International Journal of Computational Geometry & Applications
 © World Scientific Publishing Company

SIMPLIPOLY: CURVATURE-BASED POLYGONAL CURVE SIMPLIFICATION*

CHANSOPHEA CHUON

SUMANTA GUHA

PAUL JANECEK

NGUYEN DUC CONG SONG

Computer Science and Information Management Program

School of Engineering and Technology

Asian Institute of Technology

P.O. Box 4, Klong Luang, Pathumthani, 12120

Thailand

[chansophe.chuon, guha, paul_janecek]@ait.ac.th, songndc@gmail.com

Received (received date)

Revised (revised date)

Communicated by (Name)

A curvature-based algorithm to simplify a polygonal curve is described, together with its implementation. The so-called SimpliPoly algorithm uses Bézier curves to approximate pieces of the input curve, and assign curvature estimates to vertices of the input polyline from curvature values computed for the Bézier approximations. The authors' implementation of SimpliPoly is interactive and available freely on-line. Additionally, a third-party implementation of SimpliPoly as a plug-in for the GNU Blender 3D modeling software is available. Empirical comparisons indicate that SimpliPoly performs as well as the widely-used Douglas-Peucker algorithm in most situations, and significantly better, because it is curvature-driven, in applications where it is necessary to preserve local features.

Keywords: Curvature; curve simplification; Douglas-Peucker; level of detail; polygonal curve; polyline; simplification algorithm; SimpliPoly.

1. Introduction

LOD (Level of Detail) management is becoming an increasingly important issue not only in computer graphics and visualization, but also in application domains like GIS. The reason is that with the availability of powerful drawing tools, and sensory devices like high-resolution cameras, it is possible nowadays to both synthetically create extremely complex images as well as acquire them from the environment.

*A preliminary version of this paper appeared in the Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications (GRAPP 2007).

However, such images at their original or maximum resolution can be inefficient both to render and to transfer digitally. The maximum resolution, though, is often overkill for many applications. Therefore, complex images are often stored in multiple LODs, or resolutions, and the appropriate LOD – often substantially less than the original – used for a subsequent application.

Figure 1 gives an example of US map simplification done using our SimpliPoly software: the figure on the bottom right is about an eighth as complex as the original one on the top, yet is likely faithful enough for many purposes, while the one on the bottom left at half the complexity is nearly indistinguishable from the original.

Fig. 1. Multiple LODs: (a) 521 vertices (b) 263 vertices (c) 61 vertices.

A variety of techniques have been developed for the purpose of computing LODs, typically beginning with the original object, then increasingly coarse, a process called simplification. See Heckbert and Garland¹¹ and Luebke et al¹⁵ for surveys. Most of the current literature in computer graphics and available software deals with the simplification of surfaces in 3D.

We focus in this paper on a somewhat restricted LOD problem, that of simplifying a plane polygonal curve, also called polyline. Nevertheless, this problem is of interest in several application areas as well, ranging from sketch simplification to trend analysis to map simplification.

The main contribution of this paper is an algorithm and its derivative software package – together called SimpliPoly – which can be used interactively to simplify a polyline. SimpliPoly makes a novel application of the notion of curvature from differential geometry to the simplification problem. In particular, our approach is to *estimate* the curvature of a polyline at its vertices.

A polyline C , except for the trivial situation where all its vertices lie on a straight line, is not smooth. Accordingly, differential geometric measures of curvature cannot be applied directly to C itself. Our method is to, instead, first approximate pieces of C with Bézier curves, and then “lift” curvature values from points of the approximations to vertices of C . Subsequently, vertices of C with low curvature are removed on the premise that they lie on “straightish” (equivalently, “weakly featured”) parts of C .

The technique is motivated mostly by heuristic arguments, our goal first and foremost being simple and practical software. We do not prove results about the quality of the simplification. Unfortunately, it seems difficult to do so, given the nature of the heuristics employed; nevertheless, they are all fairly intuitive and justified by the context. Moreover, the SimpliPoly software package is designed to be used interactively, so that the user monitors the output until the simplification is perceived to be satisfactory, rather than depending upon any *a priori* guarantee of its quality. Our results – which the reader is invited to verify using the on-line Java version¹⁶ – indicate that SimpliPoly indeed works well in practice, producing faithful simplifications over a range of user-controlled parameters. In fact, this has led to a third-party implementation of SimpliPoly as a plug-in¹⁷ for the free open-source GNU Blender 3D modeling software².

Currently, the polyline simplification algorithm most commonly used in practice is Douglas-Peucker⁸, which is a recursive algorithm based on reducing a distance error between an input polyline and its simplification. Douglas-Peucker is easily implemented, efficient, and generally produces good-quality output – all reasons for its popularity. However, Douglas-Peucker is purely “distance-error driven”; it does not “recognize” local features, e.g., a sharp turn, a protrusion, etc., and, therefore, makes no attempt to specially preserve these. SimpliPoly, on the other hand, being “curvature-driven”, does recognize features and tends to keep them. At the same time, because stretches of a polyline with low distance error from a simplification tend in practice to be of low curvature values as well, the ability of SimpliPoly to simplify such stretches appears to be competitive with Douglas-Peucker.

At present, the theoretically best algorithms for polyline curve simplification are from Agarwal et al¹ and Chen and Daescu⁵, with further refinements in Daescu and Mi⁶. They obtain provably near-optimal simplifying approximations within specified error bounds on the distance between the original curve and the approximation. Gudmundsson, Narasimhan and Smid¹⁰ solve a related problem where the error between the original and simplified curve is measured by the difference in the rate of change of path length along either. However, all these preceding algorithms are fairly complex and there seem to be no practical implementations yet. Moreover, the

only curvature-based simplification algorithms that we are aware of are due to Kim et al¹³ and references therein, all of which deal with surfaces rather than curves. Again, these algorithms are complex with no code announced. Buzer³, Carmona-Poyato et al⁴ and Kolesnikov and Fränti¹⁴ consider the important related practical problem of simplifying digital curves.

The next section discusses SimpliPoly in detail, both the algorithm and its analysis, and introduces the software. In Section 3 we make an extensive comparison between SimpliPoly and Douglas-Peucker. Applications, and advantages to using SimpliPoly, are the topic of Section 4. We conclude in Section 5.

2. SimpliPoly

2.1. Algorithm

Let C be a polyline joining the $n+1$ vertices $v_0, v_1, \dots, v_n$, where $n \geq 5$. SimpliPoly computes the Bézier approximation of a “sliding window” of length five vertices of C as follows. First, it determines the Bézier curve $c_0(u)$, $0 \leq u \leq 1$, of order five with vertices $v_0, v_1, \dots, v_4$ as control points, then the curve $c_1(u)$ with $v_1, v_2, \dots, v_5$, as control points, and so on, till the curve $c_{n-4}(u)$ with $v_{n-4}, v_{n-3}, \dots, v_n$ as control points.

The formula for $c_i(u)$ is

$$c_i(u) = \sum_{r=0}^4 B_{r,4}(u) v_{i+r}, \quad 0 \leq i \leq n-4$$

where

$$B_{r,4}(u) = \binom{4}{r} (1-u)^{4-r} u^r, \quad 0 \leq r \leq 4$$

are the blending functions, which happen to be the Bernstein polynomials of degree four in this case. See Farin⁹ for a discussion of Bézier theory in general.

Note: There is no particular reason to choose order five for the Bézier approximation, other than that it seems to span a reasonable number of vertices of C , and yet is not large enough to be computationally problematic.

Figure 2 shows the first two approximating curves for a polyline: $c_0(u)$ (solid) and $c_1(u)$ (dashed).

Next, for each Bézier curve $c_i(u)$, $0 \leq i \leq n-4$, SimpliPoly determines the curvature at the three interior points $c_i(0.25)$, $c_i(0.5)$ and $c_i(0.75)$, and “lifts” them to the control vertices v_{i+1} , v_{i+2} and v_{i+3} , respectively. This is done as the blending function corresponding to each of these three vertices has maximum value at the point of $c_i(u)$ whose curvature is lifted. Intuitively, then, each control vertex picks up its curvature from the point of the curve at which it has most influence. Curvature at the end vertices are not used on the premise that data from a surrounding neighborhood of a point should be available if curvature there is to be applied for the purpose of simplification (even though it can be mathematically calculated).

Fig. 2. The solid curve is $c_0(u)$, the dashed one $c_1(u)$, and the arrows point in the direction curvature is lifted (only the liftings from $c_0(u)$ are shown).

Figure 2 illustrates the scheme (only the curvature liftings from the first Bézier curve are indicated). The formula used to compute curvature is from classical differential geometry⁷: the curvature of a plane curve $c(t) = x(t)\mathbf{i} + y(t)\mathbf{j}$ is given by

$$\kappa(t) = \frac{x'(t)y''(t) - x''(t)y'(t)}{(x'(t)^2 + y'(t)^2)^{3/2}}$$

As the 5-vertex window slides from one end of C to the other, all the vertices from v_3 to v_{n-3} lift curvature values from three different Bézier curves $c_i(u)$ each, while v_0 and v_n lift no curvature values, v_1 and v_{n-1} lift one each, and v_2 and v_{n-2} two each. Figure 3 is a schematic illustration.

Heuristically, next, SimpliPoly averages the set of curvature values at each vertex v_i (except for v_0 and v_n , of course) to determine the so-called *pseudo-curvature* k_i , $1 \leq i \leq n-1$, of that vertex, a value used as a curvature estimate. E.g., if the curvatures values of $c_0(u)$, $c_1(u)$, $c_2(u)$ are as indicated in Figure 3 (at parameter values 0.75, 0.5 and 0.25, respectively), then the pseudo-curvature of v_3 is 0.33.

Fig. 3. Schematic illustration: determining the pseudo-curvature of a polyline vertex from a sliding window of Bézier approximations.

Finally, SimpliPoly removes all vertices v_i whose pseudo-curvature $k_i \leq \kappa$, where κ is a user-defined *curvature threshold*.

An option in SimpliPoly allows the user to control in a simple manner the *local distance error* in the simplification as vertices are removed. SimpliPoly removes vertices by processing from the high end to the low, beginning possibly with v_{n-1} .

If a *distance error threshold* $\epsilon > 0$ is specified, then, before removing a vertex v_i according to the pseudo-curvature constraint, SimpliPoly determines the perpendicular distance d of v_i to the straight line joining v_{i-1} and v_{i+1} , and, if this distance exceeds ϵ , v_i is skipped, i.e., not removed. See Figure 4.

Fig. 4. v_i is not removed if local error distance $d > \epsilon$.

SimpliPoly can be run on-line, or, locally, by downloading the source from its site¹⁶. A Java run-time is required.

2.2. Complexity

The run-time of SimpliPoly is asymptotically linear in the number n of vertices of the polyline being simplified. This is trivial to see: there are $O(n)$ window positions, with a constant amount of curvature computation for the Bézier curve in each; subsequently, there is a constant amount of computation per vertex of the polyline to determine the pseudo-curvature at that vertex, and process it for possible removal.

3. Comparisons with Douglas-Peucker

For polygonal curve simplification, Douglas-Peucker⁸ is the *practical* algorithm “to beat”. We have experimented with SimpliPoly and Douglas-Peucker extensively. From a subjective evaluation of the two – we compared, therefore, the perceived quality of their outputs – SimpliPoly almost always matches Douglas-Peucker and, in certain cases, which we’ll discuss momentarily, seems to surpass it in preserving evident features.

Theoretically, the worst-case complexity of a straightforward recursive implementation of Douglas-Peucker is $O(mn)$, where n is the input number of vertices and m is the output number. The Douglas-Peucker worst-case complexity can be improved, though, to $O(n \log n)$ by using convex hulls to speed up distance computations, as shown by Hershberger and Snoeyink¹². As discussed earlier, SimpliPoly is always $O(n)$, regardless of output size. However, from our experiments with up to a few thousand vertices (which bounds the complexity of most practical applications) on a reasonable desktop, time consumed seems not to be an issue with either algorithm.

Qualitatively, the major difference between the two is seen in the reduction of two different example polylines each consisting of 20 vertices, along the top and bottom rows of Figure 5, respectively. The original polyline is drawn unbroken in all four diagrams, while the dashed polylines in the left two diagrams are from a reduction to 6 vertices using Douglas-Peucker, and the dashed polylines in the right two from a reduction to 6 vertices using SimpliPoly.

Fig. 5. SimpliPoly vs. Douglas-Peucker: Two different polylines, one on the top row and one on the bottom. The original polyline of 20 vertices is drawn unbroken in all four diagrams. Dashed polylines in the left two diagrams are from a reduction to 6 vertices using Douglas-Peucker, and the dashed polylines in the right two diagrams from a reduction to 6 vertices using SimpliPoly.

As can be seen, SimpliPoly seems to respect the highly-curved pieces more than Douglas-Peucker. The reason, of course, is that the former examines pseudo-curvatures through a local window, thereby detecting features that the global Douglas-Peucker method cannot. However, there is a penalty that SimpliPoly pays if it totally disregards distance error – the two experiments were both run with SimpliPoly distance error checking turned off – as can be seen particularly in Figure 5(d), where the reduced polyline strays significantly from the gently curved part of the original on the left, as well as on the right where it joins the two features. It is precisely for this reason that the option to constrain local distance error is provided.

It is difficult to compare more analytically the fidelity of SimpliPoly and Douglas-Peucker to features, either experimentally or theoretically, as there seems to exist no easily computed measure of this property. We decided, instead, to compare experimentally the total distance error, which can be efficiently defined as follows:

If, at the end of the reduction process, the remaining vertex sequence is $v_0 = v_{i_0}, v_{i_1}, \dots, v_{i_k} = v_n$, then the total distance error is

$$E = \sum_{j=0}^n e(v_j)$$

where $e(v_j)$, the distance error at the vertex v_j , is defined to be 0 if $v_j = v_{i_r}$, for some r , and, otherwise, equal to the perpendicular distance from v_j to the straight line joining v_{i_r} and $v_{i_{r+1}}$, where $i_r < j < i_{r+1}$. E.g., in Figure 6, if the original polyline $v_0v_1v_2v_3v_4$ is reduced to v_0v_4 then the total distance error is $E = e(v_1) + e(v_2) + e(v_3)$.

Fig. 6. Total distance error after simplification if v_1 , v_2 and v_3 are removed.

For a large scale practical comparison we used stock market data, taking the daily closing share prices of two companies listed on the NASDAQ, viz., AMD and DATA TRAK International between May 8, 2007 and May 2, 2008, as input curve data for both SimpliPoly and Douglas-Peucker algorithms. We multiplied the share prices by 100 to eliminate fractional values. Figure 7, in particular, shows the AMD price curve and its simplification by 0%, 26%, 51% and 66% via both SimpliPoly

and Douglas-Peucker (the Douglas-Peucker reductions are on the left, SimpliPoly reductions on the right, and they increase through 0%, 26%, 51% and 66% from top to bottom).

The table below shows the comparison between SimpliPoly and Douglas-Peucker applied to both AMD and DATA TRAK in terms of the ratio of their respective total distance errors. In the experiments we kept the (local) distance error threshold of SimpliPoly equal to the (global) error threshold of Douglas-Peucker.

AMD		DATA TRAK	
Percentage Reduction	Total Error Ratio SP/DP	Percentage Reduction	Total Error Ratio SP/DP
26	1.07	26	1.01
51	1.04	51	1.16
66	1.44	75	1.61

SimpliPoly is clearly competitive with Douglas-Peucker using the total distance error metric, even though the latter is driven by distance error, while the former by curvature (together with a local distance error check). The outputs of SimpliPoly and Douglas-Peucker are virtually indistinguishable at low resolution as can be seen by comparing the polylines on the left and right of Figure 7, while zooming in to a higher resolution will show a greater fidelity to local features, as in Figure 5, on the part of SimpliPoly.

4. Applications

Curve simplification algorithms are used to reduce the complexity of 2D graphical scenes in a variety of application domains. As discussed in the introduction, curve simplification is often invoked to pre-calculate a family of models at successively coarser levels of detail. This allows an application to dynamically make trade-offs between image quality and performance by choosing between representations of varying complexity. For example, a drawing program might use simplified models while the user is interactively scrolling or zooming a scene in order to maintain high frame rates.

The main strength of the SimpliPoly algorithm, when compared with a global error-based algorithm such as Douglas-Peucker, is its sensitivity to local curvature, as was seen in the previous section. This makes it particularly promising for applications where distinctive local features must be preserved. For example, terrain features in cartographic maps, such as coastlines and elevation lines, are often “noisy”, with many sharp local changes in direction that vary in magnitude over the length of the feature. An error-based algorithm will focus on the vertices that extend outside a particular tolerance. In contrast, a curvature-based algorithm will focus on the corners (the points of high curvature) that define the distinctive shape of the feature. Additionally, the local distance error threshold of SimpliPoly can be

Fig. 7. AMD share price polyline with 250 vertices originally: the Douglas-Peucker reductions are on the left, SimpliPoly reductions on the right, and they increase through 0%, 26%, 51% and 66% from top to bottom.

applied interactively to capture features that change more gradually (regions of low curvature).

It is difficult for any single algorithm to create a high-quality simplification for both sharp and gradual changes in curvature. Nevertheless, the SimpliPoly software allows the user to interactively adjust the curvature and local distance error thresholds of the algorithm to obtain a simplification of the appropriate complexity and quality.

Figure 8 illustrates how the local distance error check helps improve the performance of SimpliPoly. Figures 8(a)-(c) reductions are to 5, 9 and 13 vertices, respectively, with no local distance error checking, while Figure 8(d) reduces to 13 vertices with a local distance error check. Comparing Figures 8(c) and (d) one sees that a local distance error check enhances SimpliPoly’s fidelity to gentler features.

Fig. 8. SimpliPoly simplification with/without local distance error (LDE) check. Original polyline has 65 vertices.

5. Conclusions

The primary contribution of this paper is the description of a novel, efficient and high-quality curve simplification algorithm with a working implementation available for practical use.

SimpliPoly is a linear-time method based on curvature estimation via piecewise Bézier approximations. The implementations – both the authors’ version¹⁶ and a third-party plugin-in¹⁷ for the Blender 3D modeling package – allow the user to interactively adjust the curvature and local distance error thresholds to control a trade-off between quality and complexity. The algorithm gives results of comparable – and often superior – quality to the popular Douglas-Peucker algorithm.

Future work: It is clear from our experiments that, in order to be feature-sensitive, a curve simplification algorithm must be informed by curvature, in addition to distance error. SimpliPoly is curvature-driven but allows to use local distance error as a constraint (while the currently dominant Douglas-Peucker algorithm is purely driven by global distance error).

However, it might be possible to improve how the distance error threshold is input and applied in SimpliPoly. Currently, it's an independent parameter which the user is asked to set, typically from a visual estimation of the approximation quality, and applied as a local check before removing a vertex. It would be interesting if the input process could be somehow automated and the threshold used as a global upper bound on distance error (which, of course, points to a hybridization with Douglas-Peucker).

Acknowledgment. We would like to thank the anonymous reviewers for helping improve the paper.

References

1. P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. In *Tenth European Symposium on Algorithms*, pages 29–41, 2002.
2. Blender 3D Modeling Software: <http://www.blender.org>, 2002.
3. L. Buzer. Optimal simplification of polygonal chain for rendering. In *23rd Annual ACM Symposium on Computational Geometry*, pages 168–174, 2007.
4. A. Carmona-Poyato, F. J. Madrid-Cuevas, R. Medina-Carnicer, and R. Muñoz Salinas. Polygonal approximation of digital planar curves through break point suppression. *Pattern Recognition*, 43:14–25, 2010.
5. D. Z. Chen and O. Daescu. Space-efficient algorithms for approximating polygonal curves in two-dimensional space. *International Journal of Computational Geometry and Applications*, 13:95–111, 2003.
6. O. Daescu and N. Mi. Polygonal chain approximation: a query based approach. *Computational Geometry*, 30:41–58, 2005.
7. M. P. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976.
8. 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*, 10:29–41, 1973.
9. G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*, volume 5. Morgan Kaufmann, 2001.
10. J. Gudmundsson, G. Narasimhan, and M. Smid. Distance-preserving approximations of polygonal paths. *Computational Geometry*, 36:183–196, 2007.
11. P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical Report CMU-CS-95-194, Carnegie Mellon University, 1995.
12. J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Fifth International Symposium on Spatial Data Handling*, pages 134–143, 1992.
13. S.-J. Kim, C.-H. Kim, and D. Levin. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 26:657–663, 2002.
14. A. Kolesnikov and P. Franti. Polygonal approximation of closed discrete curves. *Pattern Recognition*, 40:1282–1293, 2007.

15. D. Luebke, M. Reddy, J. D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.
16. Polygonal curve simplification software: <http://www.cs.ait.ac.th/~guha/SimpliPoly/simpliPoly.html>, 2009.
17. User:Testscreenings/simplify_curves: http://wiki.blender.org/index.php/User:Testscreenings/simplify_curves, 2010.