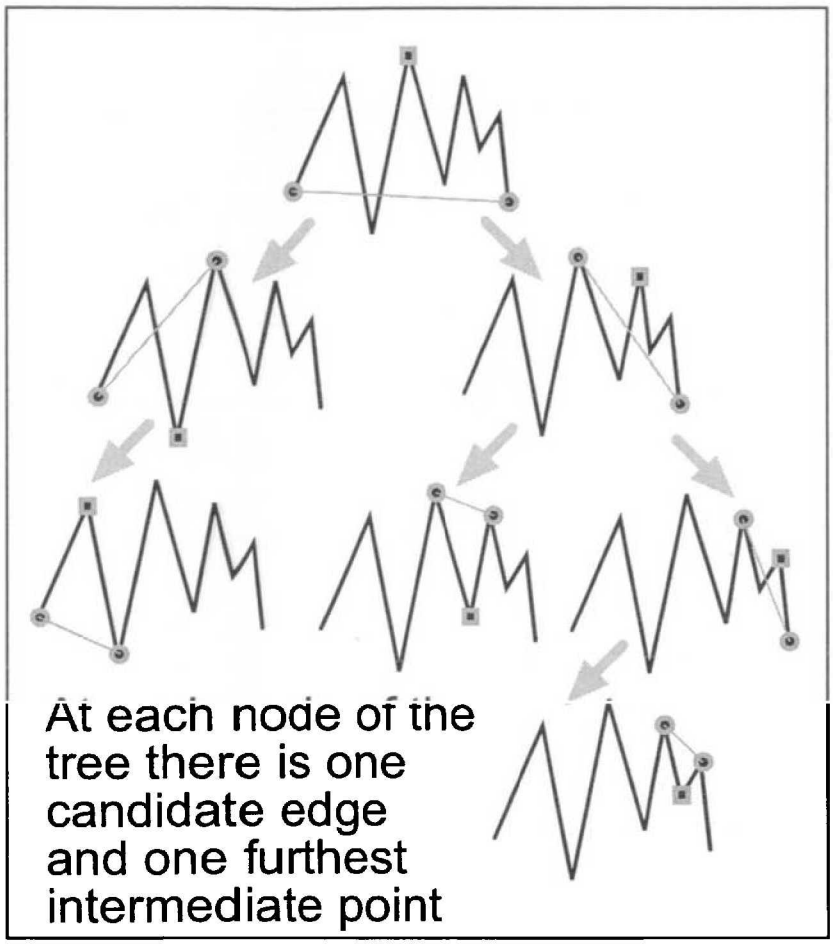

Figure 7. The order in which vertices may be added is the partial order of a binary tree.

Polyline Hulls and Their Properties

Polyline hulls are special cases of hulls of sequentially numbered vertex sets. In particular, the vertices of

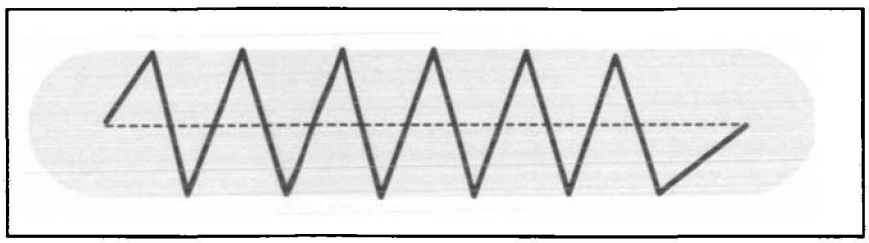

Figure 8. The only fixed ε-threshold polylines are trivial: two points or all points.

[2] If the original polyline has collinear vertices, then intermediate vertices may be missing from the simplified polyline.

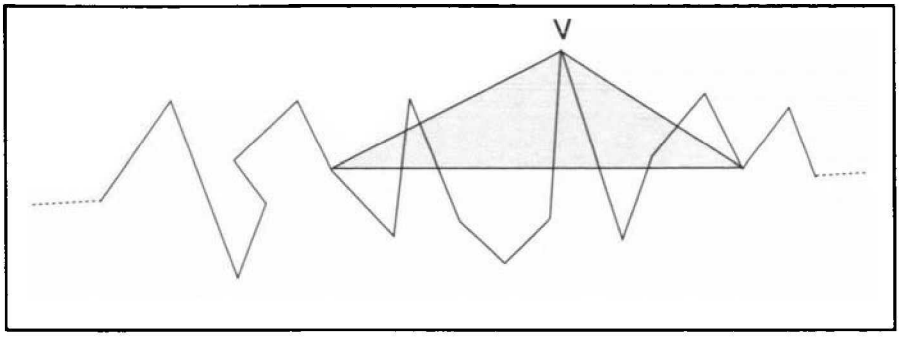

Figure 9. Each new point insertion v re-orients all points in a triangle with respect to the new simplified line.

the polyline that are extreme (i.e., belong to the convex hull) are neatly ordered on the hull (see Lemma 2). Moreover, the polyline, as it moves about inside the hull, cleanly partitions the hull polygon into sub-polygons. Each sub-polygon contains exactly one non-polygon hull edge and a sub-polyline P_{ij}, which is a corollary to the proof of Lemma 1.

Lemma 1 Suppose that P_s is a sub-sampled representation of the polyline P; i.e., P_s consists of a subsequence of the vertex sequence of P. If a line segment L intersects the simplified polyline P_s, but does not intersect the original polyline P, then at least one of the end-points of L lies in the convex hull of P.

Proof: Suppose otherwise, namely, that both end-points of L lie outside the convex hull of P, that L intersects P_s, and L does not intersect P. Since all points of P_s lie inside the convex hull of P, and we are assuming that L intersects P_s, L must enter and leave the convex hull of P. If L enters or leaves the convex hull of P at a vertex of the convex hull, then L touches P as all vertices of the convex hull of P are vertices of P itself. Suppose that L enters the convex hull between vertices v_i and v_j but does not touch either of the two vertices v_i and v_j (Figure 10). If the edge $e_{ij} = [v_i, v_j]$ is not an edge of the polyline P, then the polygon formed by e_{ij} and P_{ij} is the closed polygon that lies entirely within the convex hull of P. Because L enters this polygon and departs the convex hull there must be another point of L at which L leaves the polygon. This departure point must lie on P_{ij} (as it lies on the boundary of the polygon) and cannot lie on e_{ij}.

Corollary 1 The only line segments that might interfere with a simplified polyline and which had not already interfered with the original polyline have at least one of their endpoints in the convex hull of the original polyline.

This corollary permits a search strategy that (1) conveniently limits the search space to nested sets of convex regions, and (2) examines only line end-points, not the lines themselves. The decreasing sequence of search spaces can be built and examined quickly by using a version of the convex hull update tool modified by Hershberger and Snoeyink (1992) on the basis of Melkman (1987) and Lemma 2 which follows.

Lemma 2 Suppose that the vertices of the polyline P_{0n} are labeled in order with sub-indices 0, 1, 2, ..., n (Figure 11). Suppose that the vertices of the polyline P_{0n} that lie on the convex hull are labeled in clockwise order with sub-indices $i_0, i_1, i_2, \ldots i_k, i_{k+1}, \ldots, i_j = i_0$, where i_0 is the smallest index appearing on the hull and i_k is the largest index appearing on the hull. Then the clockwise sequence of indices on the hull is increasing from i_0 to i_k and decreasing from i_k to i_o thus: $i_0 < i_1 < i_2 < \ldots < i_k$, and $i_k > i_{k+1} > \ldots > i_j = i_0$.

Proof: Given any polyline, label the vertices of the polyline P_{0n} in order with sub-indices 0, 1, 2, ..., n as in Figure 11. Consider only the vertices of the polyline P_{0n} that lie on the convex hull as in Figure 12.

Then we must have that either hull path (clockwise or counterclockwise) from the smallest numbered hull vertex ("0" in Figure 12) to the largest

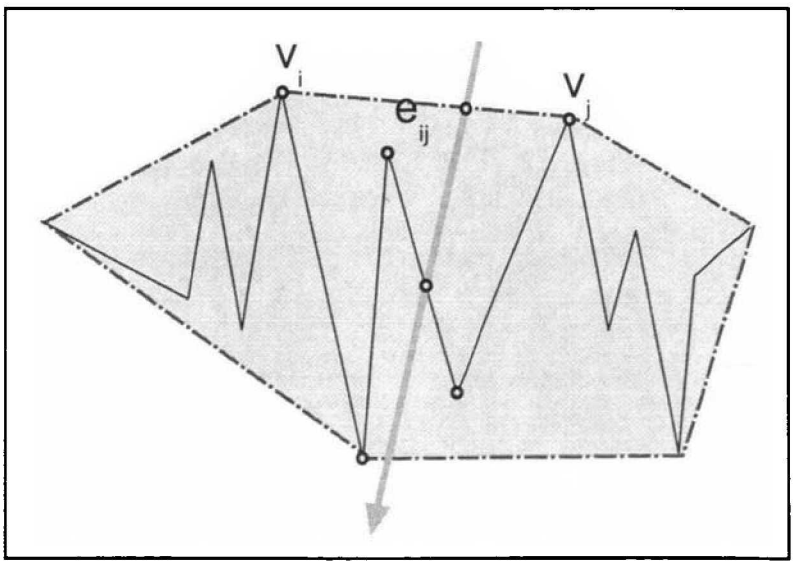

Figure 10. A line traversing the convex hull must cross the polyline.

numbered hull vertex ("21" in Figure 12) traverses an increasing sequence of indices. Suppose otherwise, namely, that the polyline doubles back after contacting the convex hull. Then the polyline would "re-enter" a region completely bounded by a hull edge and the previous subsequence of the polyline's edges, as shown in the shaded region of Figure 13. Such a polyline would be trapped in the shaded region as it could not cross the region boundary without either crossing itself or venturing outside its own convex hull.

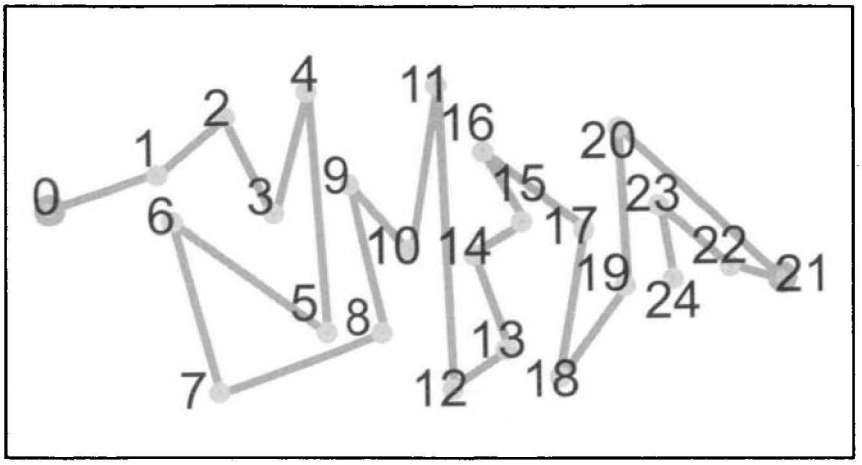

Figure 11. Polyline vertices are labeled "0" through "24".

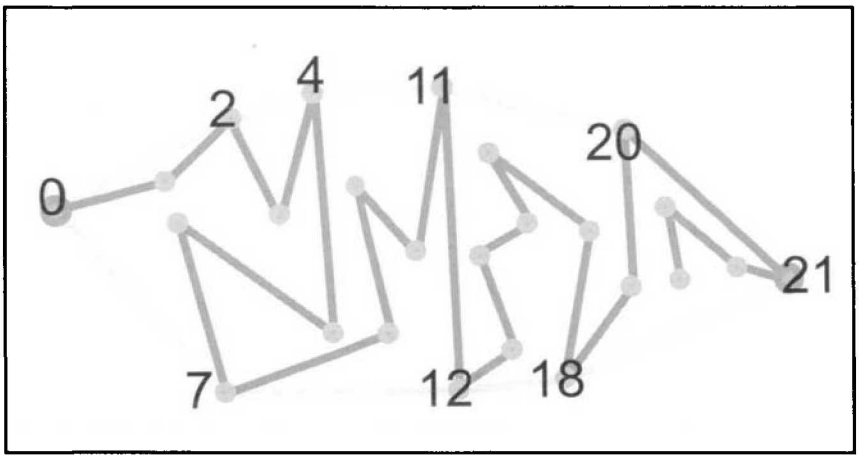

Figure 12. Only polyline vertices on the hull are shown labeled.

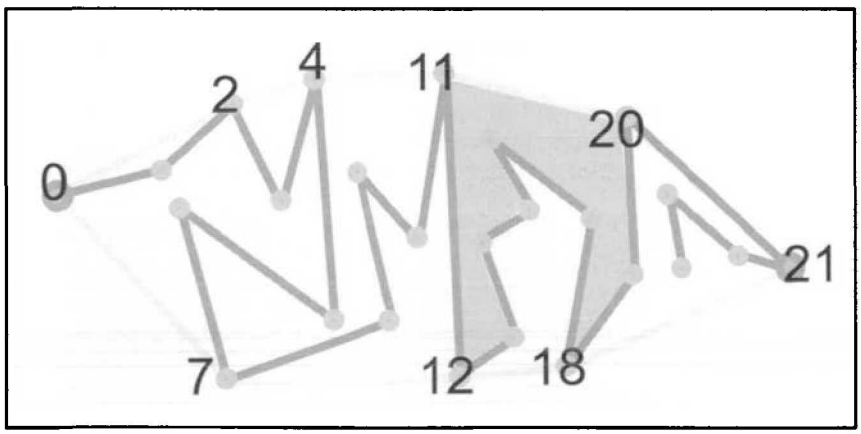

Figure 13. If polyline changes direction at vertex "20," then it enters the shaded region and produces no further hull points.

Efficient Implementation with Hulls

Hershberger and Snoeyink (1992) designed an algorithm to compute the Douglas-Peucker polyline simplification with worst-case $O(n \log n)$ time complexity, improving on the traditional straightforward implementation complexity of $O(n^2)$. To obtain a speed-up, they noticed that the farthest vertex v_k from a shortcut line segment e_{ij} always belongs to the convex hull of intermediate vertices $\{v_{i+1}, v_{i+2}, \ldots, v_{j-2}, v_{j-1}\}$. They modified Melkman's (1987) deque data structure for convex hulls of polylines to achieve rapid reconstitution of the hulls of the two sub-polylines arising from recursive descent.

Each vertex selection v_k causes us to trim the convex hull of P_{ij} in question into two possibly overlapping sub-hulls, as shown in Figure 14. Each vertex in the original hull belongs to exactly one of the two sub-hulls. Moreover, the two sub-hulls are the hulls of the sub-polylines P_{ik} and P_{kj}. We will be able to determine very easily if the two hulls can cause any topological inconsistency whatsoever by checking the directions of two critical segments.

With each point v_k chosen from the vertices in P_{ij}, the new vertices that are added to each sub-hull have the following property:

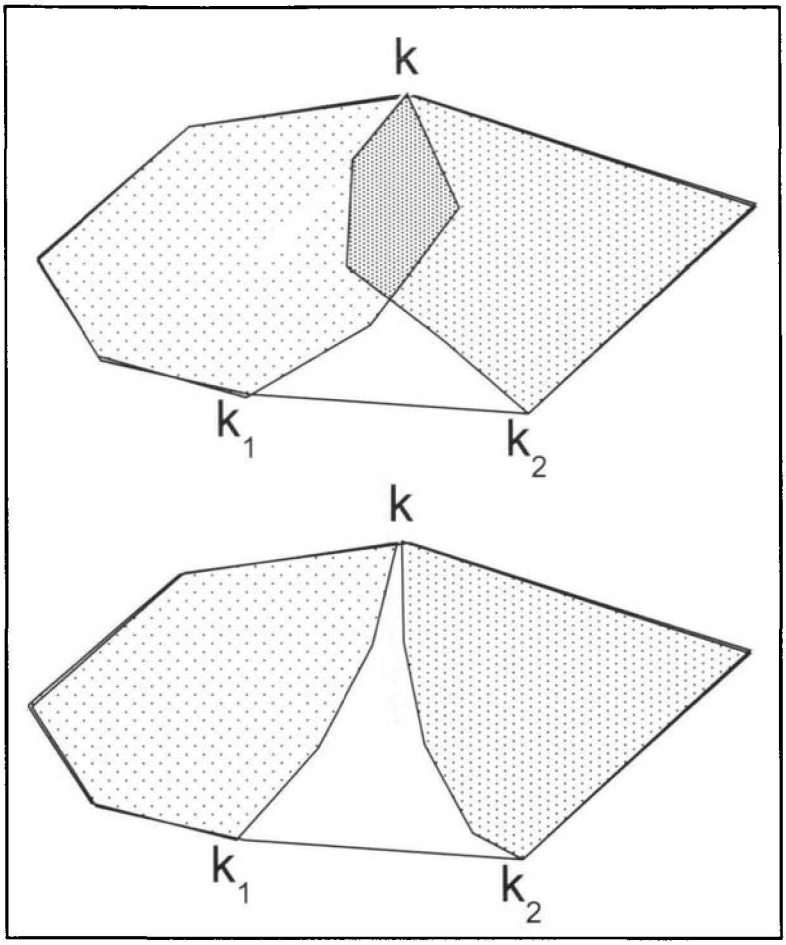

Figure 14. Interlinked hulls must have a vertex in the closed intersection.

Theorem 1 If v_k is in the upper hull of P_{ij}, then let v_{k1} be the vertex in the lower hull with the largest index less than k; and let v_{k2} be the vertex in the lower hull with the smallest index greater than k. Then the hull corresponding to P_{ik} consists of all P_{ij} hull vertices with an index less than or equal to k plus all lower hull vertices of P_{k1k}. Moreover, the hull corresponding to P_{kj} consists of all P_{ij} hull vertices with an index greater than or equal to k plus all lower hull vertices of P_{kk2}.

If, on the other hand, v_k is in the lower hull of P_{ij}, then let v_{k1} be the vertex in the upper hull with the largest index less than k; and let v_{k2} be the vertex in the upper hull with the smallest index greater than k. Then the hull corresponding to P_{ik} consists of all P_{ij} hull vertices with an index less than or equal to k plus all upper hull vertices of P_{k1k}. Moreover, the hull corresponding to P_{kj} consists of all P_{ij} hull vertices with an index greater than or equal to k plus all upper hull vertices of P_{kk2}.

At every stage in the polyline simplification we have a sub-polyline P_{ij}, the candidate simplifying line segment e_{ij}, and all of the upper (clockwise from v_i to v_j) and lower (counter-clockwise from v_i to v_j) convex hull vertices on the polyline P_{ij} to choose from.

The critical sub-paths of a polyline in its convex hull consist of those paths which link a vertex in the upper hull to the next vertex (in order) that lies in the lower hull (and vice versa). Consider the following: suppose that our vertex v_k (to be inserted to enhance e_{ij}) is in the upper hull of P_{ij}, as in the first part of Theorem 1. As in the theorem, suppose that v_{k1} is the vertex in the lower hull with the largest index less than k, and v_{k2} is the vertex in the lower hull with the smallest index greater than k. Then v_{k1} is just to the left of v_{k2} in the lower hull, and we have:

Corollary 2 The upper hull of P_{ik} consists of all of the vertices of the upper hull of P_{ij} with an index less than or equal to k; and the upper hull of P_{kj} consists of all of the vertices of the upper hull of P_{ij} with an index greater than or equal to k. The lower hull of P_{ik} consists of all of the vertices of the lower hull of P_{ik1} plus all of the vertices of the lower hull of P_{k1k} (the vertex k_1 is counted twice). The lower hull of P_{kj} consists of all of the vertices of the lower hull of P_{kk2} plus all of the vertices of the lower hull of P_{k2j} (the vertex k_2 is counted twice).

A simple geometric argument utilizing Figure 14 gives us the following result about consecutive polyline sub-intervals interfering with each other.

Corollary 3 Suppose that vertex v_k is in the upper hull of P_{ij}. Then either the hull of P_{ik} contains the vertices of P_{kj} or, the hull of P_{kj} contains the vertices of P_{ik} if and only if the lower hull of P_{kk2} starts off to the left of the lower hull of P_{k1k}.

Handling Topological Conflicts

Left- or right-sidedness may be altered when a polyline is replaced by a simplified version, as illustrated in Figure 15.

An offending point must be trapped between a simplifying segment and the polyline that it represents. Points that lie to the left of the original directed polyline may lie to the right of the simplifying edge and vice versa. If the original polyline is a river feature, for example, then the misplaced point features will wind up on the wrong riverbank in the simplified version. One may note that all of the

potential conflicts lie in the convex hulls around the set of the simplifying segments. Not all points in those convex hulls are inverted, however. Moreover, as we continue to refine (add vertices to) our simplified line, we see that some points that were previously inverted no longer are inverted (see point P in Figure 15); and some points that were not previously inverted become inverted (see point Q in Figure 15).

Finally, some points keep their relative situation with refinement, as is the case of points R and S in Figure 15 (R was inverted and remains inverted after the replacement of L_1 by L_2, and S was not inverted and continues to be not inverted). It is clear, however, from the convergence property of refinement that it is always possible to add enough vertices to the simplified polyline to remove all topological misplacements of finitely many point features with respect to the polyline. Our strategy will therefore be:

- We apply first the classic Douglas-Peucker algorithm for a reasonable, fixed threshold ε;
- Next we detect individual edges that represent sub-polylines that may require additional refinement (beyond the initial application of the Douglas-Peucker algorithm); and finally
- We systematically add vertices to the simplified polyline to correct the topological arrangement of detected mis-positioned points and to detect any newly engendered topological inconsistencies that arise during our refinement procedures.

At first glance, it may appear that our procedure "fixes" some mis-positioning at the expense of introducing new sidedness conflicts. Indeed, our procedure may "fix" the sidedness of a mispositioned point in one step, then "undo" the corrective measure in the next step. It may seem hard to keep tabs of which points are on the "right side of the tracks" when we attempt to correct matters by moving the tracks. We manage this seemingly complex interaction not by organizing points according to parity groups (i.e., those that are right-sided and those that are wrong-sided) at any stage. Instead, we focus on those that change parity at any stage, and our triangle inversion property gives us the following result:

Lemma 3 The point features which change parity (go from wrong-sidedness to right-sidedness, or go from right-sidedness to wrong-sidedness) when the simplifying line segment L_1 is replaced in the simplifying polyline by the two-segment piece L_2 are precisely those point features that lie inside the triangle bounded by L_1 and L_2.

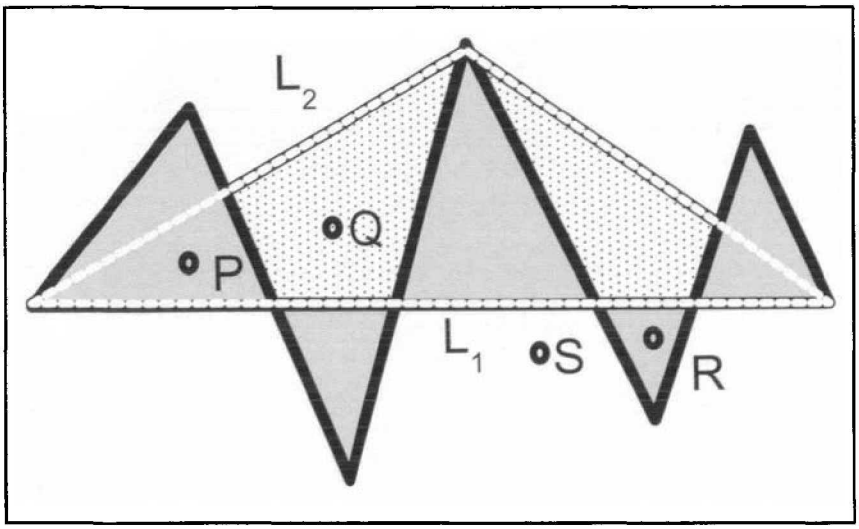

Figure 15. The line L_1 inverts five regions, the polyline L_2 inverts four regions.

Proof: Features inside the triangle go from being above L_1 to being below L_2 in the orientation of Figure 15. The relationship of those features to the original polyline does not change; so, within the triangle, the correctness or incorrectness of the approximating line is inverted when L_2 replaces L_1. By the same token, outside the triangle the relative position to either simplifying line is the same—above or below—and the correctness or incorrectness of either approximating line is the same.

With this observation, we may do the following:

1. Identify all *external* point features (i.e., not vertices of the sub-polyline P_{ij} itself) that are in the convex hull of the sub-polyline P_{ij} that is being approximated by the edge e_{ij}.

2. Determine the parity (correct(+) or incorrect(-)) for each of the features of the edge approximation e_{ij} with respect to the sub-polyline P_{ij}.

3. If any of the features have parity negative, add the farthest point v_k from e_{ij} from among the sub-polyline vertices. (This is just a continuation of the familiar Douglas-Peucker procedure—without the ε-test.)

4. Find all point features within the triangle $\Delta v_i v_j v_k$, and change their parity.

5. Continue to further subdivide e_{ik} and/or e_{kj} if any of the following occur:

(a) Farthest sub-polyline points to the approximating segment are at a distance greater than ε from the segment.

(b) There are still external point features in the two new sub-hulls with negative parity.

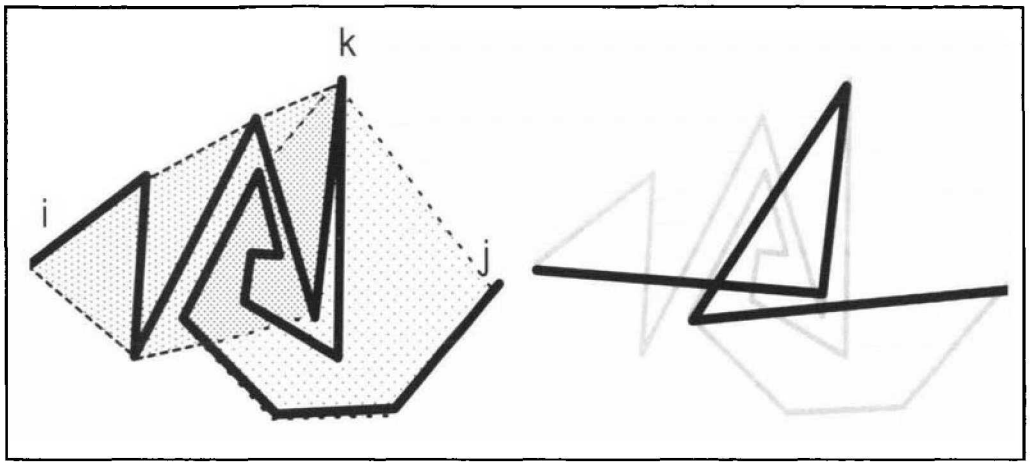

Figure 16. The subpolylines P_{ik} and P_{kj} have intersecting hulls.

(c) There are "newly external" point features with negative parity in one or both of the sub-hulls. (The vertices of the sub-polyline P_{kj} are external to the sub-polyline P_{ik} and vice versa. If the sub-polyline P_{kj} snakes back along the sub-polyline P_{ik} as in Figure 16, then extra vertices will almost certainly be needed to achieve topological/geometric consistency for this complex nesting).

We may use any spatial organizational tool such as quad-trees to select all points within a reasonably small neighborhood of each convex hull. Note that the width of each elementary convex hull corresponding to a sub-polyline approximated by a single edge in the traditional Douglas-Peucker algorithm is less than or equal to 2ε.

We can further test all points in this manageable subset for actually "belonging to the convex region" with a quick logarithmic test involving binary search of hull vertices for each point tested).

We then compute the parity of each test point using an "inside-outside" test. We project[3] a ray from the test point in any direction and count the number of crossings by both the polyline P_{ij} and the approximating edge e_{ij}. If the sum is even (including zero), then the sidedness is correct, and we say that the parity is positive.

Note that we compute parity only once. After that initial computation we merely toggle the parity of those points that are found to lie in the triangle we build at each insertion stage. We have seen that when a polyline is subdivided, the two halves of the polyline may interfere with each other, causing us to seek and find new potentially conflicting points. We know, however, that such potentially conflicting points will exist precisely when the "right lower hull" of the new splitting point starts off to the left of the "left lower hull" of that same point, as is the case in the top drawing in Figure 14 and in the left side of Figure 16.

Conclusions

What we have proposed is simply to keep adding the next "farthest" vertex, Douglas-Peucker style, even after the usual Douglas-Peucker ε-closeness is attained, provided that the resulting polyline fails to pass topological muster. The next candidate point to add for any given segment e_{ij} approximating sub-polyline P_{ij} is uniquely determined (up to ties). We have seen that we can readily test to see if adding the point (and possibly others to follow) is necessary to correct a sidedness flaw resulting from having segment e_{ij} approximate sub-polyline P_{ij}. We have argued that adding points to correct sidedness flaws will always terminate successfully (even though we may wind up adding all the original points to achieve this success.) We have provided a simple rule for quickly updating sidedness of all critical points lying in the convex hull of the sub-polyline (we toggle parity for all points in a triangle and leave the other points unchanged). Finally, we proved that it is enough to check for points on the wrong side of the approximating segment to detect failed topological criteria for all point and line features. Our final simple test detects all instances of consecutive polyline sub-intervals interfering with each other. For those sub-polylines to interweave, we must have that the lower sub-hull of one sub-polyline "invades" the sub-hull of the other, and vice versa. A simple direction check on the two initial lower hull segments either shows interaction or rules it out.

[3] Projecting "up" in the vertical direction is easily accomplished by counting the number of segments of the polyline that straddle the test point's x value and have a larger y value than the test point.

REFERENCES

de Berg, M., and M. Kreveld. 1995. A new approach to subdivision simplification. In: *AutoCarto 12*, Charlotte, North Carolina, February 27-March 2, 1995. pp. 79-88.

Cromley, R. 1991. Hierarchical methods for line simplification. *Cartography and GIS* 18(2): 125-31.

Douglas, D., and T. Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer* 10: 112-22,

Hershberger, J., and J. Snoeyink. 1992. Speeding up the Douglas-Peucker line simplification algorithm. In: *Proceedings of the 5th International Symposium on Spatial Data Handling*, Charleston, South Carolina, August 1992. pp. 134-43.

Jones, C., G. Bundy, and J. Ware. 1995. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems* 22(4): 317-31.

McMaster, R., and K. S. Shea. 1992. *Generalization in digital cartography*. Association of American Geographers, Washington, DC.

Melkman, A. A. 1987. On-line construction of the convex Hull of a simple polyline. *Information Processing Letters* 25: 11-2.

Ramer, U. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* 1: 244-56.

Zhang, Y-Y. 1996. A modified Douglas-Peucker simplification algorithm. Unpublished Master's Degree thesis, the Ohio State University, Columbus, Ohio. 70 p. ▪