A non-self-intersection Douglas-Peucker Algorithm

WU, SHIN - TING AND MERCEDES ROCÍO GONZALES MÁRQUEZ

Image Computing Group (GCI)
Department of Industrial Automation and Computer Engineering (DCA)
School of Electrical and Computer Engineering (FEEC)
State University of Campinas (Unicamp)
P.O.Box 6101, 13083-970 - Campinas, SP, Brazil
{ting,meche}@dca.fee.unicamp.br

Abstract. The classical Douglas-Peucker line-simplification algorithm is recognized as the one that delivers the best perceptual representations of the original lines. It is used extensively for both computer graphics and geographic information systems. There are two variants of this algorithm, the original $O(nm)$ method, where n denotes the number of input vertices and m the number of output segments, that works in any dimension, and the $O(n \log n)$ one, which only works for simple 2D planar polylines. In the both variants, a self-intersecting simplified line may be yielded if the accepted approximation is not sufficiently fine. Based on star-shaped subsets, we present in this paper yet another $O(mn)$ variant of Douglas-Peucker algorithm which preserves the non-self-intersection property for any predefined tolerance.

1 Introduction

Often the geometric resolution of a polyline is much higher than the resolution supported by the application, such as visualization of geographic map boundaries or visualization of curves approximated by sampling a parametric curve at regular small intervals in a raster display. For the sake of efficiency, algorithms that can extract essential features from detailed data of the original polyline and represent them on a simple one having fewer vertices that suffices for the specified resolution have been pursued by the researchers in different contexts [14, 8, 11, 7, 15, 16].

A simple line-simplification algorithm is constructing a polyline with edge segments larger than the $\epsilon > 0$ accepted tolerance. It may be achieved by discarding recursively the subsequent vertices whose distance from a prior initial vertex is less than some maximum distance $\epsilon > 0$. The vertex that is farther away than ϵ is accepted as part of the new simplified polyline, and it becomes the new initial vertex for further simplification [13].

From detailed study of mathematical similarity and discrepancy measures, the Douglas-Peucker algorithm is pointed out as the most visually effective line simplification algorithm [4, 10]. Whereas vertex reduction uses closeness of vertices as a rejection criterion, the Douglas-Peucker algorithm uses the closeness of a vertex to an edge segment. It is a recursive procedure that starts with a line segment whose extreme vertices coincide with the extreme vertices v_0 and v_n of the polyline to be simplified. Each segment $v_k v_j$ is split at the farthest vertex v_i, $k < i < j$, to it until the distance between the sequence of vertices $v_k \cdots v_i$ and $v_k v_i$ and the sequence of vertices $v_i \cdots v_j$ and $v_i v_j$ are less than the fixed tolerance ϵ.

The most time consuming part of the Douglas-Peucker procedure is the evaluation of distance formula of n input vertices to m line segments. Hence, its worst case time is $O(mn)$. Hershberger and Snoeyink [6] presented an improvement for speeding up the Douglas-Peucker algorithm, making it a $O(n \log n)$ time algorithm in the worst case. The speeding up is achieved by noting that the farthest vertex must lie on the tangents to the convex hull of the polyline which are parallel to the corresponding simplified segment. The authors proposed, then, a way to efficiently maintain the path hulls of subchains in order to use binary searches for obtaining that vertex.

However, it is known that the Douglas-Peucker algorithm does not necessarily preserve the non-intersection property of the simplified line for any predefined tolerance. Saalfeld [1, 9] observes that the self-intersection may only occur when the splitting vertex of a segment lies within the convex hull of the set of vertices associated to another segment and suggests as a solution to choose an $\epsilon\prime$ slightly less than the original tolerance ϵ for forcing further simplification.

In this paper we present a star-shaped Douglas-Peucker algorithm that preserves topological consistency between the original and the simplified 2D polylines for any predefined tolerance. We devised a sequential procedure that efficiently controls the relationship between the convex hull of the sequence of vertices associated to each segment and the rest of segments of the simplified polyline. We used the concept of signed distance: Given a sequence of counterclockwise oriented vertices $v_1 \cdots v_n$, and a point

w to which $v_i v_{i+1}$, $1 < i < n$, is the closest oriented segment, the sign of the distance of w to $v_1 \cdots v_n$ is negative if $v_i w v_{i+1}$ is clockwise oriented; otherwise, the distance is positive.

Our procedure is based on two facts: given a sequence of vertices $\mathcal{S}$, a sub-sequence $\mathcal{S}_i \subset \mathcal{S}$ of vertices on the border of a star-shaped region always builds a non-self-intersection polyline $\mathcal{P}_i$ and $\mathcal{P}_i$ may only intersect a simplified line defined by the vertices of $\mathcal{S} - \mathcal{S}_i$ if there are vertices in $\mathcal{S} - \mathcal{S}_i$ that have negative distances to the oriented segments of $\mathcal{P}_i$.

In section 2 the Douglas-Peucker algorithm is briefly described for completeness. In section 3 we give a new algorithm based on a star-shaped decomposing algorithm. A theoretical analysis of our proposal is carried out in Section 4. In section 5 some sample examples are shown. Finally, in section 6 our future research directions are presented.

2 Douglas-Peucker Simplification Algorithms

Besides its good visual results, the Douglas-Peucker algorithm is very simple to program and works for any dimension, once it only relies on the distance between points and lines. Several implementations are available at sites of internet [13, 3]. Its basic rule is that the approximation must contain (a subset of) the original data points and all the original data points must lie within a certain predefined distance to the approximation.

Given a sequence of vertices as depicted in Figure 1.a. The Douglas-Peucker algorithm has a hierarchical structure starting with a crude initial guess, namely the single edge e joining the first and last vertices of the polyline (Figure 1.b). Then the remaining vertices are tested for closeness to that edge. If there are vertices further than a specified tolerance ϵ away from the edge, then the vertex farthest from it is added to the previously simplified polyline. This creates a new approximation for the original polyline (Figure 1.c). Using recursion, this process continues for each edge (Figures 1.d,e) until all vertices of the original polyline are within ϵ (Figure 1.f).

This algorithm has $O(nm)$ worst case time and $O(n \log n)$ expected time, where n is the number of input vertices and m is the number of the segments of the simplified polyline. Note that this is an output dependent algorithm, and will be very fast when m is small, i.e. when the approximation is coarser. On the other hand, if ϵ has a larger value, the simplified polyline may intersect itself. Figure 2 illustrates a case for which three splittings on the initial segment e were sufficient for satisfying the tolerance condition, but could not avoid self-intersection. One solution is to reduce the value of ϵ, which may lead to a unnecessarily finer approximation. An alternative solution proposed by Saalfeld [1, 9] is to continue to apply the Douglas-Peucker procedure only on the part of the simplified polyline that intersects itself. Moreover, Saalfeld noted that self-intersections may only occur with vertices that lie within the closed convex hull of the vertices associated to other segments. Hence, the value of ϵ is only reduced for those vertices.

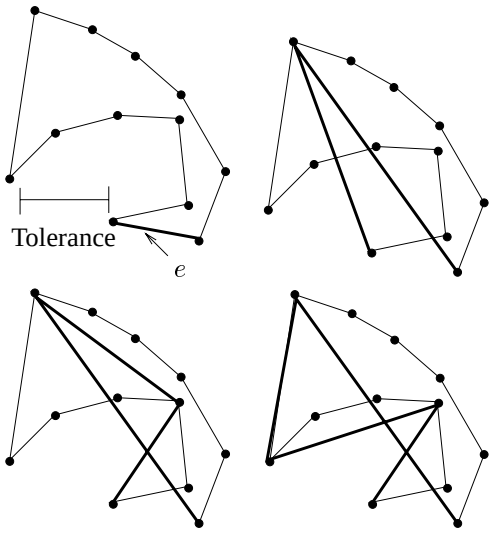


Figure 2: Self-intersection.

3 The star-shaped Douglas-Peucker Algorithm

Despite its popularity in geographical information, image processing and computational geometry applications, the Douglas-Peucker algorithm still suffers from the geometric and topological problems when the approximation tolerance ϵ is too large. In this section, we present an improvement for the method of Douglas and Peucker that can prevent self-intersections for any specified tolerance. Two facts are used: given a sequence of vertices $\mathcal{S}$, a simplified polyline of the subsequence $\mathcal{S}_i \subset \mathcal{S}$ of vertices on the border of a star-shaped region is always s non-self-intersecting polyline and, according to Saalfeld [1], this polyline may only intersect the rest of input segments if there are vertices in $\mathcal{S} - \mathcal{S}_i$ that have negative distances to the polyline.

A star-shaped region $\mathcal{R}$ is one in which the entire region, including its border, is visible from at least one fixed point $\mathcal{W}$ of $\mathcal{R}$. Hence, a polyline built by a subsequence of the original vertex sequence on the border of this region cannot cross itself. Based on this fact, we recursively partition the sequence $\mathcal{S}$ of input vertices into subsets $\mathcal{S}_0, \cdots, \mathcal{S}_i, \cdots, \mathcal{S}_m$ of sequential vertices, each of which lies in a region $\mathcal{R}_i$ and is totally visible from a fixed point $\mathcal{W}_i$. Moreover, we restrict the search domain for the farthest vertex of a simplified segment lies in $\mathcal{R}_i$ to the sequence of vertices $\mathcal{S}_i$. In this way, we ensure that the simplified poly-

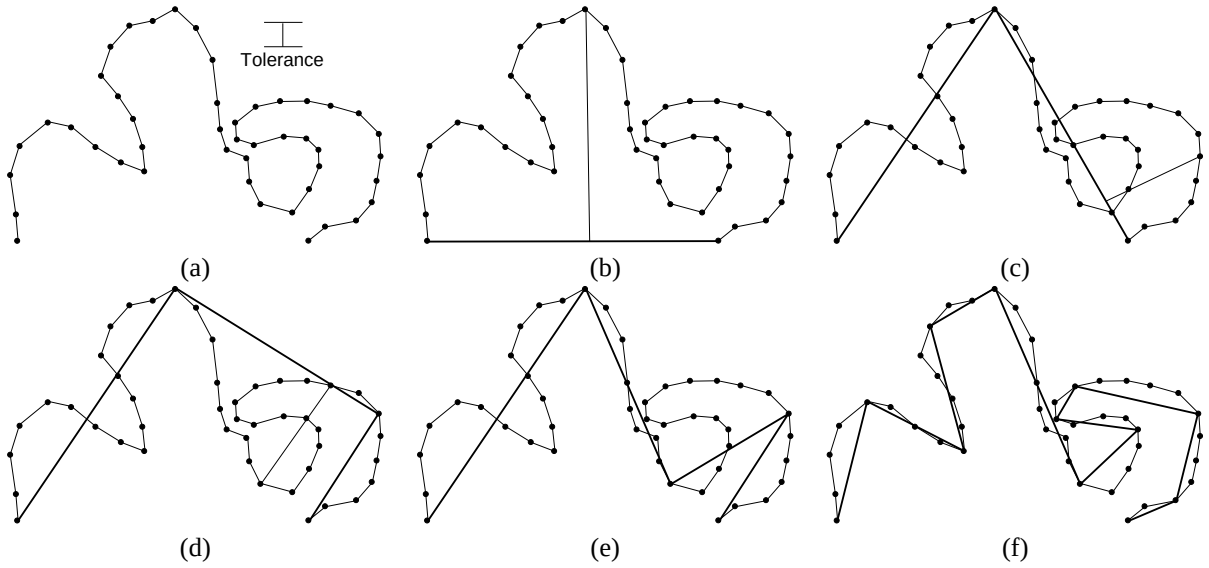


Figure 1: The basic Douglas-Peucker algorithm.

line of a sequence $\mathcal{S}_i$ cannot self-intersect.

Given a sequence $\mathcal{S}$ of input vertices of a closed non-self-intersecting polyline (Figure 3). For partitioning it into subsequences that are radially visible, we propose the following recursive algorithm: We start with a segment v_iv_j such that all the vertices v_k, $i < k < j$, have distance from v_iv_j less than the predefined tolerance ϵ and v_iv_j is inside the closed polygon. Then, we set the midpoint $\mathcal{M}_0$ on v_iv_j as the first fixed point $\mathcal{W}_0$ and determine the subset of vertices $\mathcal{S}_0 \subset \mathcal{S}$ that are visible from $\mathcal{M}_0$ (the region bordered by the solid line in Figure 3). If there are vertices whose distance to the corresponding simplified segment is greater than the tolerance ϵ, we distinguish two situations:

1. the farthest vertex v_f is in $\mathcal{S}_0$: the segment is split at it;

2. v_f does not belong to $\mathcal{S}_0$: the segment is split at the closest vertex to v_f on the border of $\mathcal{R}_0$.

The refinement is carried out until all vertices in $\mathcal{S}_0$ are under ϵ. We take the advantage of the fact that if there are vertices out of $\mathcal{S}_0$ with distance greater than ϵ, the corresponding simplified segment must have as extreme vertices two vertices on the border of the star-shaped region $\mathcal{R}_0$. The midpoint of this segment induces a new star-shaped region and is, thus, a new fixed point (e.g, the regions bordered by the dashed line in Figure 3). Successively, the procedure is performed until the distance of every vertex in $\mathcal{S}$ is under ϵ (e.g., the region visible from M_2 in Figure 3).

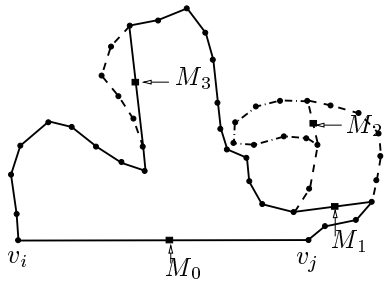


Figure 3: Star-shaped partitioning.

Observe that the determination of a fixed point for each star-shaped region is recursive, starting at a segment of the first simplified polyline. Hence, it is important to devise a procedure for obtaining this first approximation. For closed non-self-intersecting polylines, the first approximation may be the segment that joins two vertices v_i and v_j such that all the vertices v_k, $i < k < j$, have distance from v_iv_j less than the predefined tolerance and v_iv_j is inside the closed polygon. For open polylines, we should consider two cases when we join the two extreme vertices v_1 and v_n with an edge:

1. if the edge does not cross the polyline, it is then the first coarsest approximation (Figure 4.a);

2. If the edge crosses the polyline, then we distinguish two types of crossings – from left to right (LR) and from right to left (RL). If the there are alternate LR and RL crossings, then we transform it into a sequence of edges consisting of v_1, v_n and the vertices that are on the border of the convex hull of the vertices in the

half-plane and that "close" the polyline, as shown in Figure 4.b; else we order the crossings along the edge $v_1 v_n$ and take as the first approximation the segments that link the next closest vertex of each intersection point in the sequence $\mathcal{S}$ (Figure 4.c). We may also have a mix of two cases, for which we partition the sequence of input vertices into classifiable sub-sequences and handle each of them separately (Figure 4.d).

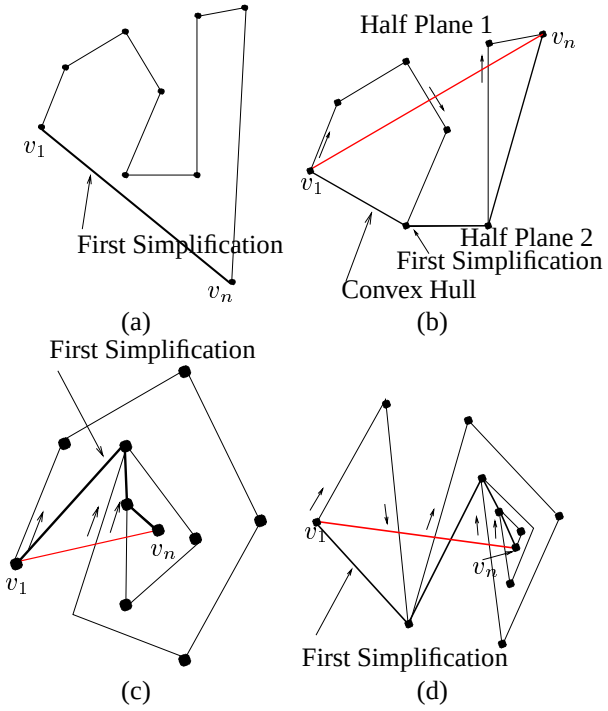


Figure 4: First approximation for an open polyline.

The restriction of the search domain to a star-shaped region guarantees that the resulting sub-polyline does not intersect, but it cannot ensure that two sub-polylines lying in disjoint star-shaped regions do not overlap (Figure 5.a). Based on the remark of Saalfeld that self-intersections may only occur with vertices that lie within the closed convex hull of the vertices associated to other subsegments [1], we propose an efficient way to identify those convex hulls by using signed distances: a segment may intersect another one only if there exist vertices associated to it with negative distance. For example, in Figure 5.a there are three segments having vertices with negative distance d_1, d_2 and d_3; thus, they are candidates for refinement. However, further refinement is only carried out if a vertex of one sub-polyline really penetrates the convex hull of the other. Figure 5.b shows the result of refinement of the segment having negative distance d_1.

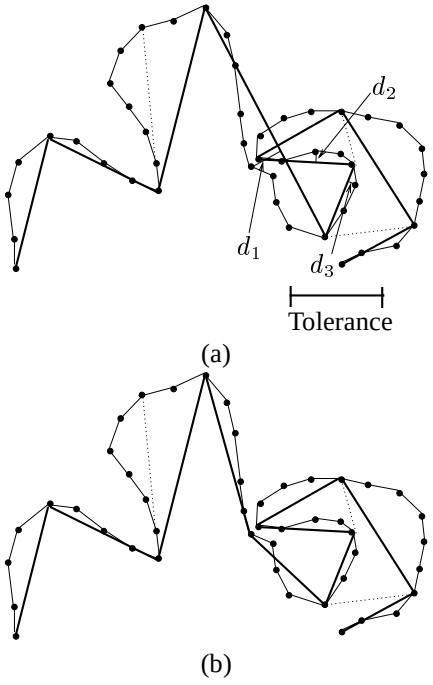


Figure 5: Further refinement.

For illustration, let us apply our procedure for simplifying the same sequence $\mathcal{S}$ of input vertices given in Figure 1.a. Figure 6.a illustrates the first star-shaped region. The start and the end vertices, s and e, are connected with a straight line as the initial rough approximation of the polyline (Figure 6.b). The Douglas-Peucker procedure is only carried out considering the vertices that lie within the first star-shaped region visible from $\mathcal{W}_0$. If all the distances between the simplified polyline and the vertices of $\mathcal{S}_0$ are less than the specified tolerance ϵ, then the procedure stops (Figure 6.c). Another star-shaped region is then determined if there are still vertices out of $\mathcal{S}_0$ that have distances greater than ϵ (Figure 6.c). The process is repeated (Figure 6.d-e) until all the distances are less than the specific tolerance (Figure 6.f).

4 Analysis

Given a polyline with n input vertices, $v_1, \cdots, v_n$:

1. Obtain the first approximation for the simplified polyline.
   (a) Determine the intersection of $v_1 v_n$ and the original polyline. If there is no intersection go to (2).
   (b) It there are alternate crossings, then determine the convex hull for the vertices lying in one half-plane and go to (2).
   (c) Order the crossings c_i along $v_1 v_n$, such that $v_1 <$

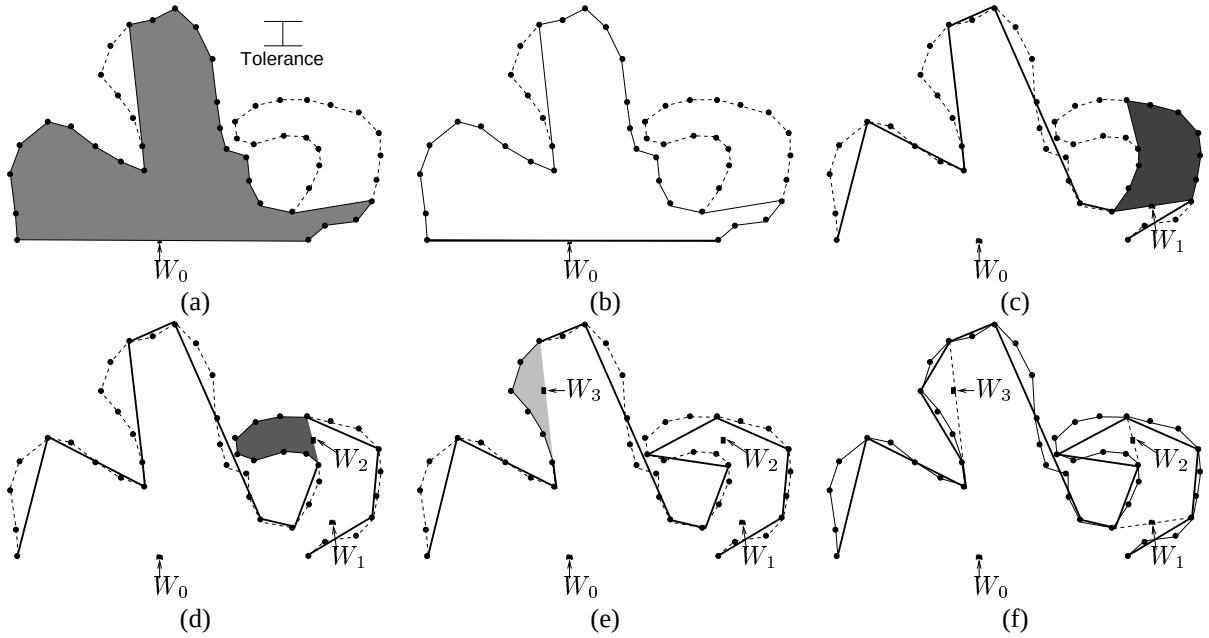


Figure 6: The star-shaped Douglas-Peucker algorithm.

$c_1 < \cdots < v_n$ and take the sequence of the nearest vertices as the first approximation.

2. Refine the first approximation:

   (a) Determine the sequence of vertices $\mathcal{S}_i$ lying in a star-shaped region.

   (b) Apply the Douglas-Peucker algorithm for these vertices.

   (c) Check the distance of vertices out of $\mathcal{S}_i$. If some have distance greater than the specified tolerance, go to (2a).

3. Further refine the polyline for eliminating the remaining self-intersections:

   (a) For each segment having vertices with negative distances, test the conflict and further refine it until the conflict is solved.

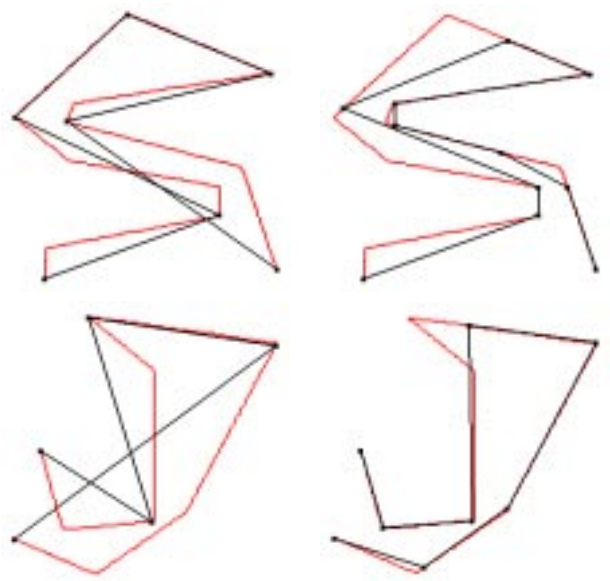
There is a variety of ways to implement our algorithm, since a number of algorithms is available for solving each subproblem. For the first step, as we already explained, the problem may be reduced to determining the convex hull of the vertices lying in a half-plane. This is a classic problem and there is a variety of algorithms. Most of them have $O(n \log n)$ time complexity, where n is the number of input vertices. The second step may be carried out in $O(nm)$ time, where n is the number of the input vertices and m the number of the output star-shaped regions. In the third step, we may use the $O(\log n)$ point-in-polygon inclusion test algorithm, where n is the number of the input polygon [5]. Hence, in the worst case, we may have $O(n \ \log \ n)$ behavior for all the inclusion tests.

Summarizing, our algorithm has in the worst case $O(nm)$ time complexity, where m depends on the number of star-shaped regions. Despite the similar timing behavior, our algorithm has over the classic Douglas-Peucker algorithm the advantage that the simplified polyline does not cross itself for any specified tolerance.

5 Results

Two sample results with larger values for ϵ are presented. For comparison purpose, we include the results output by the classical Douglas-Peucker algorithm for each input set (Figure 7). Observe that our procedure has a simplification ratio comparable to the Douglas-Peucker procedure with the advantage that no self-intersection appears in our output.

We did not perform detailed measurements for analyzing the visual effect of our algorithm. However, the simplified polylines we obtained are fair from our subjective judgment in the sense that it includes vertices where there is a large variation on the curvature of the original polyline (border of two star-shaped regions). Just for your subjective evaluation, we give in Figure 8 simplifications of the same original polyline for different values of ϵ.



(a) (b)

Figure 7: Outputs from (a) DP algorithm and (b) our proposal.

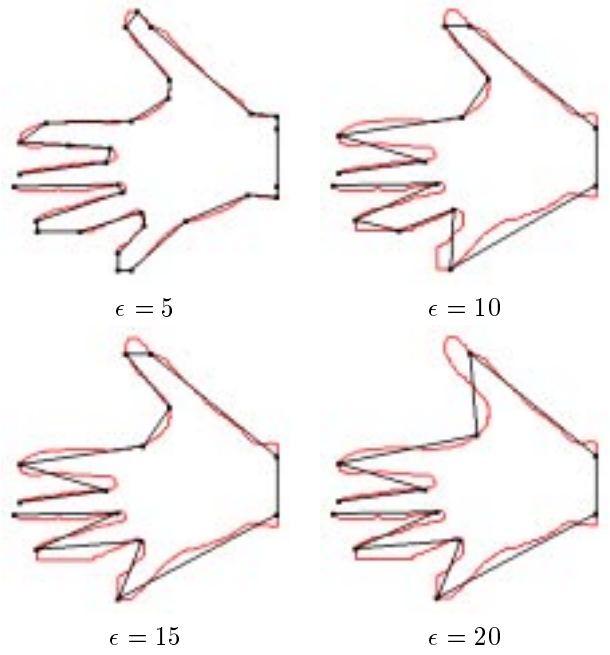


Figure 8: Simplified polyline for distinct tolerances.

The motivation of our work was the simplification of the border of a range image which is used to determine the initial simplified mesh for the 3D sample points. Hence, we also applied the proposed algorithm in the simplification of the border of real samples. Figure 9 shows the simplified polyline of the contour of the image of a frog, Figure 10 of a bunny [12] and Figure 11 presents the simplified polyline of the contour of the image of the Chopin's bust.

6 Concluding Remarks

In this paper we present, without increasing the time complexity, an improvement to the classic Douglas-Peucker line simplification algorithm in terms of preventing self-intersections. We show that slight modifications in the way that a splitting vertex is chosen at each refinement iteration, we may avoid self-intersections for the simplified lines lying in a star-shaped region. Moreover, we also present very simple criterion for identifying the conflicts that may arise among the lines lying in distinct star-shaped regions.

The time complexity of our algorithm depends on the number of induced star-shaped regions, which in its turn depends on the choice of the fixed point $\mathcal{W}$. The larger is the coverage of the field of view from a fixed point, the less tends to be the number m of the star-shaped regions. Hence, it deserves further investigation the computation of the location of fixed points to induce a less number of star-shaped regions.

Figure 9: Frog.

Figure 10: Bunny.

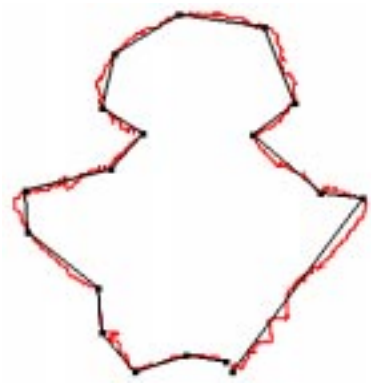


Figure 11: Chopin's bust.

Another possible research direction is to apply the proposed procedure over the the Hershberger and Snoeyink algorithm and evaluate its performance. We believe that we may take the advantages of the both methods to design a robust and efficient planar line simplification algorithm.

Our area of interest is 3D model simplification. That is, given a set of sample grid points we would like to obtain a 3D simplified polyhedron. We believe that the proposed iterative line refinement is extensible to 3D. In fact, da Silva and Wu [2] have already proposed a 3D model reconstruction that restricts the search domain to a star-shaped 3D region for avoiding self-intersections. However, their approach cannot deliver a nice perceptual representation for the simplified polyhedron. As further work, we plan to integrate in the algorithm proposed by da Silva and Wu some refinement principles adopted in the Douglas-Peucker technique for improving the visual aspect of the reconstructed 3D mesh.

7 Acknowledgments

We would like to acknowledge FAPESP for financial support under the grant number 00/10913-3.

References

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

[2] R.M. da Silva and S.-T. Wu. Reconstructing a 3d model from range images using radial flow model. In *IEEE Proceedings Sibgrapi 98*, pages 54–61, 1998.

[3] Jonathan de Halleux. An C++ implementation of Douglas-Peucker line approximation algorithm. `http://www.codeproject.com/useritems/dphull.asp`.

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

[5] Eric Haines. Point in polygon strategies. *Graphics Gems IV*, pages 24–46, 1994.

[6] John Hershberger and Jack Snoeyink. Speeding up the Douglas-Peucker line-simplification algorithm. In *Proceedings 5th Symp on Data Handling*, pages 134–143, 1992.

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

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

[9] M.Johnston, C.D. Scott, and R. Gibb. Problems arising from a simple GIS generalisation algorithm. In *Proceedings of the Eleventh Annual Colloquium of the Spatial Information Research Centre*, pages 191–200, Dunedin, New Zealand, 1999.

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

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

[12] Range images. `http://sampl.eng.ohio-state.edu/sampl/data/3DDB/RID`.

[13] Dan Sunday. Geometric algorithms. `http://geometryalgorithms.com/Archive/algorithm\_0205/algorithm\_0205.htm`.

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

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

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