MapShaper.org: A Map Generalization Web Service

Matthew Bloch and Mark Harrower
Department of Geography, University of Wisconsin-Madison
550 N. Park St, Madison WI 53703 U.S.A.
e-mail: matt@grammata.com and maharrower@wisc.edu
ph: 608-265-0012 fax: 608-265-3991

ABSTRACT: Despite recent advances in map-generalization tools and theories, this work has been slow to find its way into the hands of the map-making public. Mapshaper.org is a free-to-use generalization web service developed to help mapmakers simplify and smooth their vector linework using a suite of visual-editing tools. This paper details some of the technical barriers that had to be solved to enable real-time WYSIWIG map generalization: (1) optimized distribution of work between client and server, (2) support for Douglas-Peucker, Visvalingham-Whyatt, and weighted Visvalingham-Whyatt, and (3) data compression and file-format optimization for near real-time editing capabilities.

1. INTRODUCTION

Because maps are scale models of geographic reality they almost always require some form of graphic generalization. Furthermore, even on the same map, different data layers may require different amounts of generalization to accommodate both the purpose of the map (e.g., tourism versus navigation) and the nature of the source data (e.g., different source scale for roads and contours). Although much of this work is now done digitally—and, in some cases, automatically—for many mapmakers generalization remains a surprisingly time-consuming and labor-intensive process, since every dataset, every map, and every audience present a unique design challenge (Ormsby and Mackaness 1999). As a result, in the last 30 years, academic and industry researchers have focused their energies on developing better approaches to map generalization, such as agent-based modeling (Hardy et al. 2003), or continuous optimization approaches, such as "beams" and "snakes" (Bader 2001, Galanda and Weibel 2003), in the hopes of improving the rendering speed, visual quality, and flexibility of the maps created from our ever-growing geospatial databases. This work spans a continuum from very technical to highly conceptual (Mackaness and Edwards 2002). One current branch of this broad research effort is learning how to generalize maps on-the-fly for web-based mapping services (such as Google Maps), and how to port those on-demand maps across a range of display devices (Bédard and Bernier 2002, Burghardt et al. 2005). The large number of papers, conferences, and coordinated international efforts (such as research commissions and web-data standards that incorporate generalization principals) speaks to the relevance of this topic today.

While a few advanced map-generalization systems have been built—notably Laserscan's Clarity system—much of the research on map generalization has yet to find its way into commercial software and thus into the hands of everyday map makers. As a step in bridging that gap, this paper reports on MapShaper which is a free-to-use, Flash-based web service designed to help

professionals and novices alike generalize their cartographic data. MapShaper contains tools for both simplification (reducing the number of vertices in a polyline) and smoothing (reducing the jaggedness of a polyline). Unlike other commonly available digital-map tools—such as ArcGIS and Mapublisher—MapShaper allows the user to work visually in a WYSIWYG ("what you see is what you get") environment. MapShaper contains efficient implementations of both the Douglas-Peucker (DP) and the Visvalingham-Wyatt algorithms, enabling the user to adjust the level of generalization and see the changes almost instantaneously. This means no more tinkering with opaque variables to achieve the desired level of generalization. Figure 1 shows the main MapShaper interface.

MapShaper offers a number of significant advantages over current commercial software for *simplification* and *smoothing* of vector linework: (1) unlimited undos, (2) a new breed of tools like the *generalization brush* and *eraser*, (3) vertex locking/unlocking, (4) visual overlay of original data, (5) the conversion of polylines to Bezier curves, and (6) the automatic construction of polygon topology. As a web service, the latest version is always live, is platform independent, and allows users to upload and save their work (to continue editing at a later time if they wish). Currently, the application supports importing ESRI Shapefiles (.shp) and exporting/conversion to .shp, .ai, .eps, and .as files.

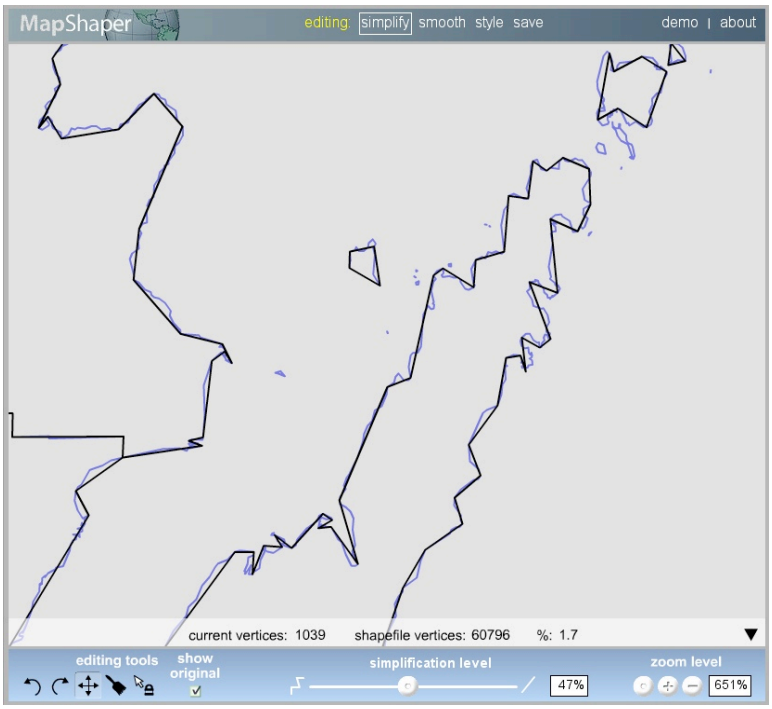

Figure 1: The MapShaper interface, showing both the original linework (blue) and the simplified linework (black) along with the amount of data reduction. The GUI was designed to maximize the amount of space for the map and minimize the complexity of the interface.

One of our key goals was to create a system that was fast enough to allow for real-time visual editing of even large geographic files, as map design is a creative and visual process. This was

achieved through a mix of server-side and client-side processing as well as optimization of data formats. Another goal was to create a service that would be free to use and transparent enough that even novice mapmakers would feel comfortable. In fact, the occasional or novice mapmaker was who we felt would most benefit from this service, since they may lack either the knowledge (*why*) or commercial software (*how*) to effectively generalize their digital data. In this regard, MapShaper was built in the same spirit as the "Cartable" system (Hubert and Raus 2002), which also presents a user-friendly, web-based "front end" that hides much of the computational complexity from users, so they are free to focus on the visual design of their maps.

This paper will demonstrate both the MapShaper application and the underlying code/architecture that was developed to support our goals. The biggest technical challenges to date have been:

1. MapShaper uses a combination of server-side preprocessing and efficient client-side line-simplification algorithms to achieve the fastest possible performance, as well as a degree of performance stability from user to user since much of the computational work is done on our servers. Determining the optimal division of labor between the various components required some experimentation and benchmarking of performance. By identifying bottlenecks and rewriting performance-critical code in C++, processing time has improved by two orders of magnitude.

2. MapShaper efficiently converts the Shapefile format into a topological data format that can support editing of polygon layers without introducing slivers, gaps, or overlaps between adjacent polygons.

3. MapShaper stores vector data in a compact format so that large Shapefiles can be transferred quickly over the Internet and processed easily at the user's end (i.e., works well on low-end hardware).

4. Building support for three kinds of line simplification into Flash: (i) Douglas-Peucker (1973), (ii) Visvalingham-Wyatt (1993), and (iii) Weighted Visvalingham (Zhou and Jones 2004).

5. Creating an efficient "history" system that supports unlimited undos.

6. Support for two kinds of line smoothing, because users may wish to export their work to either GIS or illustration software: (i) conversion of straight polyline segments to mathematically defined continuous curves (e.g., Bézier curves) that can be understood by programs such as Adobe Illustrator, or (ii) the displacement of vertices in polylines to approximate smooth curves (for output in the Shapefile format).

The rest of this paper is a report on the first three of these problems and how we developed solutions to each. For a more general introduction to MapShaper we encourage the reader to see Harrower and Bloch (2006) or to visit the website itself and see the system in action (www.mapshaper.org).

2. GENERAL SYSTEM ARCHITECTURE

MapShaper is not a conventional single-user software application. Rather, it is a web application consisting of a client interface that runs in a user's web browser and a collection of services running on a central server (Figure 2). Developing cartographic software using the web-services model introduces a set of interrelated challenges. Two central challenges that we confronted in developing MapShaper were: (1) how to efficiently exchange data between the server and client and (2) how to perform as much of the computational "heavy lifting" as possible on the server side.

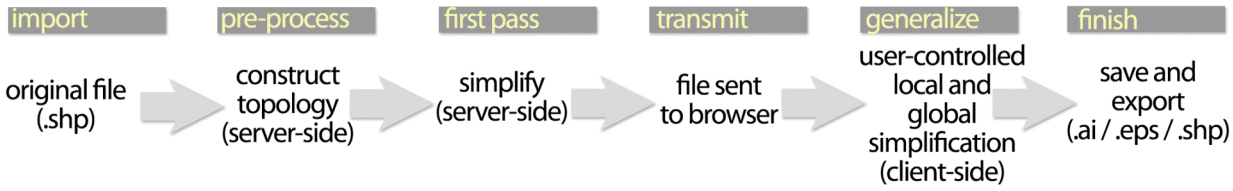

Figure 2: MapShaper does much of the computational "heavy lifting" (such as constructing topology) on the server to speed-up both transmission and client-side performance in Flash. The amount of pre-processing is driven by the user's end needs: Will they use their map as a simple web graphic or a high-resolution printed map?

Mapshaper's approach to Shapefile simplification involves two relatively intensive computational tasks, both of which are handled on the server. First, in order to edit polygon data, MapShaper must convert Shapefile polygons into a topological data model. The Shapefile format encodes polygons as polyline cycles, or rings of points. Shapefiles do not explicitly encode shared boundaries between adjacent polygons. MapServer detects shared polygon boundaries and converts them into single vector lines (polyline arcs) using an efficient hash-table algorithm developed by us. (Our method runs approximately in O(n) time, details are available on request). Topological conversion avoids the problem of creating overlaps and gaps (or "slivers") between adjacent boundaries, as would occur if adjacent polygons were simplified independently. Performance measurements for MapShaper's border-detection routine are shown in Table 1 below.

	Wisconsin Counties	Oregon Zipcodes	World Countries
Number of Vertices	60796	159681	403268
Number of Polygons	72	790	3784
Build Topology	60 msec	70 msec	160 msec
Compute Douglas-Peucker	120 msec	220 msec	490 msec
Compute Visvalingham-Wyatt	40 msec	60 msec	90 msec

* Tests were run on a 1300 MHz Pentium M computer with 1.5G RAM.

Table 1: Performance tests: Average time in milliseconds to complete core MapShaper functions.

The second major server-side task is the simplification of topologically converted vector lines. Due to the inherent performance limitations of the Flash web client, it was not feasible to perform vector-line simplification on the client. We met this challenge by designing a method for pre-computing simplification data and storing the results in a compact data structure that could be sent quickly to the client, such that a modified version of a map layer could easily be generated at any degree of simplification. Our solution is described below.

To understand how MapServer simplifies linework, it may be useful to review the basic concept of polyline simplification. Simplification is typically defined as the process of reducing the complexity of a polyline by eliminating a subset of vertices from the line. One broad class of simplification algorithms determines whether each point in a polyline should be discarded according to a metric that quantifies the amount of error or deviation from the original line that the point's removal would incur. Setting the degree of line simplification is equivalent to removing all vertices whose elimination results in less than a given amount of error. This broad category of simplification algorithms includes both top-down algorithms, such as Douglas-Peucker, and bottom-up algorithms, such as Visvalingham-Whyatt.

In MapShaper, a server-side program scans the vector lines in a map layer and calculates the error threshold at which each point drops out of its associated line. With the exception of the two endpoints, each point in a polyline becomes associated with a particular threshold. Threshold data is stored in a parallel array to the arrays containing x- and y-coordinate data. On the client-side, when a user manipulates a slider bar to set the degree of simplification, the MapShaper client translates the slider bar setting to an error threshold and redraws the layer at the new level of simplification. Only those vertices whose removal would exceed the error threshold are drawn. Array scanning is fast, so changes in the level of simplification appear almost instantaneous. In most instances, the speed at which simplification occurs is limited only by how fast the Flash client can redraw the vector lines in a map layer. This approach makes possible real-time, interactive line simplification.

3. SIMPLIFICATION ALGORITHMS

Currently, MapShaper supports three simplification methods (Figure 3): Douglas-Peucker, Visvalingham-Whyatt, and a modified version of Visvalingham-Whyatt (Zhou and Jones 2004) that is designed to filter out certain kinds of small-scale detail.

The Douglas-Peucker (DP) algorithm is a venerable workhorse that is still in common use in cartographic and GIS systems. It is a top-down algorithm with a simple implementation and adequate performance on our test data. Even on relatively large polygon layers (~500,000 vertices), execution time is well under one second using standard computer hardware (see Table 1). Various modifications to the basic algorithm have been proposed: A performance optimization using convex hulls improves performance to O(nlogn) for all input, with output that is identical to the original implementation (classic DP is O(n2), worst-case, when run to completion) (Hershberger and Snoeyink, 1992). We judged the performance of the simple implementation to be adequate. Other modifications have been proposed to prevent polyline self-intersections, a possible avenue for future development (Wu and Marquez 2003).

DP uses the maximum linear displacement of simplified line segments from the original polyline as an error metric. One consequence of DP's fixation on minimizing linear displacement is a tendency to retain protruding features. The effect has been characterized as being overly "spiky," prone to self-intersections and not ideal for general cartographic output (Dutton 1999). As an alternative to DP, we selected the Visvalingham-Whyatt (VW) algorithm. Instead of linear displacement, VW uses areal deviation to quantify the effect of vertex removal. Specifically, areal deviation is quantified as the area of the triangle formed by a vertex and its two neighbors in a polyline. Simplification is an iterative process consisting of removing the vertex with the smallest associated areal deviation from a polyline, recalculating the area associated with the two vertices adjacent to the removed vertex, then repeating the removal process until the desired level of simplification is attained. (Visvalingham and Whyatt 1993) Our implementation of VW runs in O(nlogn) time, where *n* is the number of vertices in a polyline. Performance on several sample Shapefiles is shown in Table 1. Subjectively, VW tends to produce linework that has less of the "spikiness" associated with DP.

The third simplification method in MapShaper's current repertoire is a modified version of VW that is intended to create linework with an even less jagged appearance. As described above, classic VW uses triangle area to quantify deviation. VW can easily be modified to use a combination of area and other properties of triangles, such as "flatness," "skewness," and "convexity" (Zhou and Jones 2004). We used an error metric that weights the area of triangles using the angle formed by the two line segments that are adjacent to each polyline vertex. Given two triangles of equal area, the triangle with a more acute angle will be removed first. Our modified area metric results in polylines that are smoother than those produced by standard VW.

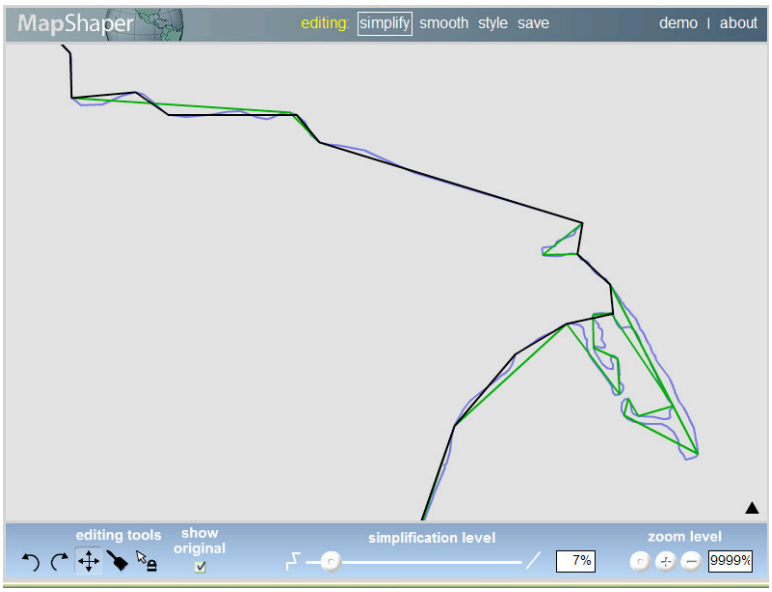

Figure 3: A comparison of DP simplification (green), modified VW simplification (black) and the original linework (blue). Note that at this level of simplification, modified VW has filtered out the narrow spits of land, while DP has created spikes and a self-intersection. Modified VW has retained more of the undulating detail in the upper left section of the screen.

Many additional polyline simplification methods can be adapted to work with MapShaper. With its current design, MapShaper is compatible with any simplification method that meets the following criteria: (1) The method must simplify lines using vertex removal, (2) retained vertexes may not be shifted from their original positions, and (3) simplification must be progressive—when vertices are dropped as the degree of simplification is increased, they may not be re-introduced into the polyline at higher levels of simplification.

4. DATA TRANSMISSION VIA FLASH REMOTING

The problem of server-client communication rounds out this discussion of core MapShaper techniques. The challenge we faced was how to transfer bundles of map-layer data comprising coordinate data, layer metadata, and simplification data from the server to the Flash client, and how to send information about modifications to the layer back to the server for output.

We chose Flash Remoting as our method for sending data between MapServer's server-side processes and the Flash client. Flash Remoting, developed by Macromedia to support passing data in and out of Flash applications, affords a number of advantages over the major alternative, XML. First, remoting uses a binary format called AMF ("Action Message Format") that is more compact than XML, so map-layer data travels more quickly between server and client over the Internet. Second, the time needed to convert AMF messages into a usable form inside the Flash client ("deserialization") is also much less than the equivalent process using XML. Finally, serialization and deserialization are performed automatically using Flash Remoting, unlike with XML, making life easier for the software developer. (Information about Flash Remoting can be found at http://www.macromedia.com.)

There are a number of available implementations of server-side Flash Remoting gateways for linking Flash applications to server-side programs. For MapShaper, we chose the open-source AMFPHP gateway (http://www.amfphp.org), which is stable, free, and has acceptable performance.

Flash Remoting provided an effective communications channel but did not entirely solve the problem of how to import layer data. An early version of MapShaper imported coordinate and simplification data in the form of arrays of floating-point numbers. We soon found that Flash's implementation of the array data structure consumed excessive amounts of computer memory, seriously impacting client performance when working with relatively large Shapefiles (>50,000 points).

We found a two-part solution. First, we transformed coordinate data from the original 64-bit floating-point numbers of the Shapefile format to 16-bit integers. This transformation reduced the quantity of data to be transmitted, and, more importantly, it enabled us to represent arrays of coordinate data as Unicode strings, which in Flash consume a fraction of the memory of arrays of equivalent length. These two modifications allowed MapShaper to import much larger Shapefiles than originally possible.

It should be emphasized that the technique of encoding array data as Unicode strings is not intended as a solution with general applicability. Rather, it has the character of a hack for overcoming the limitations of one particular application platform, in this case Macromedia Flash. Furthermore, the solution entails several tradeoffs. First and foremost is the reduction in precision incurred by converting double-precision floating-point numbers to 16-bit integers, analogous to rounding a decimal number to the nearest integer. The effect of the loss in precision is to shift the location of polyline vertices away from their original position. The amount of deviation varies from one vertex to another; our concern was to ensure that the maximum deviation remained within acceptable limits. While 16-bit precision would likely be inadequate for analytical GIS applications, visual editing is inherently imprecise. One way to visualize the practical effect of 16-bit integer transformation is to imagine using MapShaper to edit a Shapefile with the browser window set to a width of 1000 pixels. If the user is working at the 5000% zoom level, the maximum positional inaccuracy is still less than a single pixel. We considered this degree of imprecision to be acceptable.

5. DISCUSSION

The current version of MapShaper is already a working cartographic tool; nevertheless, considerable potential exists for future development. We can imagine enhancements and improvements both on the side of interface and workflow and on the side of algorithms and basic functionality. In keeping with this paper's emphasis on technical aspects of MapShaper, this portion of the paper will suggest possible improvements to MapShaper's simplification methods and other core functionality.

One avenue for improvement would be to expand MapShaper's repertoire of simplification and smoothing algorithms. As discussed above, MapShaper is designed to work with any polyline simplification method that uses progressive point removal. An expanded repertoire might include modified versions of the DP and VW algorithms for such purposes as avoiding polyline intersections and collisions or for retaining or filtering out certain kinds of detail. Additional algorithms might be optimized for generalizing certain classes of feature, for example building footprints or right-angle boundaries. We can imagine further generalizing MapShaper's polyline simplification framework to support algorithms that combine vertex removal and local vertex repositioning, such as the method developed by Garland and Zhou based on the quadric error metric (Garland and Zhou 2005).

Another possible direction for future development is to support concurrent editing of multiple polyline and polygon layers. This functionality might include enforcement of topological rules governing relationships between features on multiple layers. For example, polyline segments from a rivers layer could be linked to coinciding portions of a borders layer, such that modifications to one layer would be propagated to the associated layer (e.g., Ware et al. 2003)

Dynamic client-side line generalization has applications beyond interactive map-layer editing. It is an important step in the direction of creating web maps that display appropriately generalized linework at a continuous range of scales. Web maps could adjust their degree of generalization

dynamically to suit a range of output devices. Examples include resolution-independent web maps that reveal more or less detail according to the pixel density of the viewer's monitor or web maps that can be printed at a higher level of detail than what appears onscreen. Fast client-side line generalization could also be used to implement smooth, continuous zooming in web maps, with lines that gradually increase or decrease in detail as the scale of the map changes.

6. CONCLUSIONS

We believe that MapShaper points the way to a new class of cartographic tools that are web-based, interactive, and accessible to non-specialists. In designing Mapshaper, we have tried to strike a balance between automation and manual editing. MapShaper bridges several gaps within the field of cartography: the gap between source data and finished maps, the gap between academic cartography and non-specialist mapmakers, and the gap between automated computer cartography and graphic design. From the user's point of view, Mapshaper's most important innovation is its ability to generalize map linework interactively. Unlike most other cartographic software tools, simplification occurs in real time, as fast as the cartographer can move a slider bar or click on vertices. In order to provide this level of functionality, we overcame several technical challenges. This paper describes our solutions to three of these problems: (1) how to store the results of polyline simplification such that modified versions of a polyline can be rapidly generated at any degree of simplification; (2) how to apply this technique to a selection of simplification algorithms; and (3) how to pass simplification data and coordinate data rapidly between a web client and processes running on a remote server. Because Mapshaper is a rolling release, new functionality is being added as it is developed. The public is encouraged to visit the website (mapshaper.org) and give us feedback.

7. REFERENCES

Bader, M. (2001). *Energy Minimization Methods for Feature Displacement in Map Generalization.* Ph.D. thesis, Department of Geography, University of Zurich.

Bédard, Y., and E. Bernier (2002). Supporting Multiple Representations with Spatial View Management and the Concept of "VUEL". In *Proceedings of the Joint Workshop on Multi-Scale Representations of Spatial Data, ISPRS WG IV/3,* ICA Commission on Map Generalization (Ottawa, Canada), July 7th-8th.

Burghardt, D., M. Neun, and R. Weibel (2005). Generalization Services on the Web – A classification and an initial prototype implementation. *Cartography and Geographic Information Science*, Vol 32(4): 257-268 (12).

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

Dutton, G. (1999). Scale, sinuosity, and point selection in digital line generalization. *Cartography and Geographic Information Science*, Vol 26(1): 33-53.

Galanda M. and R. Weibel (2003). Using an Energy Minimization Technique for Polygon Generalization, *Cartography and Geographic Information Science,* Vol 30(3): 259-275.

Garland M. and Zhou Y (2005). Quadric-based simplification in any dimension. *ACM Transactions on Graphics* Vol 24(2): 209-239.

Hardy, P., M. Hayles, and P. Revell (2003). Clarity – A new environment for generalization using agents, java, xml and topology. Proceedings of the *ICA Generalization Workshop*, Paris, April 2003.

Harrower, M. and M. Bloch (2006). MapShaper.org: A Map Generalization Web Service. *IEEE Computer Graphics and Applications* (CG&A) Special Issue on Geographic Visualization. July / August 2006.

John Hershberger, J and J. Snoeyink (1992). Speeding up the Douglas-Peucker line-simplification algorithm. *Proc. 5th Intl. Symp. on Spatial Data Handling*, Vol 1: (134–143).

Hubert, F. and A. Raus (2002). A method based on samples to capture user needs for generalization. Proceedings of the *ICA Generalization Workshop*, Paris, April 2003.

McMaster, R. B. and S. Shea (1992). *Generalization in Digital Cartography*, The Association of American Geographers, Washington D.C., 1992.

Mackaness, W. and G. Edwards (2002). The importance of modeling patern and structure in automated map generalization. Proceedings of the *Joint Workshop on Multi-Scale Representations of Spatial Data*, Ottawa, Canada, July 7th-8th, 2002. CCRS – Canada Centre for Remote Sending, Ottawa, Canada.

Ormsby, D. and W. Mackaness (1999). The development of phenomenological generalization within an object-oritneted paradigm. *Cartography and Geographic Information Science*, Vol 26 (1): 70-80.

Visvalingham, M., and D. Whyatt (1993). Line generalization by repeated elimination of points, *The Cartographic Journal* 30(1): 46-51.

Visvalingam, M. and P. J. Williamson (1995). Simplification and generalization of large scale data for roads. *Cartography and Geographic Information Science* 22(4): 3-15.

Ware, J. M., C. B. Jones, and N. Thomas (2003). Automated map generalization with multiple operators: A simulated annealing approach. *International Journal of Geographical Information Systems*, 17 (8): 743-769.

Wu, S.-T., M. R. G. Márquez (2003). A non-self-intersection Douglas-Peucker Algorithm. Proceedings *Brazilian Symposium on Computer Graphics and Image Processing* (SIBGRAPHI XVI) 2003.

Zhou S and C.B. Jones (2004) Shape-Aware Line Generalisation With Weighted Effective Area. In Fisher, Peter F. (Ed.) *Developments in Spatial Data Handling 11th International Symposium on Spatial Data Handling*. Springer, pp 369-380.