

Web-client Based Distributed Generalization and Geoprocessing

Eric B. Wolf

US Geological Survey

Center of Excellence in GIScience

Denver, Colorado, USA

ebwolf@usgs.gov

Kevin Howe

US Geological Survey

Center of Excellence in GIScience

Rolla, Missouri, USA

khowe@usgs.gov

Abstract— Generalization and geoprocessing operations on geospatial information were once the domain of complex software running on high-performance workstations. Currently, these computationally intensive processes are the domain of desktop applications. Recent efforts have been made to move geoprocessing operations server-side in a distributed, web accessible environment. This paper initiates research into portable client-side generalization and geoprocessing operations as part of a larger effort in user-centered design for the US Geological Survey's *The National Map*. An implementation of the Ramer-Douglas-Peucker (RDP) line simplification algorithm was created in the open source OpenLayers geoweb client. This algorithm implementation was benchmarked using differing data structures and browser platforms. The implementation and results of the benchmarks are discussed in the general context of client-side geoprocessing. (*Abstract*)

Keywords- distributed geoprocessing; generalization; cyberinfrastructure; AJAX (key words)

I. INTRODUCTION

The past few years have been witness to a rapid increase in web-based mapping applications demonstrated by the popularity of services like Google Maps, Microsoft Live Earth and MapQuest. Standardization efforts of the Open Geospatial Consortium (OGC) provide open methods for sharing geospatial information over the web similar to the Application Programming Interfaces (APIs) provided by Google and Microsoft. These open standards combined with recent advances in client-side programming architectures set the groundwork for creating rich, browser-based mapping applications.

Geoprocessing has traditionally been the domain of complex software running on high-end workstations. In the past decade, the advent of desktop Geographic Information Systems (GIS) has allowed greater access to the analytical and representational power of geoprocessing. A full-featured GIS, however, is still well beyond the means of the average user. Most widely disseminated geoprocessing techniques require expensive desktop GIS software packages. The high cost associated with access to geoprocessing results creates a “geospatial divide” [1]. Three possible ways to address this digital divide are: freely

available open source GIS packages, server-side geoprocessing services and client-side geoprocessing.

The User-Friendly Desktop Internet GIS (uDIG) [2] and Open Java Unified Mapping Platform (OpenJUMP) [3] are representative of recent attempts to address the accessibility of geoprocessing through freely available open source GIS packages. Both uDIG and OpenJUMP implement a compelling set of geoprocessing features but their general purpose nature make them daunting to use by the average user. Deploying one of these packages for a specific analysis application can be challenging. The end-user would need to be able to maintain the appropriate version of the software package and navigate the general purpose interface.

Distributed web-based mapping forms another framework for accessible geoprocessing applications. Complex operations can be executed on a server or middleware architecture based on OGC specifications compliant servers such Geoserver [4] or ESRI ArcGIS Server [5]. Results of the geoprocessing operations are provided to the end-user via OGC standardized Web Map Service (WMS) or Web Feature Service (WFS) specifications or ESRI Internet Mapping Service (IMS) [6]. The recent OGC Web Processing Service (WPS) Interface Standard provides rules for the inputs and outputs of such a server-side geoprocessing framework [7]. Past explorations of this technique include Chow's [8] review of the various commercial web-mapping APIs available in the context of visualizing the results of this type of geoprocessing framework. Burghardt et al. [9], created an automated generalization service built using the JUMP library running as an Apache Tomcat service. Bloch and Harrower [10] created a vector geoprocessing service for line generalization called MapShaper.org which utilizes a client written in Macromedia (now Adobe) Flash to communicate with a server-side generalization application. More recently, Foerster and Stoter [11] demonstrated a geoprocessing service by extending a OGC standard server package. Michaelis and Ames [12] evaluated the new OGC WPS specification using an open source GIS application to connection to a server-side geoprocessing service.

Client-side aspects of this distributed architecture have also been explored in published reports. Sayar et al. [13] discussed a framework for accessing server-side

geoprocessing services through AJAX. Yuan and Qiuming [14] similarly demonstrated a hydrological geoprocessing model utilizing the Google Maps API for visualizing results of processing services written as middle-tier scripting in JavaScript.

None of these published studies have explored the feasibility of implementing geoprocessing within the browser. The ability to execute and display geoprocessing within the browser addresses some interesting issues: server and network loads, accessibility, and data conflation issues. In a server-side geoprocessing system, data must be uploaded to the server where operations are completed and results returned. In a client-side implementation, only the initial base data might need to be downloaded from a server. A client-side geoprocessing application can be distributed like a static hypertext markup language (HTML) file, eliminating the need for servers capable of running geoprocessing scripts. Finally, many geoprocessing techniques cannot run independently on different data sets. For instance, if “mash-up” combines road and river features accessed from different server-side generalization services, topological errors can be introduced between the features: due to conflation, the road and the river may no longer meet at the bridge [15].

The National Map is the US Geological Survey (USGS) vehicle for providing authoritative data content that has broad application within and beyond USGS. Asynchronous JavaScript and XML (AJAX) may be utilized for client-side geoprocessing, enhancing the user experience of the base map functions provided by *The National Map* [16].

AJAX provides a client-side framework for creating rich client-side applications that is supported by all major web browsers. This framework combines the JavaScript language, now standardized as ECMAScript [17], with strong support for creating Simple Object Access Protocol(SOAP) XML server requests and parsing responses. Scholten et al. [18] provide a framework for the performance evaluation of geoprocessing in OGC standards based spatial data infrastructures.. Smullen and Smullen [19] demonstrated significant savings in transmission time for applications utilizing AJAX over HTML in a database driven, non-computationally intensive application. Chang et al., [20] further examined the performance of SOAP and AJAX within the context of server-side geoprocessing services.

Most recent research utilizes AJAX to either reduce the number and size of server data requests or provide richer integration for visualization. This research, however, demonstrates the potential for reducing the burden on the server by utilizing AJAX to offload computationally intensive geoprocessing operations onto the client. Besides releasing the server from heavy processor loads, pure client-side geoprocessing can also inform and expedite geospatial decision making without requiring complex server-side implementations. As Nash et al. [21] demonstrated, “a great advantage of AJAX is that no browser plug-in is required.” The geoprocessing algorithms are downloaded like a static web page guaranteeing that the code version executed on the client is always what is intended by the application designer.

This paper attempts to parameterize the real-world capabilities of AJAX for geoprocessing tasks. To accomplish that task, we implemented a standard generalization algorithm in OpenLayers and ran a series of benchmarks on a variety of web browsers. OpenLayers is a popular open source AJAX application for creating independent “mash-up” web-mapping applications. OpenLayers can access data provided through OGC WMS and WFS protocols and utilize Google Maps, Microsoft Live Maps and Open Street Map as base maps.

II. THE RDP GENERALIZATION ALGORITHM

Basemaps provided by Google, Microsoft and Open Street Map currently provide multiple levels of detail. These levels of detail are pre-calculated and provided to the client based on the “zoom” or scale of information requested. The OGC WFS only stores the single geometric representation of a feature. If a mash-up application combines this single representation from WFS, it may be necessary to simplify the geometry to accommodate changes in scale (zooming out).

The Ramer-Douglas-Peucker (RDP) line simplification algorithm was developed independently and simultaneously by Ramer [22] and Douglas and Peucker [23]. RDP is a top-down point removal algorithm that starts with a base line segment connecting the nodes (first and last vertices in a polyline segment). The most distant vertex from this base line is determined and two new segments are drawn from the original two vertices to this vertex. The process is repeated until the distance from the approximating line to the original line is below a tolerance level. The basic implementation has a worst case complexity of $O(n^2)$. Published reports provide modifications of RDP to preserve topology and speed up the algorithm [15, 24]. By executing RDP to the fullest extent, preserving levels of detail through a tree structure, Buttenfield demonstrated progressive transfer of vector information through a distributed architecture including a stand-alone Java client [25]. Expanding on that research, Buttenfield and Wolf utilized the RDP algorithm coupled with Saalfeld’s topology corrections to preserve feature topology across feature classes [26].

Typically, the vanilla RDP algorithm quits as soon as the tolerance level has been reached. By sacrificing some speed for reusability, our implementation pre-calculates the tolerance threshold for every node, making it possible for the pre-renderer to cull the insignificant nodes in a single pass, without having to run the algorithm every time the tolerance is changed.

For this study, the vanilla $O(n^2)$ RDP algorithm was coded in ECMAScript used by OpenLayers to provide on-the-fly client-side generalization. OpenLayers forms OGC WFS requests via SOAP and receives vector data as Geographic Markup Language (GML). Within OpenLayers, the GML is converted to Scalable Vector Graphics (SVG), an XML standard for describing vector information. The SVG is then parsed by the browser and drawn to the screen as an overlay. The conversion from GML to SVG occurs on every display refresh, allowing geoprocessing tasks, like our

implementation of the RDP algorithm, to easily “hook” into the feature rendering pipeline.

III. IMPLEMENTATION CONSIDERATIONS

Several variations of the RDP algorithm were implemented in order to explore the computational environment provided by ECMAScript. The lightweight nature of ECMAScript poses some significant challenges to the implementation of geoprocessing algorithms. Our initial implementation of the RDP algorithm used recursive functions, reflecting the naturally recursive nature of the RDP algorithm. However, the ECMAScript interpreters in the web browsers we tested routinely experienced stack overflow. Based on these experiences, we concluded that current browser implementations do not handle deep recursion well, and suffer from stack overflow too often to be reliable on real world datasets. Our implementation of the RDP algorithm avoids all recursion, replacing the stack with data structures to keep track of the processed and unprocessed elements.

The built-in data structures in ECMAScript also favor light weight applications. Depending on the implementation, arrays can be treated either traditionally as a linearly dense set of data, or as a sort of associative array. The associative array has the advantage of efficiently manipulating sparse datasets, if the index of the elements is known, and traversal of the dataset, so long as the order of elements traversed is unimportant. However, this type of array is not always efficiently handled by the current generation of browsers. The cost of accessing an element, while small, is $O(m)$, m being the number of elements physically in the array, since order of the indices is not maintained. This could potentially become an issue when manipulating huge datasets. Furthermore, the built-in array methods tend to treat arrays as dense, and run in $O(n)$ time, n being the value of the largest index + 1. The interpreter decides automatically which type of array to use, though the type can be forced by maintaining the invariants for that particular type.

Two versions of our implementation of the RDP algorithm were tested using two different data structures. The first was a pair of linked lists, separating the evaluated and unevaluated nodes. The advantage of this arrangement was that endpoint nodes could be located without iteration in

TABLE I. BROWSER VERSIONS TESTED.

Browser	Version	Build
Mozilla Firefox	2.0.0.16	1.8.1.14
	3.0.1	1.9.0.1
	3.1	1.9.1b1pre
Microsoft Internet Explorer	7.0	5730.11
	8.0 beta 2	6001.18241
Apple Safari	3.1.2	525.21
Opera	9.5.2	10108
Google Chrome	0.2.149.29	1798

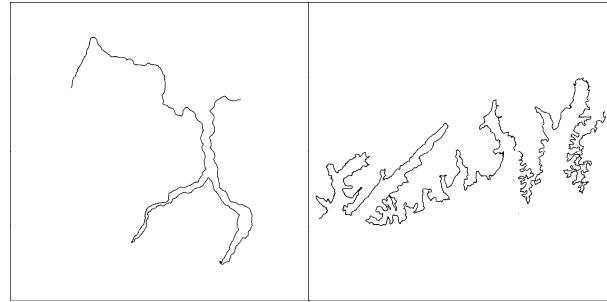


Figure 1: Fjords (a, 2711 vertices in a single feature) and (b, average 23.79 vertices/feature in 535 features)

$O(1)$ time. This version closely resembles an unrolled binary tree: each node in the outer linked list stores an inner linked list containing the unevaluated vertices from the representation.

The second version uses a table of indices to monitor which nodes remain uncommitted. Reordering is unnecessary, since the thresholds are attached directly to its corresponding node. Originally this version used an associative array to keep track of endpoint nodes, but the inability to maintain the order of the nodes during traversal without using costly sorting functions forced us to find a different approach. In the new version each group of nodes

is traversed twice per iteration: once to locate the endpoints and again to calculate distances from these endpoints. While this version has a higher complexity than the linked lists, it has less overhead and uses the faster, built-in array methods.

Benchmarks were conducted using an Intel Core 2 Duo CPU operating at 2.0 Ghz running Microsoft Windows XP SP3. All unnecessary background services were disabled.

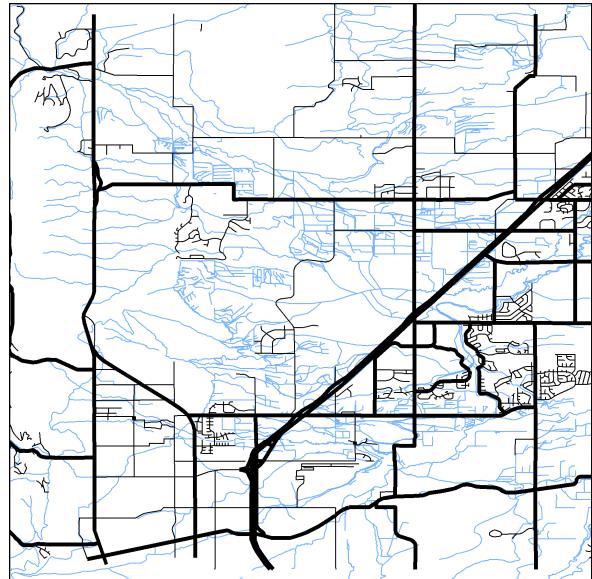


Figure 2: Roads (average 6.32 vertices/feature in 1128 features) and streams (average 17.95 vertices/feature in 12730 features)

Command-line scripts were used to launch each browser with the same, high priority in order to create a more consistent execution environment. The RDP benchmark was run on the current stable version of several mainstream web browsers and two prerelease versions (see Table 1).

Four datasets were utilized in the benchmarks. The complexity of the RDP algorithm is based on the number of vertices in a single feature. Two datasets representing the coastline of two fjords were chosen to provide degenerate cases test the operational limits of the RDP algorithm (see Figure 1). The average user will not encounter features with such high vertex density as these datasets (2711 vertices per feature and 23.79 vertices per feature). Fjord (a) is represented as a single feature comprised of 2711 vertices. Fjord (b) is broken into 535 features with 12730 vertices. Two datasets were selected representing the road and stream network in downtown Boulder, Colorado (see Figure 2). The road network comprises 1128 features with 7131 vertices and the stream network comprises 2969 features with 53300 vertices. These datasets have lower vertex density (6.32 vertices per feature and 17.95 vertices per feature). Since many web-mapping applications display road network data and fewer display stream data (and almost none display fjords), the road network dataset with it's lower vertex density and higher feature count would be considered most typical of the four datasets tested.

IV. RESULTS

The overall goal of the benchmarks was to demonstrate the capabilities of the AJAX framework for geoprocessing. While the RDP algorithm does have a complexity of $O(n^2)$,

in practice the rate of growth is very slow, and almost linear. Geographic representations tend to be fairly smooth with relatively few vertices per segment compared to the degenerate cases provided by the fjords. A line segment forcing $O(n^2)$ behavior in RDP would likely not be representative of any real feature. The test datasets used in these benchmarks were selected to both provide a worst case scenario and represent more realistic features. Vector geoprocessing algorithms, like the RDP, generally encounter large numbers of relatively simple features.

Overall performance was best for Google Chrome and FireFox 3.1. Both of these browsers were introduced as early test versions just before the completion of this study. The poor results for Internet Explorer are likely due to Microsoft's focus on its own client-based development architecture, Silverlight.

The benchmarks demonstrate poor performance for array access in older ECMAScript interpreters. For these older interpreters, a linked-list data structure was considerably faster for the more complex datasets. Newer browsers with enhanced ECMAScript interpreters like "TraceMonkey" from FireFox and "V8" Google, as well as Apple Safari, appeared to address this issue. The linked-list implementation failed to execute on multi-feature datasets in both Microsoft Internet Explorer 7 and 8.

Interestingly, generalization of the fjord (a) dataset was faster than the average time required to generalize the streams dataset. This implies that the overhead of processing multiple features was greater than the time requirements imposed by the complexity of the algorithm.

All of the tested browsers were able to generalize the road features very quickly. This is significant because the road dataset, encoded in XML, comprises a 515Kb SOAP

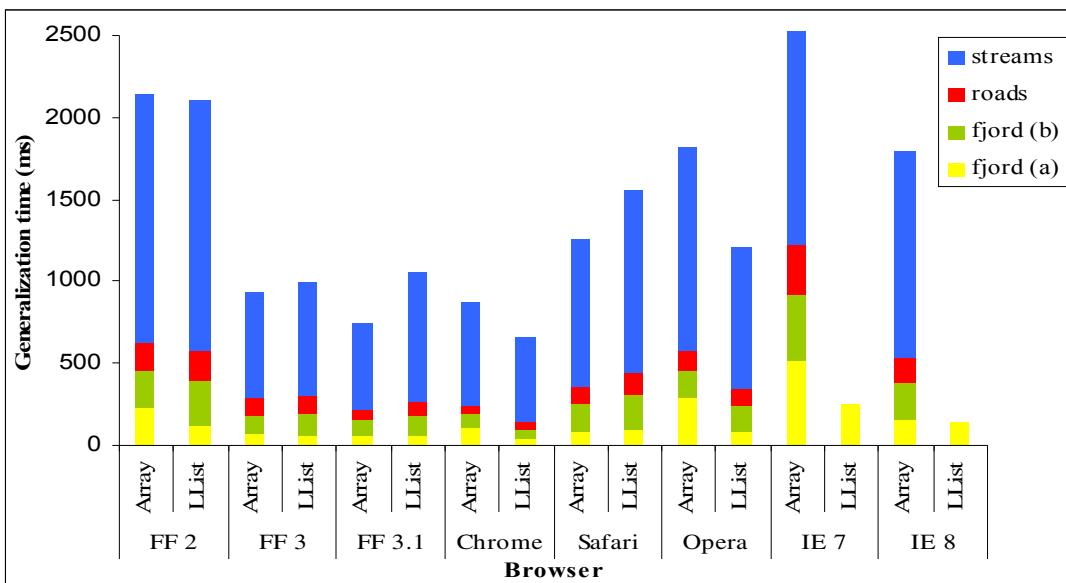


Figure 3: Benchmark results in features generalized per second

response. The WFS protocol requires a SOAP request and response for each client transitions from larger to smaller scale (i.e., during “zoom out” operations). These SOAP requests and responses can be avoided with the ability to generalize data in the client for display at coarser resolutions. In the worst case, Internet Explorer 7 required 830ms to generalize the road dataset and in the best case, Google Chrome took only 128ms.

V. CONCLUSIONS

AJAX provides a promising architecture for client-side geoprocessing. Current browsers present many challenges when implementing complex algorithms within this framework. Due to stack overflow concerns, algorithms should be implemented to avoid recursion. Linked-lists should be used for storing large data sets in place of built-in array types in performance in older browsers. The new, pre-release browsers include ECMAScript interpreters that are specifically designed to handle more substantial applications. Mozilla “TraceMonkey” ECMAScript interpreter brings recent research in programming language interpretation to the browser. Google Chrome’s V8 ECMAScript interpreter utilizes virtualization techniques to provide greater isolation of tasks. These ECMAScript implementations are considered to be the start of a new generation of interpreters that will make possible more complex geoprocessing techniques such as generalization with topology corrections and on-the-fly spatial reprojection.

The ability to move geoprocessing tasks from servers to the client, with ECMAScript embedded in the HTML content, can provide increased access to geoprocessed spatial information and inform the decision making process without the need for expensive software or complex server configurations. Client-side generalization does not interfere with user interaction compared to server-side generalization. The processing time for client-side generalization was shown to be similar to the packet latency many users experience on the internet. These techniques show promise in enhancing the end-user experience while reducing server load, making this technology significant to the future of *The National Map*.

VI. FUTURE WORK

Rigorous efforts should be made to determine the benefits and drawbacks of server-side versus client-side geoprocessing. Some geoprocessing methods may be better placed server-side while others may be better on the client. The parameters determining the best placement of geoprocessing methods need to be explored.

In the near future, more algorithms for generalization will be implemented within OpenLayers to create a set of common generalization methods including topology corrections in RDP. The WFS specification could be extended to pass information to the client as to which

generalization algorithm to use to generate smaller scale representations from the original data. The client could calculate levels of detail for smooth “zooming” or changes in scale.

In the distant future, it would be advantageous to develop a library of client-side geoprocessing methods, like the Java-based JUMP library that underlies the OpenJump GIS application. Such a library could be used within an AJAX-based geoweb client, like OpenLayers, to provide significant client-side geoprocessing capabilities.

ACKNOWLEDGMENT

This work forms a portion of the project “User-Centered Design for *The National Map*”, funded by the USGS Center of Excellence in GIS.

REFERENCES

- [1] C. C. Miller, "A Beast in the Field: The Google Maps Mashup as GIS/2," *Cartographica*, vol. 41, p. 12, 2006.
- [2] uDIG, "uDIG : Home," retrieved August 18, 2008, <http://udig.refractions.net/>.
- [3] OpenJUMP, "What is OpenJUMP?," retrieved August 18, 2008, <http://openjump.org/wiki/show/OpenJUMP>.
- [4] Geoserver, "What is Geoserver?," retrieved August 18, 2008, <http://geoserver.org/display/GEOS/What+is+Geoserver>.
- [5] ESRI, "ArcGIS Server: A Complete and Integrated Server GIS," retrieved August 18, 2008, <http://www.geographymatters.com/library/brochures/pdfs/arcgis-server.pdf>
- [6] OGC, "OpenGIS Web Services Architecture," Open Geospatial Consortium OGC 03-025, 2003.
- [7] OGC, "OpenGIS Web Processing Service (WPS) Specification," Open Geospatial Consortium OGC 05-007r7, 2008.
- [8] E. T. Chow, "The Potential of Maps APIs for Internet GIS Applications," *International Journal of Geographic Information Science*, vol. 12, p. 12, 2008.
- [9] D. Burghardt, M. Neun, and R. Weibel, "Generalization Services on the Web - Classification and an Initial Prototype Implementation," *Cartography and Geographic Information Science*, vol. 32, p. 11, 2005.
- [10] M. Block and M. Harrower, "MapShaper.org: A Map Generalization Web Service," in *AutoCarto*, Vancouver, Washington, 2006.
- [11] T. Foerster and J. Stoter, "Establishing an OGC Web Processing Service for generalization processes," in *ICA Commission on Map Generalization and Multiple Representations*, Vancouver, Washington, 2006.
- [12] C. Michaelis and D. Ames, "Evaluation and Implementation of the OGC Web Processing Service for Use in Client-side GIS," *GeoInformatica*, (In press), 2008.
- [13] A. Sayar, M. Pierce, and G. Fox, "Integrating AJAX Approach into GIS Visualization Web Services," in *AICT/ICIW*, Guadeloupe, French Caribbean, 2006, pp. 169-176.
- [14] Y. Yuan and C. Qiuming, "Integrating web-GIS and hydrological model: a case study with Google Maps and IHACRES in the Oak Ridges Moraine area, Southern Ontario, Canada," in *IGARSS*, Barcelona, Spain, 2007, pp. 4574-4577.
- [15] B. P. Buttenfield and E. Wolf, "The Road and the River Cross at the Bridge Problem: Internal and Relative Topology in an MRDB," in

ICA Commission on Generalisation and Multiple Representations,
Moscow, Russia, 2007.

- [16] USGS, "A Research Agenda for Geographic Information Science at the United States Geological Survey," National Research Council, Mapping Science Committee, Ed. 143, 2007.
- [17] ECMA, "Standard ECMA-262: ECMAScript language specification," 1999.
- [18] M. Scholten, R. Klamma, and C. Kiehle, "Evaluating Performance in Spatial Data Infrastructures," *IEEE Internet Computing*, vol. 10, p. 7, 2006.
- [19] C. W. Smullen and S. A. Smullen, "An Experimental Study of AJAX Application Performance," *Journal of Software*, vol. 3, p 30, 2008.
- [20] M. Chang, M. Bebenita, A. Yermolovich, A. Gal, and M. Franz, "Efficient Just-In-Time Execution of Dynamically Typed Languages Via Code Specialization Using Precise Runtime Type Inference," Donald Bren School of Information and Computer Science, University of California, Irvine, Irvine, CA Report No. 07-10, 2007.
- [21] E. Nash, P. Korduan, S. Abele, and G. Hobona, "Design Requirements for an AJAX and Web-Service Based Generic Internet GIS Client," in *Proceedings of the 11th AGILE International Conference on Geographic Information Science 2008*, University of Girona, Spain, 2008.
- [22] U. Ramer, "An Iterative Procedure for the Polygonal Approximation of Plane Curves," *Computer Graphics and Image Processing*, vol. 1, pp. 244-256, 1972.
- [23] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol. 10, pp. 112-122, 1973.
- [24] J. Hershberger and J. Snoeyink, "Speeding up the Douglas-Peucker line simplification algorithm," in *Proc. 5th Intl. Symp. Spatial Data Handling*, 1992, pp. 134-143.
- [25] B. P. Buttenfield, "Transmitting Vector Geospatial Data across the Internet," in *GIScience*, M. J. Egenhofer, Mark, D.M., Ed. Boulder, Colorado: Springer Berlin / Heidelberg, 2002, pp. 51-64.
- [26] A. Saalfeld, "Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm," *Cartography and Geographic Information Science*, vol. 26, 1999.