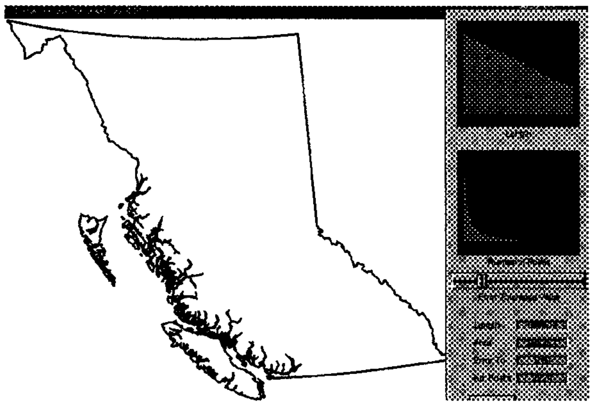
An $O(n \log n)$ Implementation Of The Douglas-Peucker Algorithm For Line Simplification

John Hershberger
Mentor Graphics

Jack Snoeyink
Department of Computer Science
University of British Columbia

Introduction

An important task of the cartographer's art is to extract features from detailed data and represent them on a simple and readable map. As computers become increasingly involved in automated cartography, efficient algorithms are needed for the tasks of extraction and simplification. Cartographers [1, 6] have identified the *line simplification problem* as an important part of representing linear features. Given a *polygonal chain*, a sequence of vertices $V = \{v_0, v_1, \ldots, v_n\}$, the problem is to find a subsequence of vertices that approximates the chain V. We assume that the input chain V has no self-intersections (but not the output).



In 1973, Douglas and Peucker (now Poiker) [3] published a simple recursive simplification method in the cartography literature that has become the favorite of many [5, 10]. This method has been independently proposed in other contexts such as vision [8] and computational geometry [9]. Our video animates two different implementations of this simplification method—the straightforward implementation suggested in Douglas and Peucker and the implementation of Hershberger and Snoeyink that improves the worst-case running time.

We describe the method, the implementations, and the animation.

Method

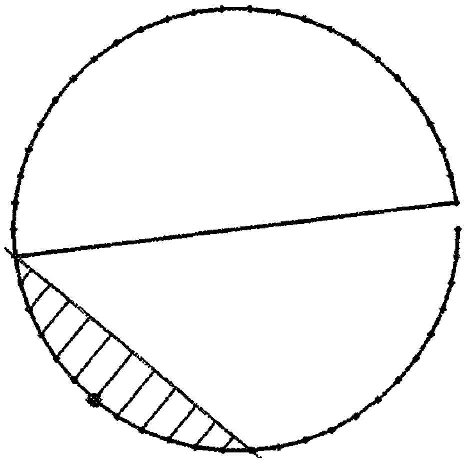
Given a polygonal chain $V = \{v_0, v_1, \ldots, v_n\}$, the recursive line simplification method proceeds as follows: Initially, approximate V by the line segment $\overline{v_0 v_n}$. Determine the farthest vertex v_f from the line $\overleftrightarrow{v_0 v_n}$. If its distance $d(v_f, \overleftrightarrow{v_0 v_n})$ is at most a given tolerance $\varepsilon \geq 0$, then accept the segment $\overline{v_0 v_n}$ as a good approximation to V. Otherwise, break V at v_f and recursively approximate the subchain $\{v_0, v_1, \ldots, v_f\}$ and $\{v_f, \ldots, v_n\}$.

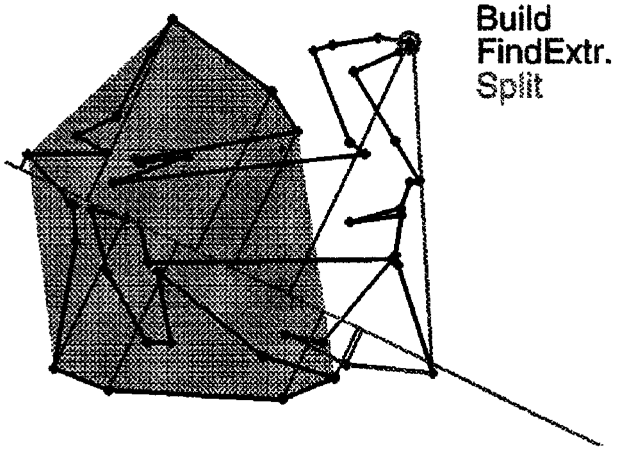
Algorithms

The two implementations differ in the way they find v_f, the farthest vertex from $\overleftrightarrow{v_0 v_n}$.

The straightforward approach measures distance from $\overleftrightarrow{v_0 v_n}$ for each vertex and takes the maximum. Thus, the running time of this algorithm will satisfy a quicksort-like recurrence, with best case $\Theta(n \log n)$. Since geometry determines the farthest vertex, the worst-case running time is $\Theta(n^2)$. One cannot use linear-time median finding or randomization to improve the running time. (If one assumes that the input data is such that farthest vertices are found near the middle of chains, then one expects $O(n \log n)$ behavior.)

The second implementation uses a *path hull data structure* [2, 4] to maintain a dynamic convex hull of the polygonal chain V. Using Melkman's convex hull algorithm [7], we compute two convex hulls from the middle of the chain outward. We can find





a farthest vertex v_f from a line by locating two extreme points on each hull using binary search. When we split V at vertex v_f, we undo one of the hull computations to obtain the hull from the "middle" to v_f, recursively approximate the subchain containing the middle, then build hulls for the other subchain and recursively approximate it. One can use a simple credit argument to show that the total time taken by the algorithm is $O(n \log n)$. The best case can even be linear, if all the hulls are small and splits occur at the ends.

Unfortunately for us, the statistical properties of cartographic data usually means that the straightforward implementation is slightly faster than the path hull implementation. Since the variance of the running time of the path hull implementation is smaller, it may be interesting for parallel or interactive applications.

Animation

The programs were run on a Silicon Graphics Crimson in the GraFiC lab at UBC and output was recorded to video in real time.

Acknowledgements

Scott Andrews extracted the data of the coastline of British Columbia from the Digital Chart of the World. NSERC and the B.C. Advanced Systems Institute supported the work on this video. DEC Systems Research Center has also provided partial support for the research.

References

[1] B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1-26, 1985.

[2] D. Dobkin, L. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Algorithmica*, 10:1-23, 1993.

[3] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112-122, 1973.

[4] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Intl. Symp. Spatial Data Handling*, pages 134-143. IGU Commission on GIS, 1992.

[5] R. B. McMaster. A statistical analysis of mathematical measures for linear simplification. *Amer. Cartog.*, 13:103-116, 1986.

[6] R. B. McMaster. Automated line generalization. *Cartographica*, 24(2):74-111, 1987.

[7] A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Info. Proc. Let.*, 25:11-12, 1987.

[8] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Comp. Vis. Graph. Image Proc.*, 1:244-256, 1972.

[9] G. Rote. Quadratic convergence of the sandwich algorithm for approximating convex functions and convex figures in the plane. In *Proc. 2nd Can. Conf. Comp. Geom.*, pages 120-124, Ottawa, Ontario, 1990.

[10] E. R. White. Assessment of line-generalization algorithms using characteristic points. *Amer. Cartog.*, 12(1):17-27, 1985.