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

Marc van Kreveld
Dept. of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

Stefan Schirra
Max-Planck-Institut für Informatik
Im Stadtwald
D-66123 Saarbrücken
Germany

Abstract

The line simplification problem is an old and well-studied problem in cartography. Although there are several efficient algorithms to compute a simplification within a specified error bound, there seem to be no algorithms that perform line simplification in the context of other geographical objects. Given a polygonal line and a set of extra points, we present a nearly quadratic time algorithm for line simplification that guarantees (i) a maximum error ϵ, (ii) that the extra points remain on the same side of the output chain as of the original chain, and (iii) that the output 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, digital image analysis, and computational geometry (see the references). 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. 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 these 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

*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.

be slow. An example where this problem can be severe is in the animation of moving objects.

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 size of the subdivision is a permanent cost in a geographic information system, whereas the processing time is spent only once in many applications.

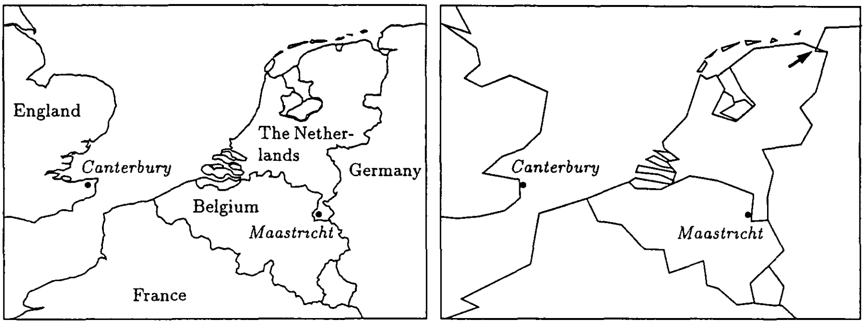

Figure 1: Part of a map of Western Europe, and an inconsistent simplification of the subdivision.

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 yield a subdivision 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 at the top of the border between The Netherlands and Germany, two borders 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.

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. In Section 5 the conclusions are given.

2 Subdivision simplification

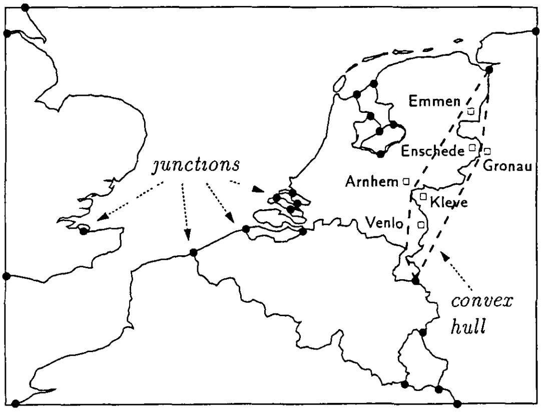

Figure 2: A subdivision with its junctions indicated.

Let S be a subdivision that models a map, and let P be a set of points that model special positions inside the regions of the map. The subdivision S consists of vertices, edges and cells. 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 as a chain, where we choose one of the vertices as start and end vertex of the chain.

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. If C is a polygonal chain, then we require from 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. All points of P lie 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 cells of the subdivision. In those two cells there may be points of P. The simplified chain between the start vertex and the end vertex will also separate two cells of the subdivision, but these cells have a slightly different shape. The fourth requirement states that the simplified chain C' must have the same subsets of points in those two cells.

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. Roughly spoken, 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. The second requirement, finally, doesn't add to the complexity 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. A simplified chain that has the points of P to the correct side 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 cells 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.

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 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 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 is the time needed in the worst case. Therefore, the algorithm may seem too inefficient for subdivisions with millions of vertices. A better analysis that also incorporates some realistic assumptions will show that the time taken in practice is much lower. It will also depend on the sizes of the chains in the subdivision, the number of extra points inside the convex hull of a chain, and the shapes of the chains themselves.

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 aclled a *shortcut* for the subchain $v_i, \ldots, v_j$. A shortcut is *allowed* if and only if the error it induces is at most some prespecified positive real value ϵ, where 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 $i \leq k \leq j$. We wish to replace C by a chain consisting of allowed shortcuts. 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 as the node set $V = \{v_1, \ldots, v_n\}$. The arc set E contains (v_i, v_j) if and only if $i < j$ and the shortcut $\overline{v_i v_j}$ is allowed. The graph G can be constructed with a simple 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 vertex 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 which 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.

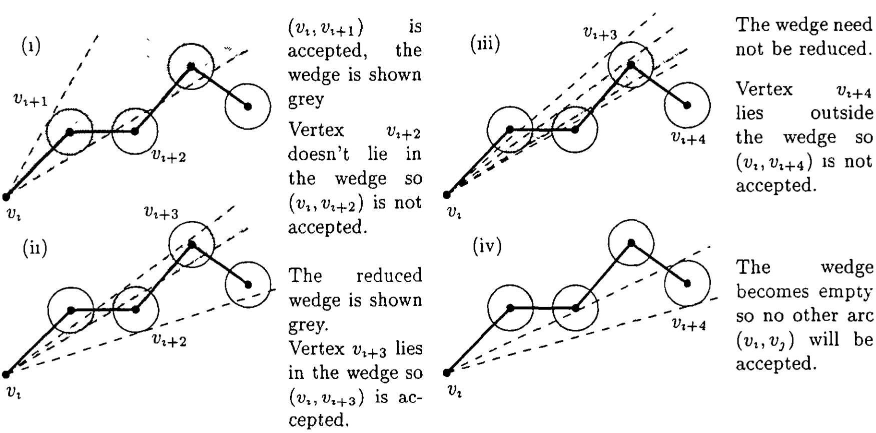

Figure 3: Deciding which arcs (v_i, v_j) with $j > i$ are accepted to G_1. Only (v_i, v_{i+1}) and (v_i, v_{i+3}) will be accepted.

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 $i \leq k \leq j$. Hence, the algorithm maintains the set of angles of half-lines starting at v_i that intersect the disks $D_i, \ldots, D_j$. 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.

With the approach sketched above, 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 overcome the two main drawbacks: it doesn't necessarily yield a simple chain and it doesn't leave extra points to the correct side. We only discuss the simplification of x-monotone chains. There are several ways to generalize our algorithms to the case of arbitrary chains. At the end of this section we sketch one method briefly; for details and extensions we refer to the full version of this paper.

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. Let C be an x-monotone simple 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 $p_1, \ldots, p_m$. From the definition of consistency we observe:

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 region formed by C and C'.*

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}. The general approach we take is to compute a graph G_3 with $\{v_1, \ldots, v_n\}$ as the node set, and an arc (v_i, v_j) whenever the bounded regions of Q_{ij} contain no points of P. So we don't consider the error of the shortcut $\overline{v_i v_j}$. This is done only later, when we determine the graph G on which the shortest path algorithm is applied. The graph G can be determined from the graphs G_1 and G_2 from the previous section, and the graph G_3 defined above. G has an arc (v_i, v_j) if and only if (v_i, v_j) is an arc in each of the graphs G_1, G_2, and G_3.

To compute arcs of the graph G_3, we consider for each vertex v_i the shortcuts $\overline{v_i v_j}$. We keep v_i fixed, and show that all arcs (v_i, v_j) with $i < j \leq n$ can be computed in $O((n + m) \log n)$ time. The first step is to sort 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 list L.

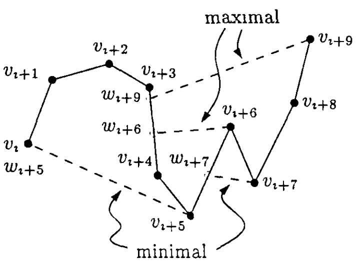

Figure 4: A part of a chain with four tangent splitters.

The second 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 4 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* (it has length zero). 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 4. 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.

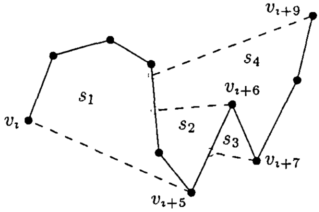

Figure 5: The corresponding subdivision S_i with cells $\gamma(1) = i + 5$, $\gamma(2) = i + 6$, $\gamma(3) = i + 7$, and $\gamma(4) = i + 9$.

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 5. The subdivision has r bounded cells, each of which is bounded by pieces of C and one or more minimal or maximal tangent splitters.

For every cell of S_i, consider the vertex with highest index bounding that cell. 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 cell the number b. The subdivision and its numbering have some useful properties.

Lemma 2 *Every bounded cell 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 cell of S_i in zero or one connected components.*

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

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

The points w_j 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 walk back quickly. For a new maximal tangent splitter we only use previously found maximal 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.

The total cost of 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 $\overline{v_j w_j}$, and once for a minimal tangent splitter). Each splitter is used as quick walk backwards only once. So we can charge the cost of the backwards walks to the $O(n)$ vertices of C and the $O(n)$ tangent splitters.

The third step of the algorithm is to distribute the points of P among the cells 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 cell of S_i can be discarded; they cannot be in a bounded region of the polygon Q_{ij} for any shortcut $\overline{v_i v_j}$. But we can discard many more points. For every cell 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 6. Similarly, if the tangent splitter with highest index is maximal, we discard all points in the cell except for the point p that minimizes the slope of the directed segment $\overline{v_i p}$. Now every cell of S_i contains at most one point of P.

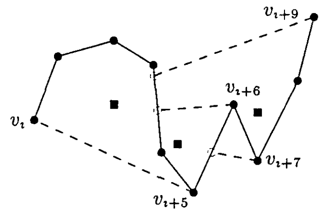

Figure 6: In each cell, only the point indicated by a square is maintained.

Lemma 5 *Any shortcut $\overline{v_iv_j}$ 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.*

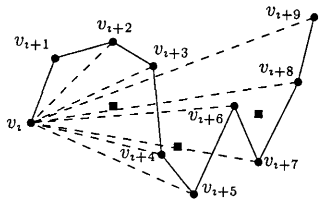

Figure 7: Only the shortcuts $\overline{v_iv_{i+1}}$, $\overline{v_i, v_{i+2}}$, and $\overline{v_iv_{i+3}}$ are accepted.

In the fourth step of the algorithm we decide which shortcuts $\overline{v_iv_j}$ are consistent and should be present in the graph G_3 in the form of an arc (v_i, v_j). We treat the cells of S_i in the order of increasing associated number. When treating a cell, we will discard any shortcut $\overline{v_iv_j}$ that has not yet been accepted and is inconsistent with respect to the one remaining point of P in that cell (if any). Then we accept those shortcut $\overline{v_iv_j}$ that have v_j on the boundary of the cell and have not yet been discarded. For discarding shortcuts, we use the order of shortcuts by slope as stored in the list L in the first step. For accepting shortcuts, we use the order along the chain C.

In more detail, the fourth step is performed as follows. Iterate through the cells $s_1, \ldots, s_r$ of S_i. Suppose that we are treating s_b. If there is no point of P in the cell s_b, then we skip the discarding phase and continue immediately with the accepting phase. Otherwise, let p_b be the point of P that lies in s_b. Assume first that the tangent splitter $\overline{w_{\gamma(b)}v_{\gamma(b)}}$ is minimal. Consider the list L of shortcuts starting at the end where shortcuts have the smallest slope. Repeatedly test whether the first shortcut at that end of the list L has larger or smaller slope than the line segment $\overline{v_ip_b}$. If the shortcut has smaller slope, then discard that shortcut by removing it from L. If the shortcut has larger slope, stop the discarding. In Figure 7, the shortcuts that are subsequently discarded when cell s_1 is treated are $\overline{v_iv_{i+5}}$, $\overline{v_iv_{i+4}}$, $\overline{v_iv_{i+7}}$, $\overline{v_iv_{i+6}}$, and $\overline{v_iv_{i+8}}$. If the tangent splitter is maximal then similar actions are taken, but on the end of the list L where the shortcuts have largest slope.

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

After the discarding phase the accepting phase starts. For all vertices v_j with $\gamma(b-1) < j \leq \gamma(b)$ on C, if the shortcut $\overline{v_iv_j}$ is still in L, accept it by removing it from L and letting (v_i, v_j) be an arc in the graph G_3.

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

The fourth step requires $O(n)$ time, which can be seen as follows. For each cell, we spend $O(d+1)$ time for discarding if d segments in L are discarded. This is obvious because discarding is simply removing from an end of the list L. To accept efficiently, we maintain pointers between the list L and the chain C so that shortcuts—once they are accepted—can be removed from L in constant time. Then 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 cells in S_i, it follows that the fourth step takes linear time.

If we perform the above steps for all vertices v_i, 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 link 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 x-monotone polygonal chain is simple. There are, however, several ways to generalize our results so that they can be applied to arbitrary, not x-monotone chains. Let C be such a chain, and let v_i be a vertex for which we wish to compute good shortcuts. One can determine a subchain $v_i, \ldots, v_k$ of C that is x-monotone after rotation of C. To assure that shortcuts $\overline{v_i v_j}$ with $j \leq k$ don't intersect edges before v_i or after v_k in the chain C, we add the vertices before v_i and after v_k to the set P of extra points. Then we run the algorithm of this section. One can show that any shortcut $\overline{v_i v_j}$ with $j \leq k$ that is consistent with respect to the extra points must be a consistent shortcut for the whole chain C, and it cannot intersect any edges of C. The generalized algorithm also runs in close to quadratic time.

5 Conclusions

This paper has shown that it is possible to perform line simplification in such a way that topological relations are maintained. Points lie above the original chain will also lie above the simplified chain, and points that lie below will remain below. Futhermore, the line simplification algorithm can guarantee a user specified upper bound on the error, and the output chain has no self-intersections. The 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. We have also developed more advanced algorithms for simplifying arbitrary chains that allow of more reduction than the algorithm based on the idea described here. These extensions are given in the full paper.

With ideas similar to ours, some other line simplification methods can also be adapted to be consistent with respect to a set of tag points. In particular, 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. There are many ideas that can be used to speed up the algorithm in practice. Therefore, we expect that the algorithm performs well in many situations, but probably not in real-time applications. Much depends on whether the quadratic time behavior of the method will actually show up on real world data.

The study in this paper has been theoretical of nature. Yet 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.

[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.

[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* **P19** (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 & 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.

[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.