A New Approach to Subdivision Simplification*

Mark de Berg
Dept. of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands
markdb@cs.ruu.nl

Marc van Kreveld
Dept. of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands
marc@cs.ruu.nl

Stefan Schirra
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
Germany
stschirr@mpi-sb.mpg.de

Abstract

The line simplification problem is an old and well-studied problem in cartography. Although there are several algorithms to compute a simplification, there seem to be no algorithms that perform line simplification in the context of other geographical objects. This paper presents a nearly quadratic time algorithm for the following line simplification problem: Given a polygonal line, a set of extra points, and a real $\epsilon > 0$, compute a simplification that guarantees (i) a maximum error ϵ, (ii) that the extra points remain on the same side of the simplified chain as of the original chain, and (iii) that the simplified chain has no self-intersections. The algorithm is applied as the main subroutine for subdivision simplification.

1 Introduction

The line simplification problem is a well-studied problem in various disciplines including geographic information systems [Buttenfield '85, Cromley '88, Douglas & Peucker '73, Hershberger & Snoeyink '92, Li & Openshaw '92, McMaster '87], digital image analysis [Asano & Katoh '93, Hobby '93, Kurozumi & Davis '82], and computational geometry [Chan & Chin '92, Eu & Toussaint '94, Guibas et al. '93, Imai & Iri '88, Melkman & O'Rourke '88]. Often the input is a polygonal chain and a maximum allowed error ϵ, and methods are described to obtain another polygonal chain with fewer vertices that lies at distance at most ϵ from the original polygonal chain. Some methods yield chains of which all vertices are also vertices of the input chain, other methods yield chains where other points can be vertices as well. Another source of variation on the basic problem is the error measure that is used. Well known criteria are the parallel strip error criterion, Hausdorff distance, Fréchet distance, areal displacement, and vector displacement.

*This research is supported by ESPRIT Basic Research Action 7141 (project ALCOM II: *Algorithms and Complexity*), and by a PIONIER project of the Dutch Organization for Scientific Research N.W.O.

Besides geometric error criteria, in geographic information systems one can also use criteria based on the geographic knowledge, or on perception [Mark '89].

The motivation for studying line simplification problems is twofold. Firstly, polygonal lines at a high level of detail consume a lot of storage space. In many situations a high level of detail is unnecessary or even unwanted. Secondly, when objects are described at a high level of detail, operations performed on them tend to be slow. An example where this problem can be severe is in animation.

Our motivation for studying the line simplification problem stems from reducing the storage space needed to represent a map in a geographic information system. We assume the map is modelled as a subdivision of the plane or a rectangular region thereof. In this application the main consideration is the reduction of the complexity of the subdivision. The processing time may be a little higher, but within reason. The description size of the subdivision is a permanent cost in a geographic information system, whereas the processing time is spent only once in many applications.

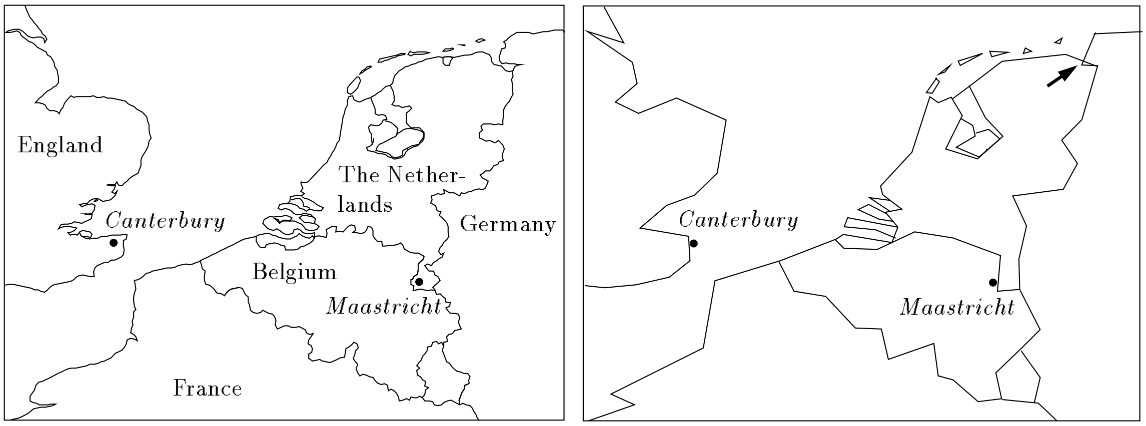

Figure 1: Part of a map of Western Europe, and an inconsistent simplification of the subdivision.

One of the most important requirements of subdivisions for maps is that they be simple. No two edges of the subdivision may intersect, except at the endpoints. This poses two extra conditions on the line simplification method. Firstly, when a polygonal chain is reduced in complexity, the output polygonal chain must be a simple polygonal chain. Several of the line simplification methods described before don't satisfy this constraint [Chan & Chin '92, Cromley '88, Douglas & Peucker '73, Eu & Toussaint '94, Hershberger & Snoeyink '92, Imai & Iri '88, Li & Openshaw '92, Melkman & O'Rourke '88]. The second condition that need be satisfied is that the output chain does not intersect any other polygonal chain in the subdivision. In other words, the simplification method must respect the fact that the polygonal chain to be simplified has a context. Usually the context is more than just the other chains in the subdivision. On a map with borders of countries and cities, represented by polygonal chains and points, a simplification method that does not respect the points can result in a simplified map in which cities close to the border lie in the wrong country. In Figure 1, Maastricht has moved from the Netherlands to Belgium, Canterbury has moved into the sea, and near the top of the border between The Netherlands and Germany, two coast lines intersect. Such topological errors in the simplification lead to inconsistencies in geographic information systems.

In this paper we will show that both conditions can be enforced after reformulating the problem into an abstract geometric setting. This is quite different from the approach reported in [Zhan & Mark '93], who have done a cognitive study on conflict resolution due to simplification. They accept that the simplification process may lead to conflicts (such as topological errors) and try to patch up the problems afterwards. We avoid conflicts from the start by using geometric algorithms. These algorithms are fairly easy to implement, and we give the necessary pseudo code.

The remainder of this paper is organized as follows. Section 2 discusses our approach to the subdivision simplification, and identifies the main subtask: a new version of line simplification. Section 3 describes the approach of Imai and Iri for the standard line simplification problem. In Section 4 we adapt the algorithm for the new version of line simplification. Section 5 summarizes the subdivision simplification algorithm, and Section 6 gives a number of practical approaches to improve—both in speed and in output quality—the basic algorithm given before. In Section 7 the conclusions are given.

2 Subdivision simplification

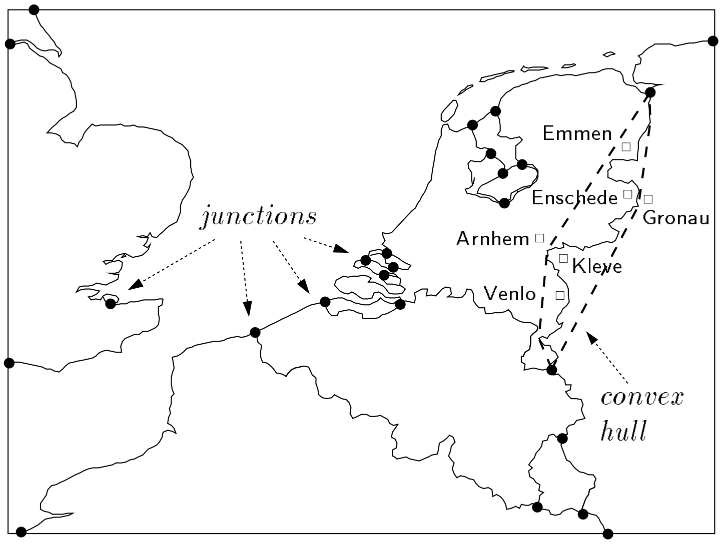

Figure 2: A subdivision with its junctions indicated.

Let S be a subdivision that models a chloropleth map. A subdivision is a geometric structure that represents an embedded planar graph. We adopt the terminology standard in (computational) geometry and say that the subdivision S consists of vertices, edges and faces. The degree of a vertex is the number of edges incident to it. A vertex of degree one is a *leaf,* a vertex of degree two is an *interior vertex,* and a vertex of degree at least three is a *junction.* See Figure 2. Generally the number of leafs and junctions is small compared to the number of interior vertices. Any sequence of vertices and edges starting and ending at a leaf or junction, and with only interior vertices in between, is called a *polygonal chain,* or simply a *chain.* For convenience we also consider a cycle of interior vertices (which occur for islands) as a chain, where we choose one of the vertices as the junction. It is the start and the end vertex of the chain.

Let P be a set of points that model special positions inside the regions of the map. Subdivision simplification can now be performed as follows. Keep the positions of all leafs and junctions fixed, and also the positions of the points in P. Replace every chain between a start and end vertex by a new chain with the same start and end vertex but with fewer interior vertices. For a polygonal chain C, we require of its simplification C':

1. No point on the chain C has distance more than a prespecified error tolerance to its simplification C'.

2. The simplification C' is a chain with no self-intersections.
3. The simplification C' may not intersect other chains of the subdivision.
4. Each point of P lies to the same side of C' as of C.

Let's take a closer look at the last requirement. The chain C is part of a subdivision that, generally, separates two faces of the subdivision. In those two faces there may be points of P. The simplified chain between the start vertex and the end vertex will also separate two faces of the subdivision, but these faces have a slightly different shape. The fourth requirement states that the simplified chain C' must have the same subsets of points in those two faces. For chains that have the same face to both sides we cannot make such an observation. Any simplification will leave the points of P in the same face of the subdivision.

The first requirement will be enforced by using and extending a known algorithm that guarantees a maximum error ϵ. The other three requirements are enforced by the way we extend the known algorithm. Intuitively, the simplified chain consists of a sequence of edges that bypass zero or more vertices of the input chain. We will develop efficient tests to determine whether edges in the simplified chain leave points of P to the wrong side or not.

We'll see that enforcing the third requirement doesn't add much to the difficulty of the algorithm. When applying the simplification algorithm to some chain of the subdivision, we temporarily add to the set P of points all vertices of other chains of the subdivision. One can show that—since C' has the vertices of other chains to the same side as C—the simplified chain C' won't intersect any other chain of the subdivision. We will apply a similar idea to avoid self-intersections. A simplified chain that has the points of P to the correct side, has no self-intersections, and doesn't intersect other chains in the subdivision is a *consistent simplification.*

A disadvantage of adding the vertices to the point set P is that P can become quite large, which will slow down the algorithm. There are two observations that can help reduce the number of points that need be added to P. Firstly, we only have to take the vertices of the chains that bound one of the two faces separated by the chain we are simplifying. Secondly, it is easy to show that only points inside the convex hull of the chain that is being simplified could possibly end up to the wrong side. So we only have to use points of P and vertices of other chains that lie inside this convex hull. In Figure 2, the chain that represents the border between the Netherlands and Germany is shown with its convex hull (dashed) and some cities close to the border (squares). No other chains intersect the convex hull, and only the cities Emmen, Enschede, Kleve and Venlo must be considered when simplyfing the chain (the pseudo code for this idea follows later in this paper).

It remains to solve a new version of the line simplification problem. Namely, one where there are extra points which must be to the same side of the original chain and the simplified chain. For this problem we will develop an efficient algorithm in the following sections. It takes $O(n(n+m)\log n)$ time in the worst case for a polygonal chain with n vertices and m extra points. This will lead to:

Theorem 1 *Given a planar subdivision S with N vertices and M extra points, and a maximum allowed error $\epsilon > 0$, a simplification of S that satisfies the four requirements stated above can be computed in $O(N(N+M)\log N)$ time in the worst case.*

The close to quadratic time behavior of the algorithm may seem too inefficient for subdivisions with millions of vertices. However, one can expect that the quadratic time behavior in the worst case won't show up in practice. It will depend on the description sizes of the chains in the subdivision, the number of extra points and their positions, and the shapes of the chains themselves. An implementation and test runs are required to examine the running time on real data.

Theoretically, it would be satisfactory to compute efficiently a minimum size simplification of the subdivision S that satisfies the given constraints. (Throughout this paper, the *size* of a chain refers to the number of edges, not to the length.) Unfortunately, this seems to be very difficult. It should be noted that some other version of the subdivision simplification problem, where the objective is to obtain a minimum complexity simple subdivision, is an NP-hard problem [Guibas et al. '93]. We do, however, guarantee a minimum size simplification for the simplification of every polygonal line that satisfies a condition weaker than monotonicity.

3 Preliminaries on line simplification

We describe the line simplification algorithm in [Imai & Iri '88], upon which our method is based. Let $v_1, \ldots, v_n$ be the input polygonal chain C. A line segment $\overline{v_i v_j}$ is a *shortcut* for the subchain $v_i, \ldots, v_j$. The *error* of a shortcut $\overline{v_i v_j}$ is the maximum distance from $\overline{v_i v_j}$ to a point v_k, where $i \leq k \leq j$. A shortcut is *allowed* if and only if the error it induces is at most some prespecified positive real value ϵ. We wish to replace C by a chain consisting of allowed shortcuts. This chain should have as few shortcuts as possible. In this paper we don't consider simplifications that use vertices other than those of the input chain.

Let G be a directed acyclic graph with node set $V = \{v_1, \ldots, v_n\}$. The arc set A contains (v_i, v_j) if and only if $i < j$ and the shortcut $\overline{v_i v_j}$ is allowed. The error of an arc (v_i, v_j) is defined as the error of the shortcut $\overline{v_i v_j}$. So $A = \{(v_i, v_j) \mid i < j \text{ and the error of } \overline{v_i v_j} \leq \epsilon\}$. The graph G can be constructed with a trivial algorithm in $O(n^3)$ time and G has size $O(n^2)$.

A shortest path from v_1 to v_n in G corresponds to a minimum size simplification of the polygonal chain. Using topological sorting, the shortest path can be computed in time linear in the number of nodes and arcs of G [Cormen et al. '90]. Therefore, after the construction of G, the problem can be solved in $O(n^2)$ time. We remark that the approach can always terminate with a valid output, because the original polygonal line is always a valid output (though hardly a simplification). The bottleneck in the efficiency is the construction of the graph G. In [Melkman & O'Rourke '88] it was shown that G can be computed in $O(n^2 \log n)$ time, reducing the overall time bound to $O(n^2 \log n)$ time. In [Chan & Chin '92] an algorithm was given to construct G in $O(n^2)$ time. This is optimal in the worst case because G can have $\Theta(n^2)$ arcs. We explain their algorithm briefly.

One simple but useful observation is that the error of a shortcut $\overline{v_i v_j}$ is the maximum of the errors of the *half-line* starting at v_i and containing v_j, and the *half-line* starting at v_j and containing v_i. Denote these half-lines by l_{ij} and l_{ji}, respectively. We construct a graph G_1 that contains an arc (v_i, v_j) if and only if the error of l_{ij} is at most ϵ, and a graph G_2 that contains an arc (v_i, v_j) if and only if the error of l_{ji} is at most ϵ. To obtain the graph G, we let (v_i, v_j) be an arc of G if and only if (v_i, v_j) is an arc in both G_1 and G_2. The problem that remains is the construction of G_1 and G_2 which boils down to determining whether the errors of the half-lines is at most ϵ or not. We only describe the case of half-lines l_{ij} for all $1 \leq i < j \leq n$; the other case is completely analogous.

The algorithm starts by letting the vertices $v_1, \ldots, v_n$ in turn be v_i. Given v_i, the errors of all half-lines l_{ij} with $j > i$ are determined in the order $l_{i(i+1)}, l_{i(i+2)}, \ldots, l_{in}$ as follows. If we associate with v_k a closed disk D_k centered at v_k and with radius ϵ, then the error of l_{ij} is at most ϵ if and only if l_{ij} intersects all disks D_k with $i \leq k \leq j$. Let l'_i be the half-line rooted

Algorithm *Compute-Allowed-Shortcuts*(C, ϵ)
Input: A polygonal chain C with n vertices $v_1, \ldots, v_n$ and a real $\epsilon > 0$.
Output: The set of all allowed shortcuts of C.
(∗ Half-line l_{ij} and disk D_j are defined as above. ∗)

- **for** $i \leftarrow 1$ **to** $n-1$
  - **do** $I \leftarrow (-\pi, \pi]$
    - $j \leftarrow i+1$
    - **while** $a \leq b$ and $j \leq n$
      - **do if** angle of l_{ij} between a and b
        - **then** accept shortcut $\overline{v_i v_j}$
      - **if** $v_i \notin D_j$
        - **then** $I \leftarrow I \cap$ angles of half-lines that intersect D_j
      - $j \leftarrow j+1$

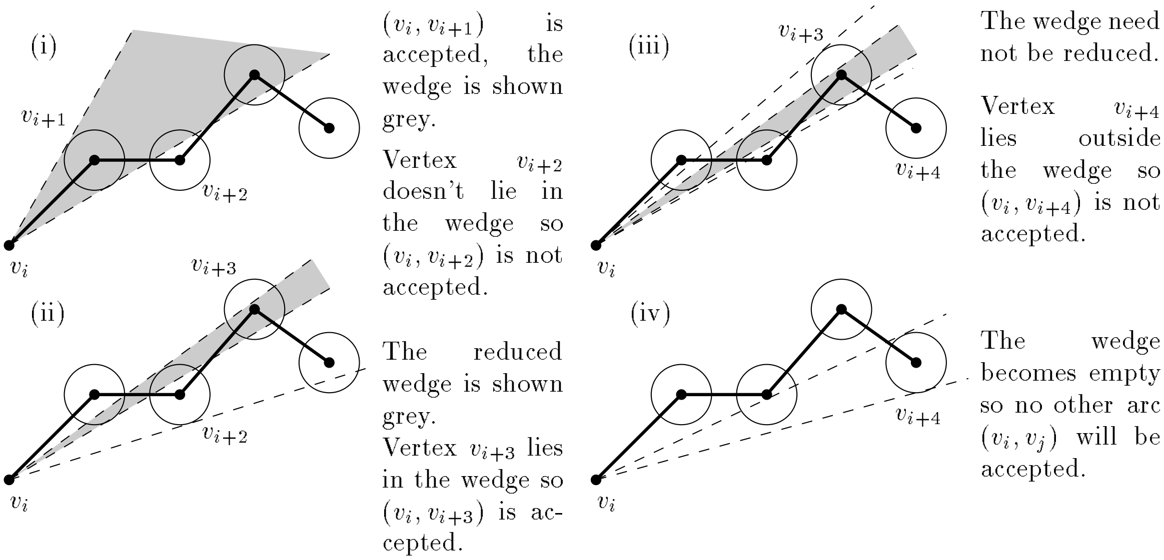

Figure 3: Deciding which arcs (v_i, v_j) with $j > i$ are accepted to G_1. Only (v_i, v_{i+1}) and (v_i, v_{i+3}) will be accepted.

at v_i and extending in positive x-direction. The algorithm maintains the set of angles with l'_i of half-lines starting at v_i that intersect the disks $D_{i+1}, D_{i+2}, \ldots$ incrementally. Initially, the set contains all angles $(-\pi, \pi]$. The set of angles will always be one interval, that is, the set of half-lines with error at most ϵ up to some vertex form a wedge with v_i as the apex. Updating the wedge takes only constant time when we take the next v_j, and the algorithm may stop the inner iteration once the wedge becomes empty. Pseudo code is given as Algorithm *Compute-Allowed-Shortcuts*, and the algorithm is illustrated in Figure 3. With the given approach, the graph construction requires $O(n^2)$ time in the worst case [Chan & Chin '92].

4 Consistent simplification of a chain

In this section we generalize the line simplification algorithm just described to respect extra points as well. We also consider the issue of computing a simplification that has no self-intersections. A polygonal chain or polygon that has no self-intersections is called *simple*.

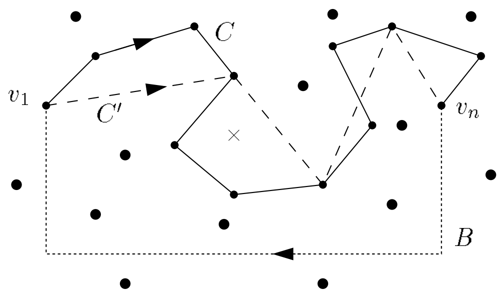

Figure 4: Two chains C and C' which are consistent with respect to the points shown as disks, but not with respect to the point shown as a cross.

Let C and C' be two simple polygonal chains between v_1 and v_n, and let P be a set of points. C is part of the boundary of a face of a subdivision, and the requirement is that the subset of points of P that lie in that face with C as the boundary be the same as the subset of points of P in the face when C is replaced by C'. Therefore, we define two polygonal chains C and C' oriented from v_1 to v_n to be *consistent* with respect to P if there exists a simple chain B oriented from v_n to v_1 that closes both C and C' to simple polygons which have the same subset of points of P in the interior. Also, these simple polygons must have all edges oriented clockwise or counterclockwise. Figure 4 gives an example of two chains C and C' that are closed to simple polygons with clockwise orientation. One can show that any chain B that completes C and C' to simple polygons with the same orientation will give the same result as to consistency of C and C'.

The general approach we take is to compute a graph G_3 with the vertices of the chain as the node set, and an arc between two nodes if the shortcut of the corresponding vertices is consistent. So we don't consider the error of the shortcuts in G_3. Recall that the algorithm in the previous section already used two graphs G_1 and G_2, and the intersections of their arc sets represented the shortcuts with sufficiently small error. If we also intersect the arc set of G_3, we have the additional property that the resulting arcs are consistent as well. In the remainder of this section we only concentrate on computing consistent shortcuts of C.

A polygonal chain is *x-monotone* if any vertical line intersects it in at most one point. In other words, an x-monotone polygonal chain is a piecewise linear function defined over an interval. It is easy to see that any simplification of an x-monotone polygonal chain is also an x-monotone polygonal chain.

In Subsection 4.1 we discuss the simplification of x-monotone chains. We show how to compute all consistent shortcuts from a vertex v_i, and by applying this algorithm to all vertices of the chain we get all consistent shortcuts. In Subsection 4.2 the possible extensions to arbitrary chains are considered. For arbitrary chains the property that any simplification of it is simple doesn't hold any more. So extra measures must be taken to avoid self-intersections.

4.1 Monotone chains

Let C be an x-monotone polygonal chain with vertices $v_1, \ldots, v_n$. We denote the subchain of C between vertices v_i and v_j by C_{ij}. Let P be a set of m points $p_1, \ldots, p_m$.

Lemma 1 *C' is a consistent simplification of C with respect to P if and only if no point of P lies in a bounded face formed by C and C'.*

Proof: For each point p in P we count the number of intersections of a half-line extending vertically downward from p with the chains C and C'. Since C and C' are both x-monotone, these counts can only be zero or one.

Suppose that C' is a consistent simplification of C. Then there exists a chain B that completes both C and C' to simple polygons—denoted BC and BC'—and $P \cap BC = P \cap BC'$. By the point-in-polygon criterion (a point p lies in a polygon if and only if a vertically downward half-line from that point intersects the boundary of the polygon an odd number of times) it follows that the counts mentioned above must be the same for C and C'. Therefore, no point of P lies in a bounded face formed by C and C'.

On the other hand, assume that no point of P lies in a bounded face formed by C and C'. Since C and C' are x-monotone chains it is easy to see that a chain B exists that completes both C and C' to simple polygons with clockwise orientations. Denote these polygons by BC and BC'. Again by the point-in-polygon test, $P \cap BC = P \cap BC'$ since no point of P lies above C and below C' or vice versa. Therefore, C' is a consistent simplification of C. $\square$

Let Q_{ij} be the not necessarily simple polygon bounded by C_{ij} and the edge $\overline{v_i v_j}$, so Q_{ij} contains $j - i$ edges of C and one more edge $\overline{v_i v_j}$. This last edge may intersect other edges of Q_{ij}. Our algorithm will decide efficiently for all vertices v_j whether the polygon Q_{ij} contains points of P in the bounded faces.

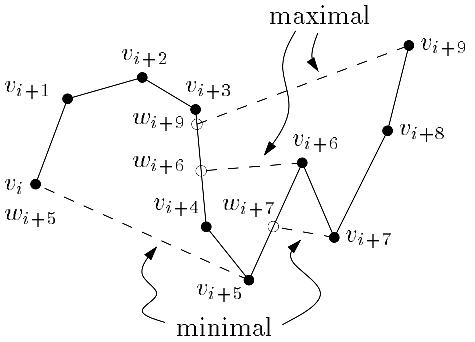

Figure 5: A part of a chain with four tangent splitters.

The first step of the algorithm is to locate all tangent segments from v_i. We define a shortcut $\overline{v_i v_j}$ to be *tangent* if v_{j-1} and v_{j+1} lie in the same closed half-plane bounded by the line through v_i and v_j, and $i+1 < j < n$. The shortcut $\overline{v_i v_n}$ is always considered to be tangent. The tangent shortcuts in Figure 5 are $\overline{v_i v_{i+5}}$, $\overline{v_i v_{i+6}}$, $\overline{v_i, v_{i+7}}$, and $\overline{v_i v_{i+9}}$. A tangent shortcut $\overline{v_i v_j}$ is *minimal (in slope)* if v_{j-1} lies above the line through v_i and v_j. If v_{j-1} lies below that line, then it is *maximal (in slope)*, and if v_{j-1} lies on the line it is *degenerate*. The *tangent splitter* is the line segment $\overline{w_j v_j}$ defined as the maximal closed subsegment of $\overline{v_i v_j}$ that does not intersect C in a point interior to $\overline{w_j v_j}$. So the point w_j is an intersection point of the chain C and the shortcut $\overline{v_i v_j}$, and the one closest to v_j among these, see Figure 5. If v_{j-1} lies on the shortcut $\overline{v_i v_j}$ then $\overline{w_j v_j}$ degenerates to the point v_j. A tangent splitter is minimal, maximal, or degenerate when the tangent shortcut is.

Let $\overline{v_i v_{\gamma(1)}}, \ldots, \overline{v_i v_{\gamma(r)}}$ be the nondegenerate tangents. The corresponding set of tangent splitters and C together define a subdivision S_i of the plane of linear size, see Figure 6. The subdivision has r bounded faces, each of which is bounded by pieces of C and one or more minimal or maximal tangent splitters.

For every face of S_i, consider the vertex with highest index bounding that face. This vertex must define a tangent splitter, so it is one of $v_{\gamma(1)}, \ldots, v_{\gamma(r)}$. Assume it is $v_{\gamma(b)}$. Then we associate with that face the number b. The subdivision and its numbering have some useful properties.

Lemma 2 *Every bounded face of the subdivision S_i is θ-*monotone *with respect to v_i, that is, any half-line rooted at v_i intersects any bounded face of S_i in zero or one connected component.*

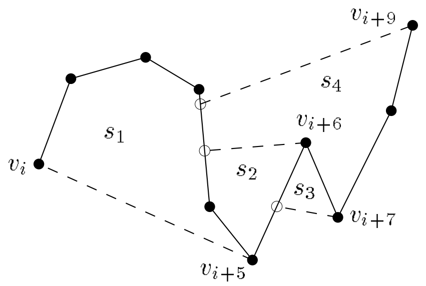

Figure 6: The corresponding subdivision S_i with faces $\gamma(1) = i+5$, $\gamma(2) = i+6$, $\gamma(3) = i+7$, and $\gamma(4) = i+9$.

Proof: Assume that there is a face s_l that is not θ-monotone. Then there must be a vertex v_j in the boundary of that face such that the half-line rooted at v_i intersects the interior of s_l both to the left and to the right of v_j. But then $\overline{v_i v_j}$ is tangent, and hence, the tangent splitter $\overline{v_j w_j}$ is an edge of the subdivision S_i. This contradicts the statement that there is a half-line rooted at v_i that intersects the interior of s_l left and right of v_j. $\square$

Lemma 3 *Every bounded face of the subdivision S_i has one connected subchain of C where half-lines rooted at v_i leave that face.*

Proof: The subdivision S_i only consists of parts of C and tangent splitters, which always extend from a vertex v_j toward v_i. Let s_l be a face of which the part of the boundary where half-lines rooted at v_i leave it consists of more than one subchain of C. Then there must be a vertex v_j between those subchains such that $\overline{v_i v_j}$ is tangent. But then a tangent splitter was chosen extending from v_j to v_i, which makes that the two subchains that were supposed to be in s_l cannot be in the same face, a contradiction. $\square$

Lemma 4 *Any directed half-line from v_i intersects faces in order of increasing number.*

Proof: By the previous lemma, there is a connected subchain of C where the directed half-lines from v_i leave a face. The tangent splitter that gives the number to a face has the vertex with highest index on that chain as one endpoint.

Suppose that there exists a directed half-line from v_i that intersects a face s_l before it intersects s_k, and $l > k$. Consider the unique points t_l and t_k where the half-line leaves s_l and s_k, respectively. Consider the subchains of C of faces s_l and s_k as referred to in the previous paragraph. Since both subchains are connected and disjoint, it follows that t_l lies left of t_k. Since by assumption $l > k$ it follows that C is not x-monotone, a contradiction. $\square$

The tangent splitters can be found in linear time as follows. Traverse C from v_i towards v_n. At every vertex v_j for which $\overline{v_i v_j}$ is tangent (and non-degenerate), walk back along C until we reach v_i or find an intersection of $\overline{v_i v_j}$ with C. In the latter case, the fact that C is x-monotone guarantees that the point we found is the rightmost intersection, and thus it must be w_j. Then we continue the traversal forward at v_j towards v_n. This approach would take quadratic time, but we use the following idea to bring it down to linear. Next time we walk back to compute the next tangent splitter, we use previous tangent splitters to walk back quickly. For a new maximal tangent splitter we only use previously found maximal

Algorithm *Compute-Tangent-Splitters*(C, i)
Input: A chain C with n vertices $v_1, \ldots, v_n$ and an integer i, where $1 \leq i < n$.
Output: The tangent splitters of C with respect to v_i.

```
for j ← i + 1 to n
    do if v_i v_j is maximal tangent
        then k ← j − 1
            while l_ij ∩ v_{k−1} v_k = ∅
                do k ← k − 1
                    if v_i v_k is maximal tangent
                        then k ← h where v_{h−1} v_h contains w_k of maximal tangent splitter
                             w_k v_k
                        else  k ← k − 1
            w_j ← l_ij ∩ v_{k−1} v_k
            Store with v_j a pointer to v_{k−1} v_k
        else  if v_i v_j is minimal tangent
                then "take analogous steps"
                else  j ← j + 1
```

tangent splitters, and for a new minimal tangent splitter we only use minimal ones. One can show that the skipped part of C never contains the other endpoint of the tangent splitter we are looking for. Algorithm *Compute-Tangent-Splitters* contains the pseudo code.

The total number of steps during all backward walks is $O(n)$, which can be seen as follows. During the walks back we visit each vertex which is not incident to a splitter at most twice, once when locating w_j for a maximal tangent splitter $\overline{w_j v_j}$, and once for a minimal tangent splitter. For all following backward walks we bypass vertices that were already visited. Similarly, each tangent splitter is used as quick walk backwards only once; the next time it will be bypassed by another tangent splitter. So we can charge the cost of the backwards walks to the $O(n)$ vertices of C and the $O(n)$ tangent splitters.

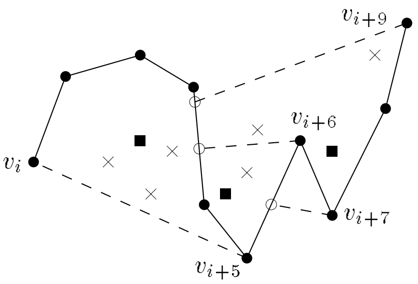

Figure 7: In each face, only the point indicated by a square is maintained.

The second step of the algorithm is to distribute the points of P among the faces of the subdivision S_i. Either by a plane sweep algorithm where a line rotates about v_i, or by preprocessing S_i for point location, this step requires $O((n + m) \log n)$ time [Preparata & Shamos '85]. All points of P that don't lie in a bounded face of S_i can be discarded; they cannot be in a bounded face of the polygon Q_{ij} for any shortcut $\overline{v_i v_j}$. But we can discard many more points. For every face of S_i, consider the tangent splitter with the vertex of highest index. If that tangent splitter is minimal, we discard all points in it except for the point p that maximizes the slope of the directed segment $\overline{v_i p}$, see Figure 7. Similarly, if the tangent splitter with highest index is maximal, we discard all points in the face except for the point p that minimizes the slope of the directed segment $\overline{v_i p}$. Now every face of S_i contains at most one point of P. Algorithm *Distribute-Points* gives the pseudo code.

Lemma 5 *Any shortcut $\overline{v_i v_j}$ is consistent with the subchain C_{ij} with respect to P if and only if it is consistent with respect to the remaining subset of points of P.*

Algorithm *Distribute-Points*(S_i, i, P)
Input: The subdivision S_i in a topological network structure, an integer $1 \leq i < n$, and a set P of points.
Output: The assignment of the points of P to the faces of S_i.
(∗ The plane sweep based method. ∗)
1. $P' \leftarrow$ the subset of P of points that have a larger x-coordinate than v_i.
2. Sort the points of $P' \cup \{v_{i+1}, \ldots, v_n\}$ by angle around v_i and put them in a priority queue Q.
3. Initialize an empty binary search tree T.
4. (∗ A half-line rooted at v_i will sweep from vertically upward clockwise to vertically downward. ∗)
5. (∗ T will store the edges of C beyond v_i intersecting the sweep-line in order of intersection. ∗)
6. **while** Not-Empty(Q)
7.   **do** $p \leftarrow$ Extract-Max(Q)
8.   **if** p is a vertex v_j
9.   **then** update T with the edges incident to v_j
10.   **else** search in T to find the leftmost edge $\overline{v_k v_{k+1}}$ to the right of p and on the sweep-line
11.   **if** this edge exists **then** store p with $\overline{v_k v_{k+1}}$
12. Traverse $\overline{v_i v_{i+1}}, \ldots, \overline{v_{n-1} v_n}$, collect the points stored with the edges, and store one of them (clockwise minimal or maximal) with the appropriate face.

Proof: By definition $\overline{v_i v_j}$ is consistent if and only if no point of P is contained in the closed faces of polygon Q_{ij}. Since every face is θ-monotone, it follows that if a shortcut is inconsistent with respect to some point of P, then it is also inconsistent with respect to the retained point of P in that face. Conversely, if a shortcut is consistent with respect to the retained point in a face, then the shortcut must also be consistent with respect to all other points of P in the face (again because faces are θ-monotone). Since these statements hold for all faces of the subdivision S_i, the lemma follows. □

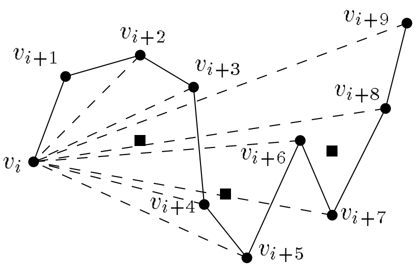

Figure 8: Only the shortcuts $\overline{v_i v_{i+1}}$, $\overline{v_i, v_{i+2}}$, and $\overline{v_i v_{i+3}}$ are accepted.

In the third step of the algorithm we decide which shortcuts $\overline{v_i v_j}$ are consistent and should be present in the graph G_3 in the form of an arc (v_i, v_j). To make the third step efficient we start by sorting the shortcuts $\overline{v_i v_{i+1}}, \ldots, \overline{v_i v_n}$ by slope. Here we consider the shortcuts to be directed away from v_i. Since C is x-monotone, all shortcuts are directed towards the right. The shortcuts are stored in a deque Q (a double-ended queue). We treat the faces of S_i in the order of increasing associated number. When treating a face, we will discard any shortcut $\overline{v_i v_j}$ that has not yet been accepted and is inconsistent with respect to the one remaining point of P in that face (if any). Then we accept those shortcut $\overline{v_i v_j}$ that have v_j on the boundary of the face and have not yet been discarded. For discarding shortcuts, we use the order of shortcuts by slope as stored in the deque Q. For accepting shortcuts, we use the order along the chain C.

In Figure 8, the shortcuts that are subsequently discarded when face s_1 is treated are $\overline{v_i v_{i+5}}$, $\overline{v_i v_{i+4}}$, $\overline{v_i v_{i+7}}$, $\overline{v_i v_{i+6}}$, and $\overline{v_i v_{i+8}}$. Then the shortcuts $\overline{v_i v_{i+1}}$, $\overline{v_i v_{i+2}}$, and $\overline{v_i v_{i+3}}$ are accepted because they end in s_1. Then the next face s_2 is treated, and $\overline{v_i v_{i+9}}$ is discarded.

```
Algorithm Discard-and-Accept(S_i, i)
Input: The subdivision S_i in a topological network structure, an integer 1 ≤ i < n, and for each
    face of S_i possibly a point of P.
Output: The shortcuts starting at v_i that are consistent.
    Sort the shortcuts v_i v_j with j > i by slope and store in a deque Q. Maintain a cross-pointer
    between the deque element v_i v_j and the vertex v_j in S_i.
    j←1 (* v_j is the current position along C for accepting *)
    Assign a numbering to the faces of S_i, being s_1, ..., s_r.
    for h←1 to r
        do if a point p ∈ P is stored with s_h
              then if s_h is bounded by a maximal tangent splitter
                        then discard shortcuts from the front of the deque as long as the slopes
                             of the shortcuts are greater than the slope of v_i p
                        else discard shortcuts from the back of the deque as long as the slopes
                             of the shortcuts are smaller than the slope of v_i p
           while v_j ≠ the tangent splitter vertex of s_h
              do if v_i v_j was not discarded
                    then accept it and remove it from Q
                 j←j + 1
```

The pseudo code is given in Algorithm *Discard-and-Accept.*

Lemma 6 *Every discarded shortcut $\overline{v_iv_j}$ is inconsistent with the subchain C_{ij} with respect to the points of P.*

Proof: By construction. □

Lemma 7 *Any accepted shortcut $\overline{v_iv_j}$ is consistent with the subchain C_{ij} with respect to the points of P.*

Proof: If a shortcut is accepted, all faces it intersects have been treated by Lemma 4 and because faces are treated in order of increasing number. Since the shortcut has not been discarded when treating any of the intersected faces, the shortcut doesn't contain a point of P in a bounded face. Therefore, it is consistent with C_{ij}. □

The third step requires $O(n)$ time, which can be seen as follows. For each face, we spend $O(d+1)$ time for discarding if d segments in Q are discarded. This is obvious because discarding is simply removing from one end of the deque Q. To accept efficiently, we maintain cross-pointers between the deque Q and the chain C so that shortcuts—once they are accepted—can be removed from Q in constant time. So we spend $O(a+1)$ time if a shortcuts are accepted. Since any shortcut is discarded or accepted once, and there are a linear number of faces in S_i, it follows that the third step takes linear time.

We have completed the description of the three steps that lead to the computation of all consistent shortcuts starting at some vertex v_i of the chain. As a result, we have obtained the following:

Lemma 8 *Given an x-monotone polygonal chain C with n vertices, and a set P of m points, it is possible to compute all consistent shortcuts from any vertex of C in $O((n+m)\log n)$ time.*

If we apply this lemma to all vertices of C, construct the graph G_3 representing all consistent shortcuts, then combine the obtained graph G_3 with the graphs G_1 and G_2 (as defined in the previous section) to create the graph G, we can conclude with the following result.

Theorem 2 *Given an x-monotone polygonal chain C with n vertices, a set P of m points, and an error tolerance $\epsilon > 0$, it is possible to compute the minimum size simplification of C that is consistent with respect to P and that approximates C within the error tolerance ϵ in $O(n(n+m)\log n)$ time.*

The simplification is also simple, but this is automatic because every simplification of an x-monotone chain is again an x-monotone chain, and every x-monotone polygonal chain is simple.

4.2 Arbitrary chains

The algorithm obtained for the simplification of x-monotone chains can be generalized to arbitrary chains in several different ways. We'll describe a fairly simple way to extend the algorithm described before that most likely will give good data reduction. From the theoretical side, however, we cannot guarantee that the simplification is the minimum link simplification as in the x-monotone case.

The main idea is the following. When deciding for a vertex v_i which shortcuts starting at v_i are consistent, we will only look for shortcuts up to a certain vertex $v_{n(i)}$ with $n(i) \leq n$. The vertex $v_{n(i)}$ is chosen such that we can run the algorithm described previously on the subchain between v_i and $v_{n(i)}$ with hardly any changes at all. A first idea may be to let $v_{n(i)}$ be such that the resulting subchain is the longest x-monotone chain from v_i. This gives poor performance if the chain extends to the left very often. Clearly, we should change the coordinate system to get a long subchain $v_i, \ldots, v_{n(i)}$ to which the algorithm can be applied. A second problem we face is that it seems more difficult to guarantee simplicity of the output chain.

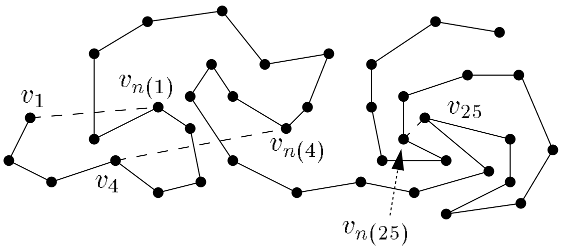

Figure 9: The vertices $v_{n(1)}$, $v_{n(4)}$, and $v_{n(25)}$ shown for the chain.

Let's consider how to determine the longest subchain from a vertex v_i to which the algorithm can be applied. One can observe that the algorithm of the previous section works for more general chains than x-monotone chains. The two features we need for the correctness are θ-monotonicity of the faces of the subdivision S_i (Lemma 2) and the intersection order of faces by half-lines (Lemma 4). These features are present if the polygonal chain doesn't cycle and doesn't have backward tangents. Intuitively, no backward tangents means that there shouldn't be an edge $\overline{v_j v_{j+1}}$ that is closer to v_i than the preceding edge $\overline{v_{j-1} v_j}$. More formally, a segment $\overline{v_i v_j}$ is a *backward tangent* if for the angles $\angle v_i v_j v_{j-1}$ and $\angle v_i v_j v_{j+1}$ (measured counterclockwise) we either have $\angle v_i v_j v_{j+1} < \angle v_i v_j v_{j-1} < \pi$ or $\pi < \angle v_i v_j v_{j-1} < \angle v_i v_j v_{j+1}$, where $i+1 < j < n$. Whether a segment is a backward tangent can easily be determined in constant time. The second condition—that the polygonal chain doesn't cycle—would give problems in the second step of the algorithm, the half-line rotating about v_i to distribute the points of P. The subchain $v_i, \ldots, v_{n(i)}$ on which we run the algorithm of the previous section is such that $v_{n(i)}$ is the first vertex after v_i for which $\overline{v_i v_{n(i)}}$ is a backward tangent, and

Algorithm *Determine-Subchain*(C, i)
Input: A simple polygonal chain C with vertices $v_1, \ldots, v_n$, and an integer $1 \leq i < n$.
Output: The vertex $v_{n(i)}$ such that $v_i, \ldots, v_{n(i)-1}$ doesn't have backward tangents and $v_i, \ldots, v_{n(i)}$ doesn't cycle.

$j \leftarrow i + 1$
$I_i \leftarrow (-\pi, \pi] -$ the angle of $\overline{v_i v_j}$
(* the interval I_i represents the angles around v_i such that a half-line starting at v_i doesn't intersect the subchain $v_i, \ldots, v_j$ *)
while $(j < n)$ and $(I_i \neq \emptyset)$ and
    $((\angle v_i v_j v_{j+1} < \angle v_i v_j v_{j-1} < \pi)$ or $(\pi < \angle v_i v_j v_{j-1} < \angle v_i v_j v_{j+1}))$
  do $j \leftarrow j + 1$
    $I_i \leftarrow I_i -$ the angles of half-lines that intersect $\overline{v_{j-1} v_j}$
if $I_i = \emptyset$
  then return v_{j-1}
  else return v_j

$C_{in(i)}$ doesn't cycle around v_i. In Figure 9, $v_{n(1)}$ and $v_{n(4)}$ are chosen because of the angle condition, and $v_{n(25)}$ is chosen because of the cycling condition. After running Algorithm *Determine-Subchain* (C, i), the coordinate system can be chosen so that the negative x-axis is any half-line with angle in I_i. The rotating half-line of the second step will make a full rotation of 2π starting and ending at the negative x-axis (instead of making a half turn from the positive to the negative y-axis in Algorithm *Distribute-Points*).

The second problem we must take care of is the parts of the chain C before v_i and after $v_{n(i)}$. These subchains can intersect shortcuts $\overline{v_i v_j}$ with $i+1 < j \leq n(i)$, which can cause self-intersections in the simplification of C. So we must prevent shortcuts from being consistent if they intersect any edge of C before v_i or after $v_{n(i)}$. There is a simple remedy here: add all vertices of $v_1, \ldots, v_{i-1}$ and of $v_{n(i)+1}, \ldots, v_n$ as extra points to the set P. Now we can show that any shortcut $\overline{v_i v_j}$ that is consistent with C_{ij} with respect to the extra points cannot destroy the condition that the output chain be simple. We conclude:

Theorem 3 *Given a simple polygonal line C with n vertices, a set P of m points, and a maximum allowed error $\epsilon \geq 0$, a simplification of C that lies within distance ϵ of C, that is consistent with C with respect to the points in P, and that is simple, can be computed in $O(n(n+m)\log n)$ time.*

5 Subdivision simplification in summary

All ingredients for subdivision simplification have been given. We next combine them—as a summary—for the general subdivision simplification algorithm. In Algorithm *Simplify-Subdivision* calls are made to previously given algorithms. However, these were assumed to work on x-monotone chains, and we've generalized the method to arbitrary chains. Therefore, the algorithms of Subsection 4.1 have to be changed slightly. They need an extra parameter to specify the vertex $v_{n(i)}$. Instead of computing the consistent shortcuts $\overline{v_i v_j}$ for $i + 1 \leq j \leq n$, we restrict to the values $i + 1 \leq j \leq n(i)$. It is trivial to adapt the algorithms this way. Furthermore, the Algorithm *Distribute-Points* should be changed to using a full sweep of the half-line, as explained in Subsection 4.2. After computing the consistent and allowed shortcuts, we construct a graph G whose arcs represent all consistent and allowed shortcuts. In this graph a minimum link path is computed. Such a path is usually called a shortest path

```
Algorithm Simplify-Subdivision(S, ϵ, P)
Input: A simple polygonal subdivision S, a positive real ϵ, and a set P of points.
Output: A simplified simple polygonal subdivision S′ such that S and S′ are consistent with
    respect to P, and that the error is at most ϵ.
1.  Determine all vertices of degree 1 or at least 3 in S. Let them be the endpoints of a collection
    𝒞 of polygonal chains.
2.  for every polygonal chain C ∈ 𝒞
3.      do Let v_1, ..., v_n be the vertices in order along C.
4.         Let G_3 be the graph with nodes v_1, ..., v_n.
5.         for i←1 to n − 1
6.             do n(i)← Determine-Subchain (C, i)
7.                Change the coordinate systems appropriately.
8.                P_i←P ∪ all vertices of S except v_i, ..., v_n(i)
9.                (* The calls to the algorithms below have an extra parameter n(i) to replace
                  n in the given algorithms for the x-monotone case. *)
10.               Compute-Tangent-Splitters (C, i, n(i))
11.               Distribute-Points (S_i, i, P_i, n(i))
12.               Discard-and-Accept (S_i, i, n(i)) and add the accepted edges as arcs to G_3
13.        Compute the graphs G_1 and G_2 using Algorithm Compute-Allowed-Shortcuts (C, ϵ).
14.        Compute graph G from G_1, G_2, G_3 by intersecting their arc sets.
15.        Determine a minimum link path from v_1 to v_n in G, and let it be the simplification
           for C.
```

in graph literature, and its computation is standard [Cormen et al. '90]. This path represents the simplification of a chain.

6 Practical considerations

In this section we will discuss some ideas that will lead to more efficient implementations or more efficient variants of subdivision simplification. We also consider two extra conditions one may want to enforce, and how they can be included in the algorithm.

A first and important speed-up can be achieved by reducing the size of the set P_i used in Algorithm *Simplify-Subdivision*. It was the union of the extra points P and all vertices of the whole subdivision, except those on the part of the chain to be simplified. In Section 2 it was already noted that one need not take this entire set. Points that lie far enough away cannot influence the choices in the simplification anyway. One observation to make is that any simplification of a chain stays within the convex hull of that chain. So only the points inside the convex hull can be relevant. For each chain C, we'll compute its convex hull and determine the subset P_C of all points of P and all vertices of other chains of S that lie inside the convex hull of C. There should be far fewer points in P_C than in any P_i. Inside the inner for-loop, we use the set P_C and add the vertices $v_1, \ldots, v_{i-1}, v_{n(i)+1}, \ldots, v_n$.

To compute the convex hull of a simple polygonal chain one can use a simple algorithm that uses only one deque in its execution [Melkman '87]. After the convex hull has been computed we test for every point in P whether it lies inside the convex hull or not. This test can be done using binary search for each point. Algorithm *Reduce-Extra-Points* gives the pseudo code. It can be called with the appropriate parameters just before the inner for-loop of Algorithm *Simplify-Subdivision*.

A second idea that can be used to speed up the simplification algorithm is the following.

Algorithm *Reduce-Extra-Points*(C, P)
Input: A simple polygonal chain C with vertices $v_1, \ldots, v_n$, and a set P of points that includes the vertices of other chains of S.
Output: The set $P' \subseteq P$ of points that lie inside the convex hull of C.
(∗ The algorithm maintains the convex hull incrementally. ∗)
1. Initialize a deque Q with the three vertices v_2, v_1, v_2.
2. (∗ Q stores the vertices of the convex hull in cyclic order starting and ending with the same vertex. ∗)
3. **for** $i \leftarrow 3$ **to** n
4.   **do for** each end of Q
5.   **do while** abv_i is a reflex turn, where b and a are the last two points in Q
6.   **do** remove vertex b from Q
7.   **if** any point was removed **then** add v_i to both ends of Q
8. Store the contents of Q (the convex hull of C) in an array to allow for binary search.
9. **for** each point p of P
10.   **do if** p lies in the convex hull **then** add p to the set P'

Suppose that we are simplifying a chain C, and thus computing for the shortcuts whether they are allowed (within distance ϵ) and consistent (respect the extra points). Suppose we know for every vertex of C the most distant vertex to which an allowed shortcut exists. Then the computation of consistent shortcuts need not go beyond that most distant vertex. We could also argue from the other perspective: if we know for each shortcut the most distant vertex to which shortcuts are consistent, then the computation of allowed shortcuts need not go beyond that vertex. Which version will be more efficient is difficult to predict. It depends on which condition is more restrictive: the (seemingly) more restrictive condition should be computed first. Our guess is that being allowed is more restrictive than being consistent (unless ϵ is large and there are many extra points). The idea sketched here can easily be incorporated in the given algorithms and therefore we omit further details.

We next discuss two issues for the extension of the algorithm to account for an extra condition that may be required in practice. The first condition is guaranteeing that the points of P cannot get too close to the simplified chain. Presently, the algorithm only keeps points to the correct side when points are considered infinitesimally small and polygonal lines infinitesimally narrow. When drawing the subdivision and the points on an output device, this assumption no longer is valid. We may want to specify a minimum clearance between the points of P and the polygonal chains of the subdivision. Such a clearance can only be guaranteed if it holds for the unsimplified subdivision. One idea to handle the situation is the following. Replace each point of P by a small number (four, six or eight) points on a small circle around it. Put these points in the set P before running the algorithm. Since these new points are respected by the algorithm, the original points of P will have some minimum distance from the simplified subdivision (assuming we have a minimum clearance in the initial situation).

A second condition that may be required for the output subdivision is that it doesn't contain any small angles between two consecutive line segments. This condition cannot always be enforced in practice. But there is a method that can help to avoid small angles. First the algorithm as it is described is run up to the point where the minimum link path in the graph G is computed. Instead, we transform graph G to another graph G_L, called the *line graph* of G. This transformation is done as follows. For every arc in G we make

Algorithm *Good*$(C, P, i, j, \epsilon, k)$
Input: A simple polygonal chain C with vertices $v_1, \ldots, v_n$, two indices i, j with $1 \leq i < j \leq n$, a real $\epsilon > 0$, and a set P of points.
Output: True if $v_i, \ldots, v_j$ can be simplified to $\overline{v_i v_j}$, and false otherwise, in which case k is the index of the farthest vertex v_k from $\overline{v_i v_j}$ for which $i < k < j$.

Determine the farthest vertex v_k from the segment $\overline{v_i v_j}$.
if $\text{dist}(v_k, \overline{v_i v_j}) > \epsilon$
  then return false and k
  else determine the polygons in $\overline{v_i v_{i+1}}, \ldots, \overline{v_{j-1} v_j}, \overline{v_i v_j}$
    for each point p in P
      do if p lies in any polygon
        then return false and k
    return true

Algorithm *Douglas-Peucker-Extra*(C, P, i, j, ϵ)
Input: A simple polygonal chain C with vertices $v_1, \ldots, v_n$, two indices i, j with $1 \leq i < j \leq n$, and a set P of points. The initial call has $i = 1$ and $j = n$.
Output: A simplification of $v_i, \ldots, v_j$ similar to the Douglas-Peucker algorithm but which respects the extra points in P.

if *Good* $(C, P, i, j, \epsilon, k)$
  then return $\overline{v_i v_j}$
  else (* v_k is returned as the farthest vertex from $\overline{v_i v_j}$ *)
    Douglas-Peucker-Extra (C, P, i, k, ϵ)
    Douglas-Peucker-Extra (C, P, k, j, ϵ)

a node in G_L. Let a and b be two nodes in G_L corresponding to arcs (v_i, v_j) and (v_h, v_k) in G, where $i < j$ and $h < k$. We connect a and b in G_L by an arc if and only if $j = h$ and the angle between the line segments $\overline{v_i v_j}$ and $\overline{v_h v_k}$ is sufficiently large. A minimum link path in G_L corresponds to a simplification where all angles of two consecutive shortcuts are sufficiently large. Unfortunately, there are a number of drawbacks with this approach. Firstly, a simplification with sufficiently large angles may not exist at all, in which case the algorithm fails to find a simplification. Secondly, the running time and the space requirements of the algorithm go up because the size of G_L can be much larger than of G. In the worst case, the algorithm needs cubic time and space. Thirdly, small angles may still show up between two consecutive chains if they are simplified separately. Fourthly, even in one simplified chain it may appear that small angles are present if there are very short line segments connecting two other line segments that make a small angle. A satisfactory solution for avoiding small angles remains to be found.

As a last issue in this section we consider the adaptation of the line simplification algorithm of Douglas and Peucker to respect extra points. In this paper we considered the adaptation of another line simplification algorithm because it yielded better data reduction. On the other hand, the Douglas-Peucker algorithm is more simple and its running time is better in practice. It depends on the application which of the two algorithms is preferable.

Roughly, the Douglas-Peucker simplification algorithm for a chain with vertices $v_1, \ldots, v_n$ initially chooses v_1 and v_n for the simplified chain. If the current simplification satisfies the ϵ-condition then we are done. Otherwise, the furthest point from the current simplification is chosen and added to the simplification. This gives two recursively defined subproblems. More can be found in [Douglas & Peucker '73]. To adapt the algorithm we need only reimplement the test whether the current simplification is good enough. It should not only depend on the ϵ-condition, but also on the extra points in P. Algorithms *Douglas-Peucker-Extra* and the subroutine *Good* gives the pseudo code for such an adaptation.

If we also wish to guarantee that the simplified chain has no self-intersections (which may arise when using the Douglas-Peucker's algorithm), further adaptations are necessary. One could test—after running the given algorithm—whether the simplification has any self-intersections. If it does, then the intersecting segments must be handled further by continuing the recursion.

The algorithm can be made more efficient in practice by reducing the size of the set P before the recursive calls are made. To this end, the convex hull of $v_i, \ldots, v_k$ is computed and all points of P that lie within this convex hull are selected to be used in the first recursive call. A similar computation is made before the second recursive call.

7 Conclusions

This paper has shown that it is possible to perform line simplification in such a way that topological relations are maintained. Points that lie above the original chain will also lie above the simplified chain, and points that lie below will remain below. Furthermore, the line simplification algorithm can guarantee a user specified upper bound on the error, and the output chain has no self-intersections. The line simplification method leads to an efficient algorithm for subdivision simplification without creating any false intersections. To obtain these results we relied on techniques from computational geometry. With ideas similar to ours, some other line simplification methods can also be adapted to be consistent with respect to a set of extra points. In particular, we showed that the algorithm in [Douglas & Peucker '73] can be extended.

The given algorithm takes $O(n(n+m)\log n)$ time to perform the simplification for a chain with n vertices and m extra points. This leads to an $O(N(N+M)\log N)$ time (worst case) algorithm for simplifying a subdivision with N vertices and M extra points. We have given several ideas to speed up the algorithm in practice. Therefore, we expect that the algorithm performs much better in practical situations than the worst case analysis suggests. An implementation is nevertheless required to discover the practical behavior of our algorithms.

The given algorithms should be fairly straightforward to implement. We plan to implement our algorithm and run it on real world data. This way we can find out in which situations the efficiency of the method is satisfactory.

References

[Asano & Katoh '93] T. Asano and N. Katoh, Number theory helps line detection in digital images – an extended abstract. *Proc. 4th ISAAC'93*, Lect. Notes in Comp. Science 762, 1993, pp. 313–322.

[Buttenfield '85] B. Buttenfield, Treatment of the cartographic line. *Cartographica* **22** (1985), pp. 1–26.

[Chan & Chin '92] W.S. Chan and F. Chin, Approximation of polygonal curves with minimum number of line segments. *Proc. 3rd ISAAC'92*, Lect. Notes in Comp. Science 650, 1992, pp. 378–387.

[Cormen et al. '90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms,* MIT Press, Cambridge, 1990.

[Cromley '88] R.G. Cromley, A vertex substitution approach to numerical line simplification. *Proc. 3rd Symp. on Spatial Data Handling* (1988), pp. 57–64.

[Douglas & Peucker '73] D.H. Douglas and T.K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer* **10** (1973), pp. 112–122.

[Edelsbrunner et al. '86] H. Edelsbrunner, L.J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15** (1986), pp. 317–340.

[Eu & Toussaint '94] D. Eu and G. Toussaint, On approximating polygonal curves in two and three dimensions. *Graphical Models and Image Processing* **5** (1994), pp. 231–246.

[Guibas et al. '93] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink, Approximating polygons and subdivisions with minimum-link paths. *Int. J. Computational Geometry and Applications* **3** (1993), pp. 383–415.

[Hershberger & Snoeyink '92] J. Hershberger and J. Snoeyink, Speeding up the Douglas-Peucker line simplification algorithm. *Proc. 5th Symp. on Spatial Data Handling* (1992), pp. 134–143.

[Hershberger & Snoeyink '94] J. Hershberger and J. Snoeyink, Computing minimum length paths of a given homotopy class. *Computational Geometry – Theory and Applications* **4** (1994), pp. 63–97.

[Hobby '93] J.D. Hobby, Polygonal approximations that minimize the number of inflections. *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms* (1993), pp. 93–102.

[Imai & Iri '88] H. Imai and M. Iri, Polygonal approximations of a curve – formulations and algorithms. In: G.T. Toussaint (Ed.), *Computational Morphology*, Elsevier Science Publishers, 1988, pp. 71–86.

[Kurozumi & Davis '82] Y. Kurozumi and W.A. Davis, Polygonal approximation by the minimax method. *Computer Graphics and Image Processing* P**19** (1982), pp. 248–264.

[Li & Openshaw '92] Z. Li and S. Openshaw, Algorithms for automated line generalization based on a natural principle of objective generalization. *Int. J. Geographical Information Systems* **6** (1992), pp. 373–389.

[Mark '89] D.M. Mark, Conceptual basis for geographic line generalization. *Proc. Auto-Carto 9* (1989), pp. 68–77.

[McMaster '87] R.B. McMaster, Automated line generalization. *Cartographica* **24** (1987), pp. 74–111.

[Melkman '87] A. Melkman, On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.* **25** (1987), pp. 11–12.

[Melkman & O'Rourke '88] A. Melkman and J. O'Rourke, On polygonal chain approximation. In: G.T. Toussaint (Ed.), *Computational Morphology*, Elsevier Science Publishers, 1988, pp. 87–95.

[Preparata & Shamos '85] F.P. Preparata and M.I. Shamos, *Computational Geometry – an introduction.* Springer-Verlag, New York, 1985.

[Sarnak & Tarjan '86] N. Sarnak, and R.E. Tarjan, Planar point location using persistent search trees, *Comm. ACM* **29** (1986), pp. 669–679.

[Zhan & Mark '93] F. Zhan and D.M. Mark, Conflict resolution in map generalization: a cognitive study. *Proc. Auto-Carto 13* (1993), pp. 406–413.