CLIENT-SIDE VERSUS SERVER-SIDE GEOPROCESSING

Benchmarking the performance of web browsers processing geospatial data using common GIS operations.

by

Erin L. Hamilton

A thesis submitted in partial fulfillment of

the requirements for the degree of

Master of Science

(Cartography and Geographic Information Systems)

at the

UNIVERSITY OF WISCONSIN–MADISON

2014

Acknowledgements

The completion of this thesis could not have been accomplished without the help and support of many people.

Thank you to Jim Burt for serving as my advisor. Your ability to see the big picture and ask hard questions made my work infinitely stronger. Your feedback on my drafts pushed me to produce my best work and I am thankful for that.

Thank you to Qunying Huang and David Hart for serving on my committee. Your feedback on this thesis was much appreciated. More importantly, the support you provided me in improving my programming skills over the past three years enabled me to execute the technical pieces of this research. I am extremely grateful for that.

Thank you to the members of the GIS group. The questions asked during my research progress presentations over the last year were significant in shaping and refining my work.

Big thank you to the people in the Cartography Laboratory. Your guidance, support, and friendship were critical in keeping me moving forward. I especially want to thank Tanya Buckingham and Daniel Huffman for providing me with so much insightful perspective and advice along the way and Bill Buckingham for providing tremendously helpful recommendations on my writing.

Thank you to my Mom and Dad for being my cheer team across the country. Your belief in my abilities kept me going. I could not have made it here without your support.

Most importantly, thanks to Bob Baddeley. You served as my rock through this whole process. I could not have done this without your guidance, patience, and reassurance through these years. Every step of the way you had a major role and I will never be able to thank you enough for that.

Abstract

Web-based GIS and mapping applications are traditionally based on a client-server model, where most of the data processing work is placed on the server, but current trends in web applications are moving towards more interactivity and processing tasks on the client. This study examines what happens when that processing load is shifted to the client using JavaScript to process geospatial data directly in the browser.

The time needed to complete common GIS tasks using the JavaScript library JSTS Topology Suite were benchmarked in popular web browsers including Chrome, Firefox, Internet Explorer, Opera, and Safari. The GIS operations buffer, union, and Voronoi diagram were tested with a suite of points, lines, and polygons ranging in size from 10 up to 100,000 vertices. The testing platforms included Windows, Mac, and Linux desktops and laptops.

The same geoprocessing tests were conducted on a cloud-based Linux server using the Java library JTS Topology Suite as a performance comparison of server-side processing. The various testing configurations were then analyzed to see how browsers compare to the performance of traditional client-server applications.

The results indicated that the server was faster than the client in all testing scenarios, and only when processing generalized small scale data, was the client performance in an acceptable time range. These results demonstrated that the current implementation of web browsers are limited in their ability to execute JavaScript geoprocessing and not yet prepared to process data sizes larger than about 7,000 to 10,000 vertices before either prompting an unresponsive script warning in the browser or potentially losing the interest of the user.

Table of Contents

1. Introduction

Visit the websites of most government organizations or news sites and one will likely find interactive geographic information systems (GIS) or mapping applications. Many of these applications aim to provide a browser based experience that is normally associated with that of a desktop, also known as a rich internet application (Kay 2009). Although the presence of these web applications is not new, the technology used to build them has been changing for nearly twenty years. One example of technology that was widely used for building web mapping rich internet applications in the last twenty years is Adobe Flash with Flex.

Purchased by Macromedia in 1996, Flash was originally developed for drawing and animating in the browser (Warren 2012). In 2004, Macromedia released Flex, a server-based application that seamlessly integrated with the front-end Flash technology, creating an unrivaled product for developing rich internet applications (Adobe 2012). Shortly thereafter, Adobe acquired Macromedia and released Flex 2.0, moving Flex away from a solely server-based environment and instead to a fully integrated development environment making it easier for developers to deploy a full rich internet application to a web server (Adobe 2012).

ESRI recognized the opportunity to use Flex in GIS application development and introduced the ESRI Flex API in 2008 (ArcGIS Server Development Team 2008). However, in 2010 Steve Jobs released a memo saying that Apple would no longer support Flash on mobile devices, delivering a huge blow to the popularity of the technology (Jobs 2010). ESRI published a similar statement that put into question their future support of the Flex API and instead encouraged developers to focus their efforts on using JavaScript and HTML5 (Powell 2014).

The withdrawal of support for Flash made way for HTML5 and JavaScript to dominate mobile and front-end web development. HTML5 is the latest release of the HyperText Markup Language (HTML) which is used to create web pages. HTML5 enabled the HTML/JavaScript combination to compete as a rich internet application technology by introducing seamless integration of multimedia, improved performance and support on a range of devices, including mobile, as well as support for vector graphics and animation (Mozilla Development Network 2014).

JavaScript is an object-oriented scripting language most commonly used for creating interactive client web applications within HTML and is supported by all major web browsers (Koch 2006). Improvements in general performance of browsers, including faster JavaScript interpreters, and an explosion of free and open source JavaScript libraries, has led to an influx of JavaScript-based web applications, including map-based applications that provide fast and smooth functionality in the browser.

Mapping software companies such as MapBox, Leaflet, CartoDB, and ESRI have introduced JavaScript coding libraries to build mapping user interfaces that exploit the interactive capabilities of the web browser. Generally, these libraries are meant to build applications for visualization of data in map form, but the CartoDB and ESRI libraries also provide geoprocessing methods for basic analysis and data manipulation capabilities through the browser interface. These two libraries are built to function in a client-server model where the request for the geoprocessing is made by the browser, but the processing work is executed on a server. However, as browsers become more powerful and richer JavaScript libraries are developed, it is reasonable to assume that developers will incorporate increasingly sophisticated tasks for browsers to execute, including those used in GIS analysis and data manipulation. This

trend is demonstrated by the recent availability of JavaScript geoprocessing libraries developed specifically for the client.

The most well-known and extensive of these client geoprocessing libraries is JSTS Topology Suite. JSTS is a JavaScript port of the Java library JTS Topology Suite, which is a large and robust application programming interface (API) of 2D spatial algorithms (Vivid Solutions 2013). JSTS was originally developed for inclusion in the web mapping library OpenLayers version 2.11, but is now offered as a stand-alone geoprocessing library (Hartell 2013). Another JavaScript port library is Shapely.js, which is a port of the Python geometry operations library Shapely (Helm 2013). However, this library is not nearly as well supported or extensively ported as JSTS.

A few other JavaScript geoprocessing libraries deserving mention are Njord.js, a lightweight alternative to JSTS (Sveen 2014); jQuery Geo, a jQuery plugin that provides both mapping capabilities along with operations similar to those in JTS (Westphal 2014); and Turf.js, another geoprocessing library that relies on JSTS (Herlocker 2014). All of these libraries appeared within the last several years and continue to be updated, but their successful incorporation and use in client GIS and mapping applications has yet to be seen.

The goal of this research is to assess the viability of this trend towards JavaScript-based client mapping applications that provide geoprocessing directly in the browser. Understanding the limits of browsers is important for website developers when determining how to structure the processing tasks between the client and the server. To test this, the performance of browsers executing geoprocessing is compared to the traditional architecture of server-side geoprocessing. If it is more efficient to perform processing work in the browser than on the server, then it could save developers time and money from purchasing and maintaining a server.

The layout of this thesis will begin with reviewing the landscape and trends of geoprocessing platforms. It will then describe the methodology used in this study, the results found, and will conclude with a discussion on what the results could mean for the future configuration of web GIS applications.

2. Literature Review

Geoprocessing is defined as, "A GIS operation used to manipulate GIS data. A typical geoprocessing operation takes an input dataset, performs an operation on that dataset, and returns the result of the operation as an output dataset (Wade and Sommer 2006, p.89)." Geoprocessing is the central role of a GIS and what this thesis used for testing the performance of a web browser. There are various platforms and configurations on which to execute geoprocessing. This section details the common technologies used and also touches on where the field is heading.

2.1 Traditional GIS

For decades, most GIS users have performed geoprocessing on desktop computers. Many desktop GIS software applications have been built for such a purpose: ArcGIS, MapInfo, SAGA, and QGIS to name a few. These applications are installed on a user's desktop machine and use resources and data stored directly on the desktop or on a server through a local-area network (LAN). Although this setup is convenient for users who frequently perform varied GIS tasks and is common in enterprise setups, it is limited by the sometimes prohibitive cost of the software, the resource size of the local computer, compatibility with the operating system, and the restriction of single-user access to the software (Peng and Ming-Hsiang 2003).

2.2 Distributed GIS

An increasingly common configuration is distributed GIS which is defined as, "... a network-centric (wired or wireless) GIS tool that uses the Internet or a wireless network as a primary means of providing access to distributed data and other information, disseminating spatial information and conducting GIS analysis (Peng and Ming-Hsiang 2003, pg. 12)." This is a

broad term that covers many configurations, but any GIS application that has the ability to use multiple servers and systems falls under this category, including those based on the web. The form of distributed GIS that is based on the web is commonly referred to as web GIS.

Web GIS employs a client-server model where the client (a web browser) sends an explicit request to the server in the form of Hypertext Transfer Protocol (HTTP) and the server performs the request and responds to the client (Kuuskeri and Mikkonen 2009). The advantage of web GIS is that it is not limited by operating system or type of machine; as long as there is an internet connection, the user has access to the application (Peng and Tsou 2003, pg.14).

Some of the limitations of this model are the prohibitive cost of both the software and hardware of the server as well as the more technical knowledge to build and maintain such a system. It also becomes challenging to provide interoperability which is "... the ability of various autonomous systems to bring together parts and to operate in collaboration (Vckovski 1998, pg. 9)." For example, inconsistencies in metadata can make finding data difficult if features are described using varied terminology or if descriptions are missing altogether. These interoperability challenges are currently a popular and important research field in web GIS and one that faces many trials for resolution.

2.3 Web Processing Services

One form of web GIS that grapples directly with interoperability challenges, and is still fairly new, is Web Processing Services (WPS) developed by the Open Geospatial Consortium (OGC). "WPS defines a standardized interface that facilitates the publishing of geospatial processes, and the discovery of and binding to those processes by clients (Schut 2007)." WPS is implemented in the client-server model where the client submits a processing task to be completed on the server using a strict format defined by the WPS specification (Michaelis and

Ames 2009). These defined specifications are meant to provide greater interoperability, making it easier for WPS to be published for use and for clients to find and bind to these various published WPS (Michaelis and Ames 2009). WPS relies on a server to perform the processing work. Although it has been recommended for future consideration, at the time of this writing the WPS specification does not support the ability to distribute any processing work to the client (Brauner et al. 2009).

A related field of processing is called live-streaming geoprocessing. This involves streaming data to a server which starts processing the data as soon as the first packet arrives, which means geoprocessing can begin before the entire dataset loads, which improves the total processing time (Foerster et al. 2012). The server-side GIS application 52North now has a version of WPS enabled which allows for streaming WPS when using the QGIS WPS plugin (Carillo 2012). Although there is much promise in WPS capabilities, this service is still in its early stages and has yet to be widely adopted and implemented, mainly due to the challenges of interoperability.

2.4 New Geoprocessing Direction

As mentioned, web GIS works through the use of HTTP, sending an explicit request to the server over the Internet to perform work and return a result. In web GIS, a common application architecture is one that uses the client browser as a graphical user interface (GUI) where requests for geoprocessing work are sent to the server, with the browser used only for rendering the results returned back to the client.

A configuration of web GIS that has not received as much research is the shift from entirely server-based geoprocessing to one in which the web browser handles some, or most, of

the data processing load. As web browsers become more powerful, the ability to handle more complex tasks is becoming increasingly common.

2.4.1 Client Geoprocessing

Most of the current web-based geoprocessing research focuses on server-side processing and WPS. The configuration not as commonly explored in this field, and the focus of this thesis, is browser geoprocessing, which places the work of geoprocessing on the client. There are a few studies deserving mention which have examined the utility of browsers for performing GIS tasks.

Much of the research related to client web GIS focuses on rendering vector data using HTML5 (Boulos et al. 2010; Corcoran et al. 2011). However, as will be described in the methods section, rendering is used for visual display and is handled by a different part of the web browser than JavaScript processing. While this research has important implications for usability and visualization of interactive maps, it did not test GIS analysis or processing tasks.

More applicable to geoprocessing is research performing generalization on vector data in the browser (Harrower and Bloch 2006; Wolf and Howe 2009). Generalization is commonly used in cartography as a means to make vector lines appear smoother in small scale maps. Generalization is a geoprocessing task, as it modifies the original input dataset. Harrower and Bloch built a client application that generalized vector files using the Douglas-Peucker-Ramer algorithm. While the application was successful in processing vector data, it was built in Adobe Flash, which is a browser plugin that needs to be installed in order to use, and therefore was not using JavaScript for performing any processing.

Wolf and Howe's 2009 study is most closely related to this thesis. They evaluated AJAX and ECMAScript (JavaScript) to determine how browsers performed when running the generalization algorithm Ramer-Douglas-Peucker (2009). This study confirmed the ability to

process data in a browser, but noted limitations with the study implementation, mainly due to the inefficiency of the browsers' JavaScript engines at the time of the study. Since this study was conducted, all major web browsers have released significant improvements to their JavaScript engines. Fairly complex datasets were used for testing in Wolf and Howe's study, with the number of vertices ranging from 2,711 to 53,300 (2009, p.126.). However, only four datasets were used for testing and all datasets were polylines. Wolf and Howe suggested that future research look at running the same processes on the client and server to determine if some processing tasks were more appropriate for the server or client and what parameters to use to determine those configurations.

Looking at a different geoprocessing task, Huang et al. (2011) built an application that performed spatial analysis on scalable vector graphics (SVG) in the browser (2011). This study looked at load balancing between the client and server for performing spatial queries by using SVG-based spatial information representation and SVG-based spatial extended SQL. It showed that performing spatial querying on SVG in the browser was possible and that it could save on high transmission loads, as results did not have to be sent back to the client over a network. The study also showed the feasibility of incorporating their solution into a load balancing client-server application. However, this solution only focused on using SVG to process data, which is not a common spatial analysis data format because it does not retain geographic coordinates. As this was a proof-of-concept study, it did not measure browser performance in analysis tasks, only looking at the transmission load from server to the client.

These studies did not address testing several different geoprocessing operations using JavaScript, running tests on a suite of data sizes, or benchmarking how the same tests run on the client compared to the server. In order to tackle that gap, this thesis will analyze the current

capability of web browsers in processing geographic data running common GIS operations with JavaScript in several different hardware platforms and web browsers.

2.4.2 Research Questions

As revealed by the literature review, there is little research that used JavaScript, a limited application of different GIS operations, and only handful of dataset sizes. To address these gaps and the overall goal to determine the viability of JavaScript-based client geoprocessing, this research will consider the following questions:

1. How do the various web browsers compare in geoprocessing performance?
2. How do client computers with different operating systems, processors, and memory sizes compare in geoprocessing performance?
3. How do the various client test configurations compare to server-side geoprocessing performance?
4. Are client geoprocessing times in an acceptable range for incorporation into web applications?

These questions will be addressed through a series of benchmarking tests using a suite of geospatial datasets, with three different GIS operations, in multiple web browsers, on several operating systems and computing platforms. For comparison purposes all of these tests will be repeated on a server. The next section details the methodology used in these benchmarking tests, including the building of the testing application and the metrics used to measure performance.

3. Methods

The goal of this research was to assess the viability of JavaScript-based client geoprocessing. To accomplish this goal, web browser geoprocessing performance and server geoprocessing performance were recorded and compared. To execute these tests, a web application was built that implemented the same processes on both the client and the server and recorded the duration of each process in milliseconds.

This section is structured as follows: the first part describes the geoprocessing library and operations used, the second details the data acquisition and modification, the third describes the tools and technologies of the testing application (on both the client and server sides). The final sections discuss the testing platforms and present the performance metrics.

3.1 Geoprocessing Library

In order to reduce disparities in the performance of the operations used to conduct the processing on both client and server testing, a server-side geoprocessing library was chosen that also had a client port available. That library was the Java library JTS Topology Suite (JTS) and its JavaScript port JSTS Topology Suite (JSTS).

JTS is a widely used Java 2D computational geometry API. It is the processing library behind GeoServer, OpenJump, gvSIG, uDig, and GeoTools. JSTS Topology Suite is a JavaScript port of JTS that was originally created for use in OpenLayers 2.11, but later modified for use as a free standing geometry library (Hartell 2013).

3.2 Geoprocessing Operations

The operations used for this research are based on what are considered some of the fundamental spatial operations for a GIS. The selection of these operations is based on the Michael Goodchild's *Towards an enumeration and classification of GIS functions* (1987), AAG's *GIS & Technology Body of Knowledge* (2006) and Jochen Albrecht's *Universal analytical GIS operations — a task-oriented systematization of data structure-independent GIS functionality* (1997). According to these foundational papers, the core locational analysis operations of a GIS are buffer, overlay, and Theissen polygons.

Although this research tested using some of the most common operations in a GIS, it was restricted to the operations available in JTS and how they were implemented. The first common process used was buffer. A buffer is a proximity analysis operation which involves creating an enclosure of a specified distance from the outer edge of a point, line, or polygon (Wade and Sommer 2006). JTS implements buffer using a buffer class, accepting points, lines, or polygons and returning a multipolygon.

The overlay method, union, extracts the overlapping portion of two or more polygons combining them into a single polygon (Wade and Sommer 2006). According to the developer of JTS, Martin Davis, the fastest implementation of union in JTS is Cascaded Union (Davis 2007). Cascaded Union recursively combines subsets of the overlapping areas together and continues to combine these increasingly larger subsets until the entire overlap is amalgamated (Davis 2007).

The Theissen polygon operation, resulting in a so-called Voronoi diagram, takes a set of points as input and creates an individual area around each point that represents the area that is closer to that point than any other point in the input (Wade and Sommer 2006). In JTS, the Theissen polygon operation returns a collection of polygons when given a set of input points.

Hereafter the terms "Voronoi diagram operation" and "Voronoi diagram" will be used to mean construction of a Voronoi diagram (or Theissen polygons).

3.3 Data Structure

JTS Topology Suite (JTS) conforms to the *OGC Simple Feature Access – Part 1: Common Architecture* with how it handles data (Davis and Aquino 2003, p. 7). This specification was taken into account when determining how to structure data for the testing application. For this reason, Well-Known Text (WKT) MultiPolygon, MultiPoints, and MultilLineStrings were chosen as the data formats for the testing polygons, points, and lines, respectively, as these formats are defined in the *OGC Simple Feature Access – Part 1: Common Architecture* (Herring 2011).

In order for these WKT types to meet the specification standards, each had to conform to a "simple" standard, which in various ways requires that features cannot overlap or have duplication of features. This is defined more specifically for each feature type. For example, "A MultiPoint is simple if no two Points in the MultiPoint are equal (have identical coordinate values in X and Y) (Herring 2011, p. 21)."

Similar to MultiPoints with their restrictions on duplication of coordinates, a MultiLineString, "... is simple if and only if all of its elements are simple and the only intersections between any two elements occur at Points that are on the boundaries of both elements (Herring 2011, p. 24)." Finally, MultiPolygons must meet the following two conditions:

> a) The interiors of 2 Polygons that are elements of a MultiPolygon may not intersect.
> b) The boundaries of any 2 Polygons that are elements of a MultiPolygon may not "cross" and may touch at only a finite number of Points. (Herring 2011, p. 31)

Because of this specification, all polygon results from a buffer that contain elements which intersect or touch are automatically dissolved into one polygon. There is no method for separating the action dissolve from the buffer operation in the JTS library, therefore the results

from buffer automatically include time to dissolve if the resulting buffered features overlap. To help mitigate including time to dissolve in buffer results, a small buffer distance was used (20 arc-seconds) which prevented roughly 95% of overlap.

3.4 Data Source

The starting data was procured from the Los Angeles County GIS Data Portal. This source was chosen for its wide selection of shapefiles, the availability of very large datasets (500MB or larger), and because of its policy of free download, use, and modification (County of Los Angeles 2014). The research design required data containing many features. From LA County, address points, road center lines, and building outlines satisfied this requirement. ***Table 1*** describes the LA County data acquired.

Table 1 Description of starting data obtained from LA County GIS.

Feature	**Points**	**Lines**	**Polygons**
Description	LA County Address Points	Road Center Lines	Countywide Building Outlines
No. of Features	2,900,000+	450000	3,000,000+
Size (MB)	1310	950	1370

3.5 Data Creation

The LA County GIS shapefiles were modified substantially for use in this study. The research design required data that all originated from the same source, but increased in size at controlled intervals. To create suitable data, LA County address points, road center lines, and buildings were used as starting places from which features were randomly selected to make data sets of predetermined size. The data used to test buffer and Voronoi diagrams were created differently than the data used to test union, due to union requiring two datasets as input.

Buffer tested points, lines, and polygons and Voronoi diagrams tested only points. The first step was converting the LA County shapefiles into WKT. For this step, QGIS 2.0.1-Dufour was used to save the file into a text file that was formatted as WKT (QGIS Development Team 2014). Next, a Python script was executed that randomly selected vertices from the WKT files and created smaller WKTs based on an array of predetermined vertex sizes ranging from 10 to 100,000 vertices (***Table 3***).

Union was tested on two sets of overlapping polygon datasets. A subsection of the LA County buildings shapefile was selected and saved to a new smaller shapefile of roughly 32 megabytes in size. This shapefile was then copied to create a duplicate identical shapefile. The second shapefile was offset from the first by 0.001 degrees in the x and y using the ArcGIS 'Move' tool in the Editor Toolbar. This test data was unusual in that in the real-world, one would not encounter buildings overlapping other buildings. However, in order to ensure that the union operation could execute similarly on data with any number of features, overlap had to occur with every feature. The two resulting shapefiles from this process were converted into WKT using QGIS and ran through the same Python script as the buffer and Voronoi data to create a data suite ranging in size from 10 to 100,000 vertices (***Table 3***).

The data sizes created represent fairly generalized data used for mapping. ***Table 2*** displays examples of data from Natural Earth, which is a widely used data source for cultural and physical GIS data (Natural Earth, 2014). Small (1:110m), medium (1:50m), and large scale (1:10m) data sizes were acquired from Natural Earth as a reference for how the test data compared to commonly used data sizes for mapping. The 10 to 100,000 vertices test data range covers these ***Table 2*** data sizes, with the exception of the largest data size acquired from Natural Earth of 548,449 vertices. Data sizes larger than 100,000 vertices were not included for testing, due to

alpha testing of the application that had browsers struggling to process data with only 50,000 vertices.

Table 2 Natural Earth points, polylines, and polygons at three scales: small, medium, and large. Used as reference for commonly used data sizes in mapping.

Data Name	**Data Type**	**Number of Vertices**	**Number of Features**	**File Size**
Large Scale Countries	Polygon	548,449	255	20.3 MB
Large Scale Boundary Lines	Polyline	77,116	461	2.8 MB
Large Scale Populated Places	Points	7,343	7,343	6.8 MB with attributes/ 338 KB without attributes
Medium Scale Countries	Polygon	99,423	241	3.8 MB
Medium Scale Boundary Lines	Polyline	19,423	360	751 KB
Medium Scale Populated Places	Points	1,249	1,249	1.2 MB with attributes/ 58 KB without attributes
Small Scale Countries	Polygon	10,421	178	462 KB
Small Scale Boundary Lines	Polyline	2,642	184	62 KB
Small Scale Populated Places	Points	243	243	243 KB with attributes/ 12 KB without attributes

Table 3 Sizes of test datasets in vertices and bytes used for union, buffer, and Voronoi diagram.

Vertices	**Union Polygon A (Bytes)**	**Union Polygon B (Bytes)**	**Points (Bytes)**	**Polylines (Bytes)**	**Polygons (Bytes)**
100000	4078191	4078095	4078044	4078077	4078239
90000	3670255	3670838	3670270	3670555	3670355
80000	3262448	3262335	3262423	3262950	3262848
70000	2854539	2854471	2854583	2854670	2854904
60000	2446920	2446711	2446823	2446924	2447057
50000	2039254	2038942	2039052	2039067	2039170
40000	1633469	1633352	1631197	1631247	1631231
30000	1224654	1224611	1223316	1223336	1223360
20000	815739	815655	815600	815634	816388
10000	408066	408026	407781	408840	408086
9000	367082	367066	367005	367405	367030
8000	326858	326839	326249	326275	326307
7000	285470	285460	285453	285454	285882
6000	245002	244976	244652	244676	244974
5000	203889	203879	203880	203942	204348
4000	163501	163488	163114	163311	163473
3000	122595	122580	122348	122413	123008
2000	82159	82157	81560	81678	82029
1000	40886	40875	40780	40787	41025
900	37136	37113	36699	36880	36707
800	32723	32688	32625	32684	32869
700	28630	28608	28547	28739	29123
600	24632	24607	24469	24492	24612
500	20636	20627	20397	20609	20857
400	16406	16399	16327	16344	16651
300	15237	15227	12249	12255	13279
200	8663	8663	8171	8228	8436
100	4661	4664	4087	4109	4102
90	3730	3726	3678	3730	4102
80	3730	3726	3270	3296	3517
70	3221	3219	2862	2886	3517
60	2829	2829	2456	2477	2738
50	2122	2124	2047	2159	2738
40	2122	2124	1641	1694	2738
30	1461	1458	1233	1504	2738
20	1030	1027	824	1504	2738
10	444	444	416	418	677

3.6 Testing Application

It was necessary for the testing application to reside on dedicated server space on the web and contain both client code, which would download and execute in a web browser, and server code, which would execute on the server. The application included both Java (server code) and JavaScript (client code) along with various application programming interfaces (API) in each language to aid in communication between the client and server and perform geoprocessing. The development environment used was Eclipse Kepler 4.3.1 with the Amazon AWS Toolkit for Eclipse plugin on a Windows 7 machine.

3.6.1 Client Structure

` The client application was developed in JavaScript and built around JSTS, while async.js and microajax.min.js were also used to help run the processing steps in serial (in order to prevent two processes running simultaneously) and communicate with the server, respectively. As visualized in ***Figure 1***, the flow of client testing occurred as follows: client sent request to server to retrieve data, WKT retrieved from the database, WKT returned to client, WKT parsed into JSTS geometry, geoprocessing of geometry, parsing geoprocessing results back to WKT. ***Table 4*** provides descriptions of the JavaScript contained in the application.

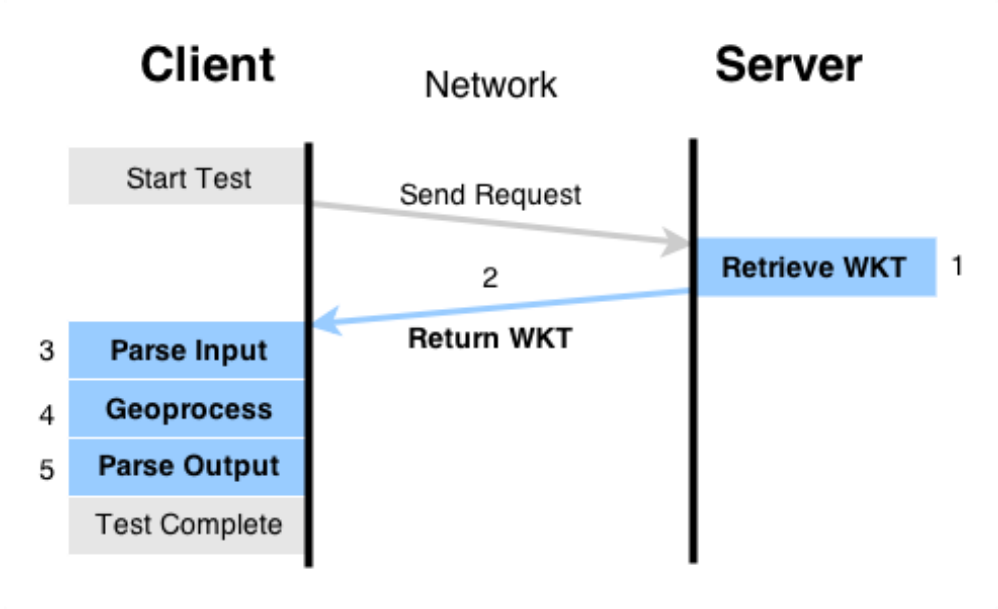

Figure 1 Client application test flow and steps recorded. 1. WKT retrieved from database on server. 2. WKT returned to client. 3. WKT parsed to JSTS geometry. 4. Run geoprocess. 5. Parse output geometry back to WKT.

Table 4 JavaScript libraries and custom classes for client-side the testing application.

Name	**Description**
JSTS.js	JSTS Topology Suite Client-side geoprocessing library
Javascript.util.js	JavaScript library to enable porting from JAVA (for JSTS)
Async.js	Flow control for creating a series of functions that will not run until the previous one has completed.
Microajax.min.js	Small library for performing cross-browser AJAX.
Boomerang.js	Yahoo! JavaScript library for measuring network performance
Main.js	Runs base latency and bandwidth tests, starts client and server tests from HTML button click.
Flow.js	Controls the flow of the data using ajax and async and sends data for processing

3.6.2 Server Structure

The server was a cloud-based, virtual server hosted on the Amazon Elastic Compute Cloud (EC2) platform. The virtual server was an Amazon 64-bit Linux m1.medium instance which was allocated up to 1 CPU, 3.75 GiB of memory, and accessed an Intel Xeon Family

processor. The environment was Java based, JRE 7 with a Tomcat 7 web server. The JAVA APIs JTS, Jersey JAX-RS, and SQLite JDBC were used for geoprocessing, RESTful web services, and SQLite database access, respectively. As visualized in ***Figure 2***, the flow of server testing occurred as follows: client sent request to server to begin main test, data retrieved from the database, WKT parsed into JTS geometry, geoprocessing of geometry, parsing geoprocessing results back to WKT, and results returned to the client. A description of each API and Java class can be viewed in ***Tables 5***.

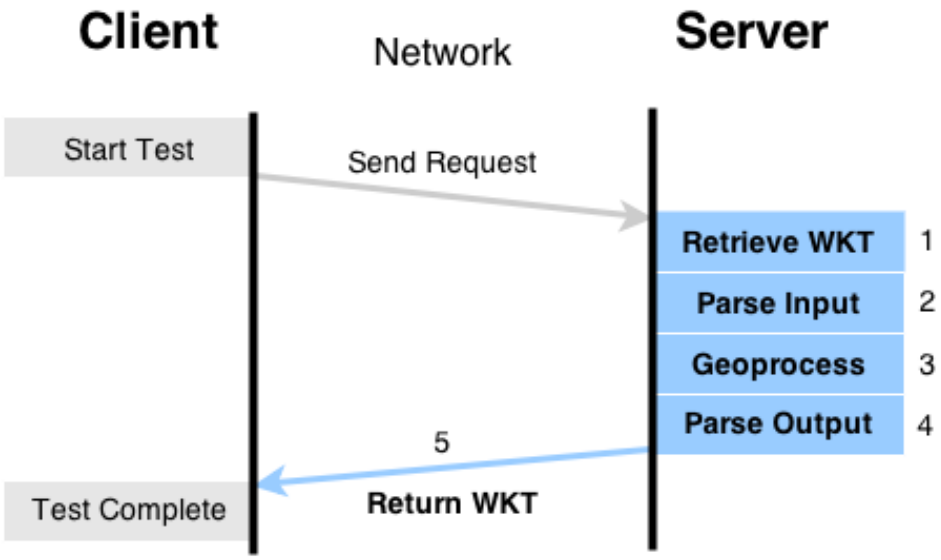

Figure 2. Server application test flow and steps recorded. 1. Retrieve WKT from database. 2. Parse WKT to JTS geometry. 3. Run geoprocess. 4. Parse geometry of geoprocess result back to WKT.

Table 5 Java libraries and custom classes for the server-side testing application.

Name	**Description**
Tomcat 7	Tomcat web server
Java Runtime Environment (JRE) 7	Version of Java
JTS Topology Suite 1.13	JTS Topology Suite used for server-side geoprocessing
Jersey JAX-RS 2.0	Java Restful Web Services API used for creating custom Java web services.
SQLite JDBC 3.7.15	Relational database for well-known text and results storage.
Flow.java	The main class of the library that controls flow of the server-side processing code
Services.java	Class that handles all web services of application
Storage.java	Class that handles all communication with database
Algorithms.java	Class that performs all geoprocessing on data
Format.java	Performed parsing and formatting functions on data

3.6.3 Network

When testing performance between the client and the server there are network variables beyond the developer's ability to modify: the three-way handshake, variability in bandwidth of the network, and latency. Because latency changes based on the physical distance of the user from the server (for example, the farther away the slower the response) and bandwidth fluctuates depending on the number of individuals on a network at a given time, the specific network speed for each geoprocessing test was not recorded and instead an average network metric was applied. Eighty-seven readings of the network were recorded using the Yahoo! Boomerang library and an average of these measurements was used for the network performance reading. These readings were then used to calculate the estimated time required to download WKT data for the client tests and the estimated time to return data results to the client from the results of the server tests.

3.6 Web Browsers

Web browsers are made up of two main components that provide the functionality for websites: the web browser engine (also called the layout or rendering engine) and the JavaScript engine. The web browser engine handles the layout and display based on the Document Object Model (DOM) and Cascading Style Sheet Object Model (CSSOM) which are tree objects that allow JavaScript to manipulate of browser layout and display. The rendering engine parses the HTML and CSS to construct the DOM and CSSOM, respectively (Grigorik 2013). The DOM and the CSSOM are then combined into what is called a render tree.

In order for browsers to execute JavaScript, they need an interpreter to transform and interpret the code to a machine readable format enabling the local machine to execute the task. In browsers this interpreter is known as the JavaScript engine and different browsers have different JavaScript engines working behind the scenes. Both the web browser and JavaScipt engine have impacts on performance, but this thesis focuses on the JavaScript engine, which is used when performing geoprocessing.

JavaScript engines often have different JavaScript interpreters that either convert the JavaScript directly to machine code or first convert to bytecode and then machine code (Chrome V8 2014; Bynens 2014; Miadowicz 2012). Although browsers are built with different JavaScript engines, the performance gaps between these engines are quickly narrowing as the organizations developing them work on improving the interpreters and optimizers. This research will look for remaining performance differences by running the same tests in each of the most commonly used browsers and comparing the results. The assumption is that the differences in performance between the browsers is primarily due to the implementation of these JavaScript engines. A list of browsers with their respective JavaScript Engines is listed in ***Table 6***.

Table 6 Web Browsers Versions and JavaScript Engines

Browser	**Browser Version**	**JavaScript Engine**	**Developed By**
Chrome	33	V8	Google
Opera	20	V8	Google
Firefox	27	SpiderMonkey	Mozilla
Internet Explorer	11	Chakra	Microsoft
Safari	6	SquirrelFish Extreme (SFX) aka Nitro Extreme	WebKit

Browsers use the resources and hardware of the local machine in order to run. Another component of this research will examine the importance of the hardware in the performance of JavaScript by testing the same code on platforms with varying CPU and memory sizes as well as different operating systems.

3.7 Testing Platforms

Testing platforms included Windows, Linux, and Mac laptops and desktops with varying sizes of processors and available memory (***Table* 7**). This mix of testing platforms was driven by their availability to the author. The hardware ranged in age from seven years to one year old. Because of this difference in age and specification, benchmark tests were conducted on all machines to obtain a more straightforward comparison between the platforms (***Table 7***). Benchmarks were conducted using Geekbench 3, a cross-platform processor benchmark from Primate Labs (Primate Labs 2013). All platforms were tested with Chrome for platform comparison. The Windows laptop tested all browsers, with the exception of Safari, which was tested on the Apple laptop.

Table 1 Client-side platform specifications. Benchmark scores recorded using GeekBench 3.

Brand	Operating System	OS Version	Processor	CPU (GHz)	Memory (GB)	Benchmark Score
Lenovo Y480	Windows	7 Home Premium Service Pack 1 (64-bit)	Intel(R) Core(TM) i7-3610QM	2.3	8	2478
Lenovo T61	Linux Mint	16 "Petra" Cinnamon (32-bit)	Intel Centrino Core 2 Duo CPU	2.5	6	1397
MacBook Pro	Mac OS X	10.7.5	Intel Core i7	2.8	8	2732
MacMini	Mac OS X	10.9.1	Intel Core i7	2.3	4	2708
Custom Built	Linux Mint	13 "Maya"	2x Intel(R) Core(TM)2 Duo CPU	3	8	1715
Custom Built	Windows	(32-bit)	2x Intel(R) Core(TM)2 Duo CPU	3	8	1637

3.8 Performance Tests and Measures

Performance was measured in milliseconds of execution time, with shorter times indicating better performance. Every task was broken down into separately timed steps. First, the time it took for the server-side code to retrieve the WKT data was recorded. Next, geoprocessing was measured, broken down into the time to parse the incoming WKT into JTS geometry, perform the geoprocess (buffer, union, or triangulation), and parse the geometry results back into valid WKT. All of the timed steps were then added together for a total processing time. All of the same processing steps in the same order was completed on the server and also timed in milliseconds.

The browsers were tested with only one tab open and nothing else running in the browser, including developer tools in Chrome or Firebug in Firefox. All background applications on the client testing machine were stopped and the computers were unattended

during tests. The tests began with buffer, starting with the smallest WKT, moving up in size to the largest, and iterating through points, lines, and polygons. The next test was Voronoi diagram, which only tested points. Finally, union was tested using polygons.

3.8.1 Testing Client Geoprocessing Usability

Most web browsers prompt timeout warnings or simply crash when a JavaScript task is taking too long to complete. The time length that is considered too long varies from browser to browser, but most prompt these warnings after five or ten seconds (Buckler 2010). Because this range in unresponsiveness and crashing varies so wildly, a different metric was needed to determine usability of geoprocessing in the browser. That metric was comparing the client geoprocessing results to web usability metrics to help understand how long users might expect processing to run in the browser.

Web usability research indicates that users will notice a lag in performance after 1,000 milliseconds (one second) for a task in a web browser to finish and will completely abandon a task if it takes longer than 10,000 milliseconds to complete (Grigorik 2013). Along with comparing the client geoprocessing results to the server, the performance results were also compared to these usability metrics to get a sense for how large of a dataset could be processed in these time frames.

4. Results

Performance was determined by the processing time for a given data size, with faster processing times indicating better performance. Browser and platform performance was compared using Voronoi diagrams. Usability of the client geoprocessing library was partially determined by comparing the results of the various geoprocessing operations, along with results from the browser and platform tests. The processing results were also analyzed against web usability metrics discussed in Chapter 3: 1,000 milliseconds and 10,000 milliseconds. This chapter begins by reviewing the results for browser performance, next looking at the platform results, and finishes by comparing the results against usability metrics. Each section also reviews the client results in comparison to the server results.

4.1 Web Browser Performance

Web browser performance was tested using Voronoi diagrams in Firefox, Chrome, Opera, Internet Explorer, and Safari. All browsers were tested on the same Windows laptop, with the exception of Safari, which only runs on Apple and therefore was tested on a MacBook Pro laptop. Browser performance was not only compared between browsers, but also against the server to determine if any browsers had comparable or better processing times than the server. Performance differences between browsers were assumed to primarily be due to differences in the underlying JavaScript engines. ***Table 8*** displays the average processing time in milliseconds of thirty test runs for each browser and the server. The processing times increased linearly, with the server times faster than the browsers by at least an order of magnitude on each data size

Table 8 Voronoi diagram average processing time in milliseconds of thirty test runs for each browser on the Windows laptop and the server.

Number of Vertices	**Chrome (ms)**	**Firefox (ms)**	**IE (ms)**	**Opera (ms)**	**Safari (ms)**	**Server (ms)**
10	42	66	65	44	22	10
20	33	42	25	31	17	3
30	34	47	29	36	17	5
40	33	44	30	35	19	5
50	35	45	29	38	22	6
60	26	44	29	29	21	6
70	28	41	29	32	24	8
80	36	44	32	37	28	6
90	23	43	31	24	22	7
100	23	47	33	24	24	8
200	38	62	51	42	37	11
300	48	86	81	46	60	17
400	61	110	114	55	92	18
500	91	131	159	78	130	25
600	89	158	208	70	173	28
700	112	194	268	86	230	32
800	133	231	344	97	284	35
900	163	271	424	116	352	41
1000	189	308	513	131	429	44
2000	646	930	1952	427	1593	77
3000	1402	2255	4354	931	3512	110
4000	2401	3260	7768	1540	6181	148
5000	3682	4892	12113	2403	9603	179
6000	5276	7024	17864	3372	13770	192
7000	7030	9458	24126	4505	18689	213
8000	9354	12007	31772	6082	24506	250
9000	11688	15193	40848	7587	30709	284
10000	14755	14461	51762	9408	37833	331
20000	79198	59109	288848	58402	149167	641
30000	246900	81280		194153		994
40000		145208				1371
50000		224744				1784
60000						2149
70000						2593
80000						3116
90000						3372
100000						3947

Based on the results in ***Table 8***, there did not appear to be large absolute processing differences among the web browsers for tasks with fewer than 2,000 vertices. The browsers either crashed or became unresponsive between the 30,000 and 50,000 vertices data sizes, which stopped the testing from continuing on to larger data sizes. Internet Explorer and Safari appeared to have the consistently slowest times, with processing times about thirty seconds slower than the fastest browsers when processing data of 9,000 vertices and larger (***Table 8***). While Opera outperformed Firefox until about the 20,000 vertices data size, Firefox was the only browser able to process data sizes larger than 30,000 vertices without crashing (***Table 8*** and ***Figure 3***).

One surprising result was the difference in performance between Opera and Chrome. As shown in ***Table 6*** in Chapter 3, Chrome and Opera are built with the same JavaScript engine, V8. However Opera was consistently faster than Chrome when processing more than about 400 vertices. At 4,000 vertices and above, it was faster by at least one second and by 20,000 vertices was twenty seconds faster (***Table 8***).

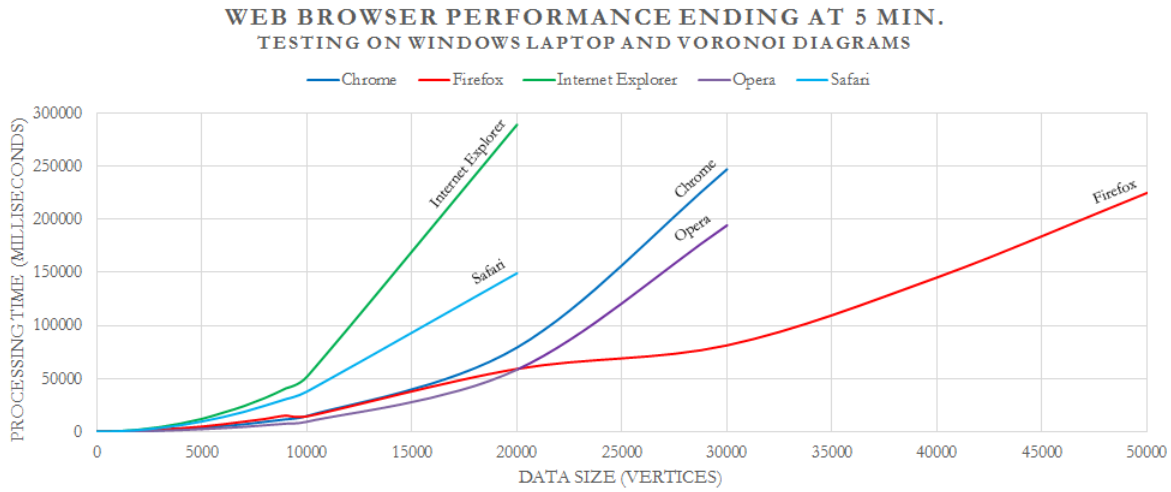

Figure 3 Web Browser Geoprocessing Results at Five Minutes

4.1.1 Web Browsers Compared to the Server

Results displayed in ***Table 8*** and ***Figure 4*** show a clear performance advantage for the server. In just under one second, the server processed a data size of 30,000 vertices, a data size an order of magnitude larger than largest data size processed by the client in the same time period (***Table 8*** and ***Figure 4***). In under two seconds, the server processed a data size of 50,000 vertices, which was the largest data size any of the browsers (Firefox), were able to process before crashing (***Table 8*** and ***Figure 4***). The server processed the largest dataset available in the test suite (100,000 vertices) in under four seconds (***Table 8***).

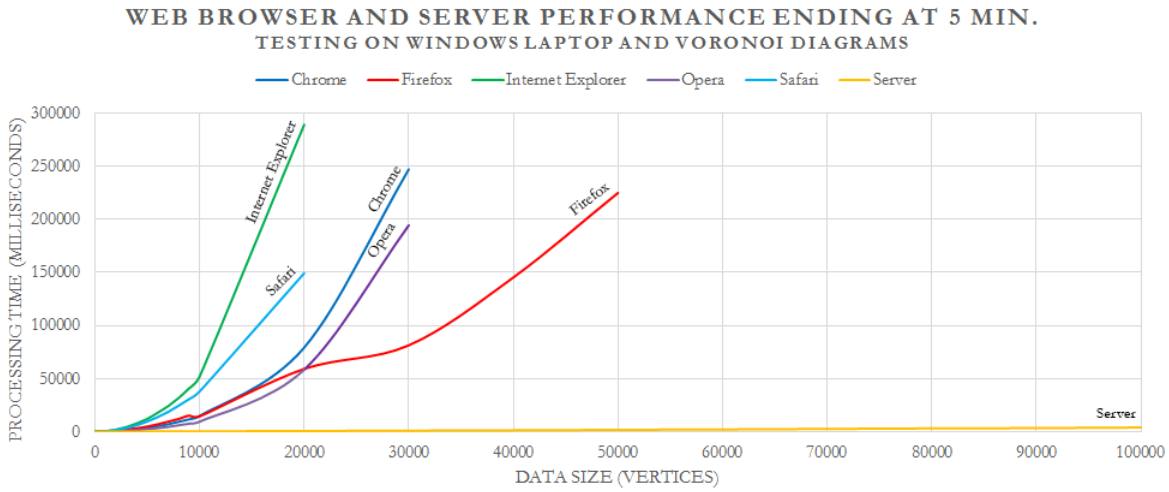

Figure 4 Web Browser and Server Geoprocessing Results at Five Minutes

4.2 Hardware and Operating System Performance

To understand how the hardware of computers (mainly the CPU and memory size) and operating systems influence browser performance, tests were conducted across six different client platforms. These tests covered three different operating systems (Linux, Apple, and Windows) as well as six different combinations of CPU and memory (***Table 7***). To eliminate

possible variability between browser implementation, all platform tests were conducted in Chrome with the Voronoi diagram operation.

The Apple laptop and desktop and the Windows laptop had the highest scores from the Geekbench processor benchmarking tests, at 2732, 2708, and 2478, respectively (***Table 7*** and ***Table 9***). These same platforms also had comparable processing times, and overall were slightly faster than the other platforms: two seconds faster than the slowest platforms at 4,000 vertices and five seconds faster than the slowest platforms by 8,000 vertices (***Table 9*** and ***Figure 5***). The Linux desktop and Windows desktop were on the same dual boot machine and had similar processor benchmark scores of 1715 and 1637, respectively. The processing time of these results were within 100 milliseconds of one another through 10,000 vertices (***Table 9***). At 20,000 vertices, the Windows desktop outperformed the Linux desktop by twenty seconds. This indicates that the operating system possibly influenced results processing data 20,000 vertices and larger.

Comparing processor and memory size did not determine the performance of the client platforms. Despite the Windows and Linux desktops having the largest CPU/memory combination at 3GHz/8GB, they had some of the lowest processor benchmark scores of 1637 and 1715, respectively. This likely explains the slower performance results from these two machines, which had processing times lagging behind the fastest machines by forty seconds at 20,000 vertices (***Table 7*** and ***9***). The Apple desktop had the smallest CPU/memory combination at 2.3GHz/4GB, but had one of the larger benchmarking scores at 2708 and had the fastest overall processing times for data larger than 20,000 vertices (***Table 7*** and ***Table 9***).

Table 9 Voronoi diagram geoprocessing time results for the server and various client test platforms. Numbers in parenthesis are benchmark results from Geekbench 3.

Number of Vertices	**Apple Desktop (2708)**	**Apple Laptop (2732)**	**Linux Desktop (1715)**	**Linux Laptop (1397)**	**Windows Desktop (1637)**	**Windows Laptop (2478)**	**Server**
10	36	36	26	37	23	42	10
20	23	22	14	24	14	33	3
30	23	28	16	26	16	34	5
40	21	24	16	27	16	33	5
50	21	23	18	26	18	35	6
60	17	19	13	19	12	26	6
70	19	20	12	19	11	28	8
80	23	24	18	26	17	36	6
90	15	16	10	15	10	23	7
100	16	17	10	15	9	23	8
200	23	22	21	25	21	38	11
300	29	30	34	37	33	48	17
400	38	37	50	45	49	61	18
500	56	57	85	77	85	91	25
600	52	54	86	69	86	89	28
700	66	68	116	83	118	112	32
800	80	78	147	104	150	133	35
900	98	94	181	119	179	163	41
1000	114	108	221	136	221	189	44
2000	406	374	805	512	806	646	77
3000	930	868	1822	1053	1825	1402	110
4000	1451	1339	3078	1810	3126	2401	148
5000	2306	2152	4873	2753	4904	3682	179
6000	3236	3011	6786	4131	6897	5276	192
7000	4389	4077	9320	5724	9344	7030	213
8000	5986	5723	12112	7591	12224	9354	250
9000	7562	7411	15259	10177	15294	11688	284
10000	10337	9783	17821	12763	17856	14755	331
20000	61870	96657	102634	70616	79568	79198	641
30000	208047	292897	428638	305142	410338	246900	994
40000							1371
50000							1784
60000							2149
70000							2593
80000							3116
90000							3372
100000							3947

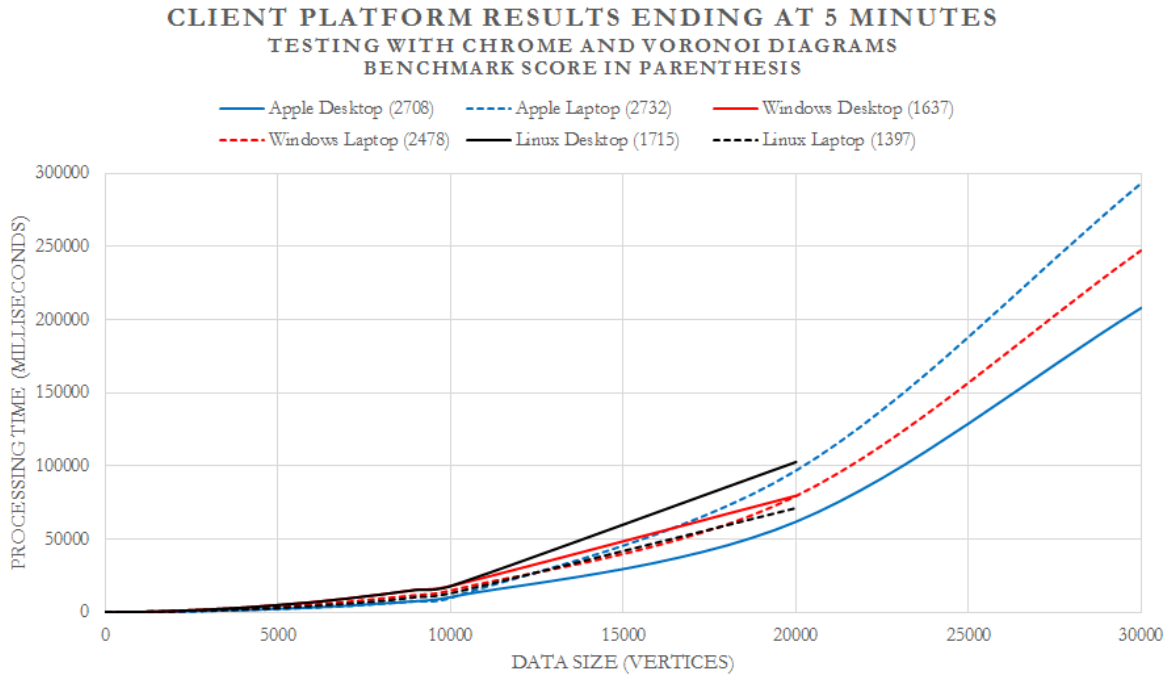

Figure 5 Voronoi diagrams geoprocessing in Chrome on client platforms ending at five minutes.

4.2.1 Client Platforms Compared to the Server

As found in the web browser performance section, the server again outperformed all of the client platforms by at least an order of magnitude for each data size processed (***Table 9***). In under one second the server processed a data size of 30,000 vertices, which was the largest data size any of the hardware platforms were able to process in under five minutes (***Table 9*** and ***Figure 6***). The server processed the largest dataset available in the test suite (100,000 vertices) in under four seconds (***Table 9***).

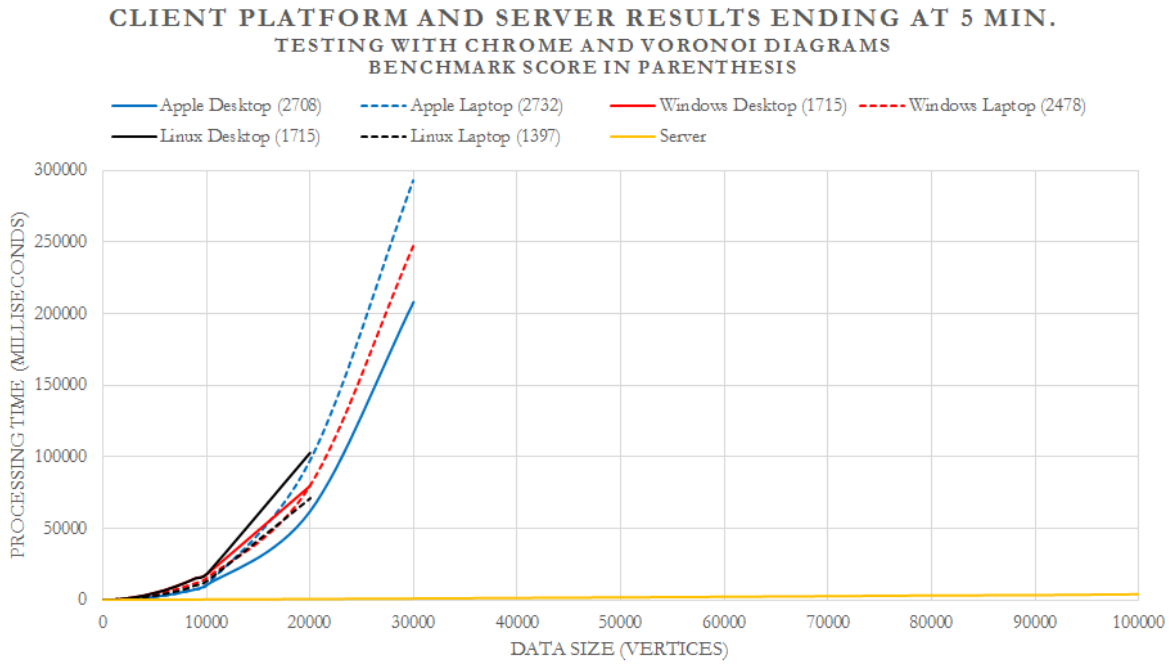

Figure 6 Voronoi diagram geoprocessing in Chrome on client platforms compared to the server ending at five minutes.

4.3 Client Geoprocessing Usability

Beyond comparing the client to the server, the viability of client geoprocessing was also assessed using web usability metrics. Chapter 3 discussed the usability metrics of 1,000 and 10,000 milliseconds when determining site usability. This section looks to frame the results within the context of those usability metrics to get a sense for how long users would be willing to wait for a process to run in the browser before abandoning the task.

First, examining the results from the platforms testing Voronoi diagrams in Chrome in ***Table 10*** and ***Figure 7***, it appears that after one second, all of the platforms were able to process a data size of 2,000 to 3,000 vertices. After ten seconds, those sizes ranged from 7,000 to 10,000 vertices. Comparing the results of the server, it is found that the server processed a data size of 30,000 vertices in one second and 100,000 in ten seconds.

Table 10 Voronoi diagram tests in Chrome with maximum data size (number of vertices and bytes) processed in under the specified time across the various hardware platforms and operating systems.

Hardware	1 sec (vertices)	1 sec (kb)	10 sec (vertices)	10 sec (kb)
Windows Laptop	2,000	79	8,000	318
Windows Desktop	2,000	79	7,000	278
Apple Laptop	3,000	119	10,000	398
Apple Desktop	3,000	119	9,000	358
Linux Desktop	2,000	79	7,000	278
Linux Laptop	2,000	79	8,000	318
Server	30,000	1,194	100,000	3,982

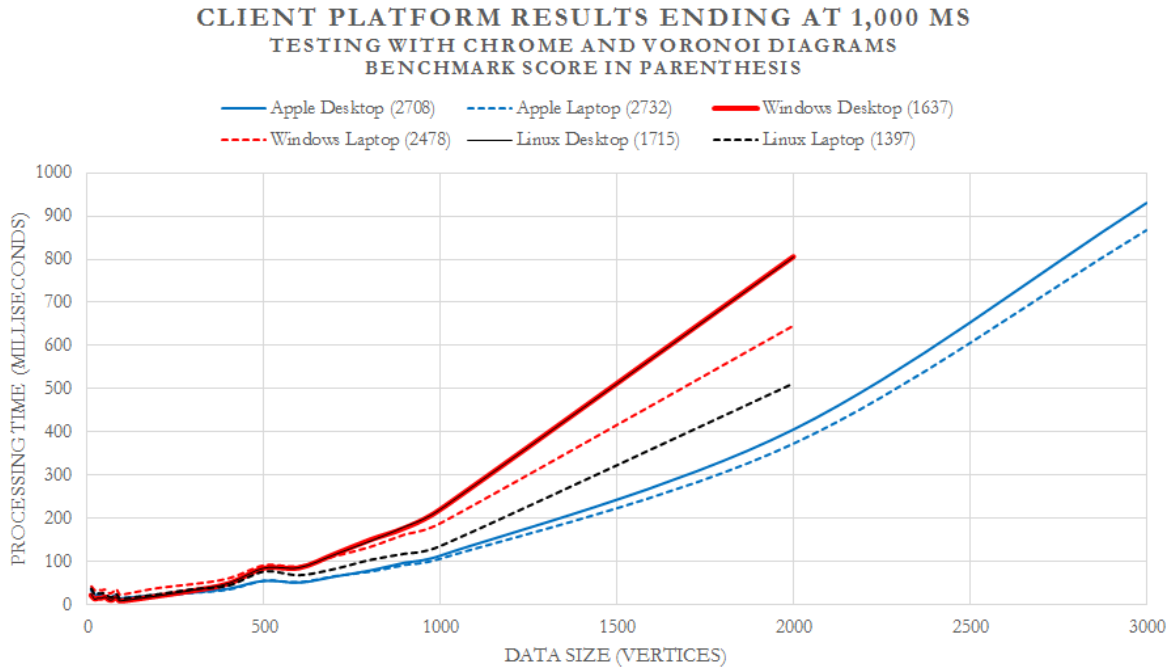

Figure 7 Platform Geoprocessing Performance at 1,000 Milliseconds. Linux and Windows desktop results fall on the same line.

Table 11 and ***Figure 8*** shows the web browser performance of Voronoi diagram geoprocessing results in relation to usability metrics. After one second, the data sizes range from 1,000 to 3,000 vertices and after ten seconds, 4,000 to 10,000 vertices. The server has the same

results as the previous table with 30,000 and 100,000 vertices for one and ten seconds, respectively.

Table 11 Voronoi diagrams on Windows laptop testing web browsers. Data sizes shown are the max data size that can run a process in under the various cutoff times listed.

Browser	**1 sec (vertices)**	**1 sec (kb)**	**10 sec (vertices)**	**10 sec (kb)**
Chrome	2000	80	8000	318
Opera	3000	119	10000	398
Firefox	1000	40	7000	278
Internet Explorer	1000	40	4000	159
Safari	1000	40	5000	199
Server	30000	1194	100000	3982

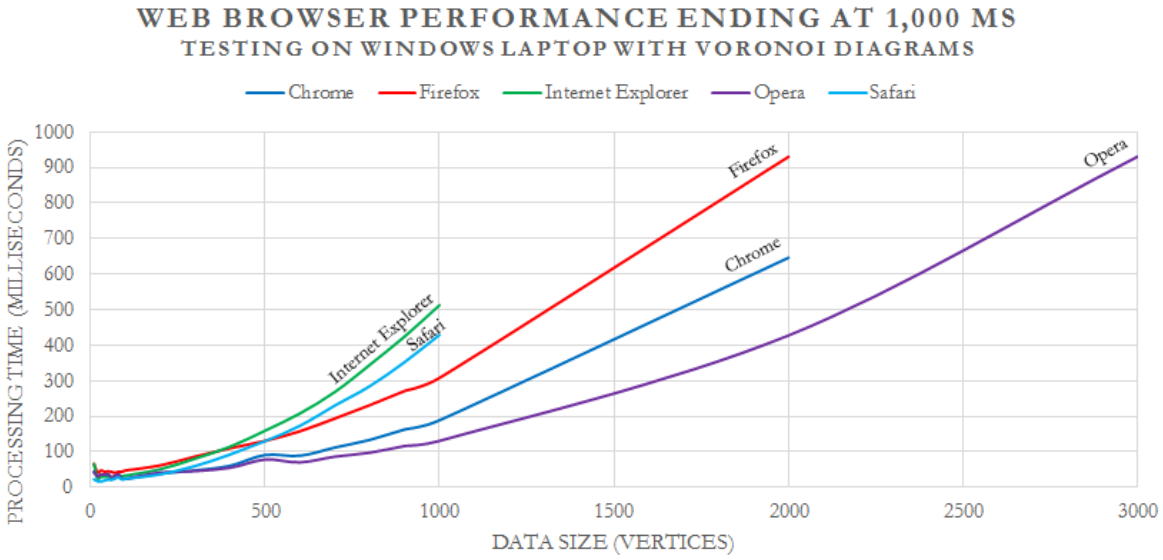

Figure 8 Web Browser Voronoi diagram Geoprocessing Performance at 1,000 Milliseconds on a Windows laptop.

Table 12 and ***Figure 9*** displays the results of Chrome on the Windows laptop running the various geoprocessing operations. It appears that the client results varied in size from 900 vertices (union) to 6,000 vertices (buffer polygons) after one second and 3,000 vertices (union) up to 10,000 vertices (buffer polygons) in ten seconds depending on the data type and operation

performed. In comparison, on the server after one second the data sizes range from 4,000 vertices for buffer points up to 30,000 vertices for Voronoi diagrams and buffer polygons. After ten seconds, those sizes jump to 20,000 and 100,000 for the same operations, respectively.

Table 12 Geoprocessing tasks on client and server tested on Chrome and a Windows laptop.

Client/ Server	**Geoprocess**	**1 sec (vertices)**	**1 sec (kb)**	**10 sec (vertices)**	**10 sec (kb)**
Client	Buffer Points	2,000	79	7,000	278
Client	Buffer Polygons	6,000	239	8,000	318
Client	Buffer Lines	4,000	159	10,000	399
Client	Voronoi Diagrams	2,000	79	8,000	318
Client	Union Polygons	900	35	3,000	119
Server	Union Polygons	10,000	398	50,000	1,991
Server	Buffer Lines	10,000	399	30,000	1,194
Server	Buffer Polygons	30,000	1,194	100,000	3,982
Server	Buffer Points	4,000	159	20,000	796
Server	Voronoi Diagrams	30,000	1,194	100,000	3,982

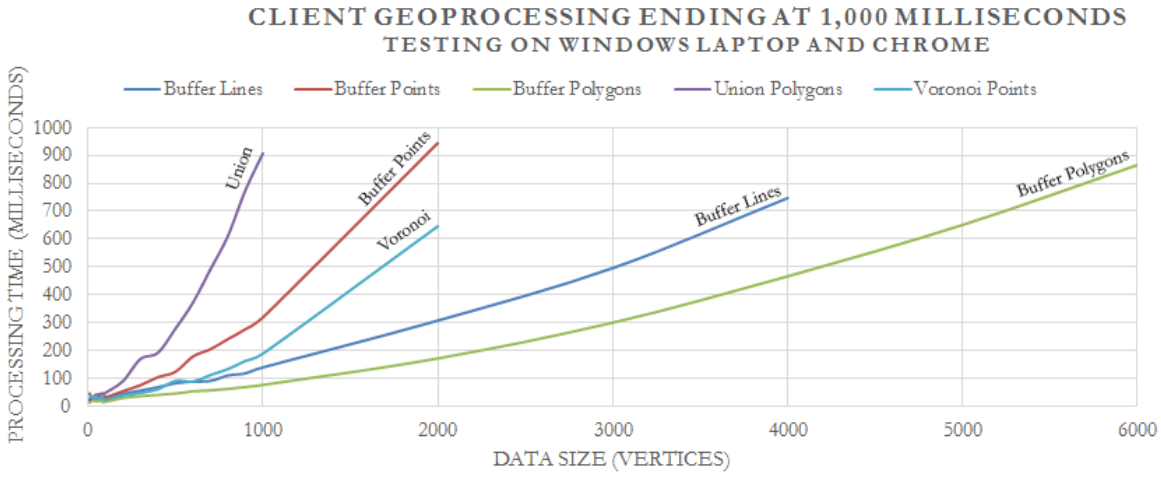

Figure 9 Geoprocessing operations tested in Chrome and a Windows laptop.

Depending on the client platform, browser, and geoprocessing operation, the clients can process between 900 and 6000 vertices in one second. In ten seconds between 3,000 and 10,000 vertices can be processed. For the server, the comparable one-second and ten-second problem sizes are 4,000 to 30,000 vertices and 20,000 to 100,000 vertices, respectively.

5. Discussion

In the beginning of this thesis, four questions were posed to address the goal of assessing the viability of the trend toward client JavaScript geoprocessing. This section considers each question individually based on the results found in Chapter 4.

1) How do various web browsers compare in geoprocessing performance?

Internet Explorer and Safari had the slowest processing times among the browsers, processing data of 9,000 vertices thirty seconds slower than Opera, Chrome, or Firefox. Firefox was the only browser able to process a data size larger than 30,000 vertices without crashing. Although Chrome and Opera are built on top of the same JavaScript engine, V8, their processing times were slightly different, with Opera having consistently faster processing times than Chrome. At 4,000 vertices, Opera was one second faster and by 20,000 vertices was twenty seconds faster than Chrome. It is possible that the implementation of V8 between the two browsers is slightly different, but the exact explanation for these differences is unknown at this time.

2) How do client computers with different operating systems, processors, and memory sizes compare in geoprocessing performance?

Processor and memory size values did not indicate performance, instead processor benchmarking scores were a better indicator of performance. The platforms with the largest CPU/memory combination (Windows/Linux desktop at 3.0Ghz/8.0GB) had the slowest performance overall, seeing results overall 45% slower than the fastest platform, Apple desktop, which had the smallest CPU/memory combination at 2.3Ghz/4.0GB. However, comparing the processor benchmark scores, the Apple desktop has a score of 2708, while the Windows/Linux

desktops have scores of 1637/1715, respectively. The platforms with the largest processor benchmarking scores (Apple desktop and laptop and Windows laptop) had processing times 60% faster at 30,000 vertices than the platforms with the smallest processing benchmarking scores (Windows/Linux desktop and Linux laptop).

One factor that might have contributed to these benchmark differences was age of the computers; the Windows desktop was the oldest machine of the testing platforms and the Apple desktop was one of the newest. This might indicate an overall performance improvement in the hardware beyond the processor and memory size.

Operating system possibly had an influence in results while processing data larger than 20,000 vertices. The dual boot Windows/Linux desktop had the Windows machine outperforming the Linux by twenty seconds at 20,000 vertices. At the smaller data sizes, the processing times were within 100 milliseconds. More tests on dual boot machines would need to be performed before differences in processing times could be attributed to operating systems.

3) How do the various client test configurations compare to server-side geoprocessing performance?

The results in Chapter 4 showed that the server performed geoprocessing at least an order of magnitude faster than any of the testing configurations on the client. There did not appear to be a client testing configuration that was even comparable to that of the server. This indicates that if speed is the only consideration, it is advantageous to place geoprocessing work on the server rather than the client. It is almost essential that a server be used when processing more than about 10,000 vertices in order to prevent unresponsiveness and client crashes.

This research did not test the scenario of multiple users requesting server geoprocessing at one time. Multiple users making requests for geoprocessing on the server would likely see a

decline in server performance results. Geoprocessing smaller data sizes on the client is possibly advantageous over the server if the server is handling a heavy load of user requests.

4) Are client geoprocessing times in an acceptable range for incorporation into web applications?

Although the server geoprocessing outperformed clients in all geoprocessing tests, clients are a viable option for many problems. In particular, tasks with fewer than about 10,000 vertices are within client-side processing capabilities.

To have a clearer picture as to when the user's patience would likely end when waiting for a processing task to complete, web usability metrics were used as a benchmark for performance. As shown in ***Tables 10, 11,*** and ***12*** it was determined that browsers were able to process data ranging in size from 900 to 6,000 vertices in one second and data sizes of 3,000 to 10,000 vertices in ten seconds. Comparing these size ranges to Natural Earth data in ***Table 4***, it appears these results fall in the range of the small scale vector data available for download on the Natural Earth website. Small scale data on Natural Earth are very generalized and recommended for use at a scale of 1 to 110 million (Natural Earth 2014).

As described in Chapter 3, the results obtained were recorded under an ideal browser test environment: no additional tabs open, background processes running, or interactive interfaces requesting additional tasks. It is likely that in real-world use, the geoprocessing performance would decline, not seeing the same results as found in this thesis. All of this information indicates that in real-world implementation, the geoprocessing ability of JSTS would be constrained to only processing generalized, small data sets (under 10,000 vertices).

5.1 Conclusions

The technologies to develop web mapping internet applications have been changing for the last twenty years and most recently moved to a combination of JavaScript and HTML5. Major mapping software companies introduced their own JavaScript-based APIs for both visualization and geoprocessing in the past several years. The appearance of JavaScript libraries that perform geoprocessing directly in the browser is a recent trend, but has not yet been successfully incorporated into GIS web mapping applications.

This research aimed to assess the trend toward JavaScript client geoprocessing by testing the performance of web browsers compared to a more traditional server-side architecture of web-based geoprocessing. The results indicated that servers are faster than clients, but when processing generalized small scale data, client performance is still in an acceptable time range. These results demonstrated that the current implementation of web browsers are limited in their ability to execute JavaScript geoprocessing and not yet prepared to process data sizes larger than about 8,000 to 10,000 vertices before either prompting an unresponsive script warning in the browser or potentially losing the interest of the user.

As said in the introduction, JSTS was originally developed for incorporation into OpenLayers 2.11 (Hartell, 2013). In December 2013, OpenLayers 3.0.0 was released as a complete rewrite of the library developed for better performance. As of the writing of this thesis, JSTS has not been included in this new OpenLayers release. The results of this research indicate that JSTS is ready for incorporation into web mapping applications with the stipulation that geoprocessing is restricted to small, generalized data. This author recommends reincorporating JSTS into OpenLayers 3.0 as a plugin.

As browser technology continues to improve and more JavaScript libraries are developed, additional tests will need to be conducted using the most recent technology to assess if browser-based geoprocessing can become viable for all problem sizes.

6. Bibliography

Adobe Systems Incorporated. 2012. *Adobe's view of Flex and its commitments to Flex in the future*. San Jose, CA: Adobe Systems Incorporated, Adobe Developer Connection. Available at http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/flex/pdfs/flex_roadmap.pdf (last accessed 19 July 2014).

ArcGIS Server Development Team. 2008. *The ArcGIS API for Flex 1.0 has been released*. Redlands, CA: ESRI, ArcGIS Resource Center. Available at http://blogs.esri.com/esri/arcgis/2008/10/24/the-arcgis-api-for-flex-1-0-has-been-released/ (last accessed 20 July 2014).

Boulos, M., J. Warren, J. Gong, and P. Yue. 2010. *Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping*. International Journal of Health Geographics, 9, 14. doi:10.1186/1476-072X-9-14

Brauner, J., T. Foerster, and B. Schaeffer. 2009. *Towards a research agenda for geoprocessing services*. 12th AGILE International Conference on Geographic Information Science 2009 (Vol. 1, pp. 1-12). Hannover, Germany. Available at http://www.ikg.unihannover.de/fileadmin/agile/paper/124.pdf

Buckler, C. 2010. *JavaScript execution and browser limits*. Available at http://www.sitepoint.com/javascript-execution-browser-limits/ (last accessed 19 July 2014).

Bynens, M. 2014. *Opera 20 Released*. Dev.Opera. Available at http://dev.opera.com/articles/view/opera-20/ (last accessed 15 May 2014).

Chrome V8. 2012. *Introduction*. Mountain View, CA: Google, Google Developers. Available at https://developers.google.com/v8/intro (last accessed 16 April 2014).

County of Los Angeles. 2014. *Disclaimer*. Los Angeles, CA: Los Angeles County GIS Data Portal. Available at http://egis3.lacounty.gov/dataportal/about/disclaimer/ (last accessed 6 January 2014).

Countywide Building Outlines [computer file]. 2008. Los Angeles, CA: Los Angeles Region Imagery Acquisition Consortium Program. Available via LA County GIS Portal FTP: http://egis3.lacounty.gov/dataportal/wp-content/uploads/2012/11/lariac_buildings_2008.zip [January 6, 2014]

Davis, M. 2007. *Fast Polygon Merging in JTS using Cascade Union*. Victoria, BC: Lin.ear th.inking. http://lin-ear-th-inking.blogspot.com/2007/11/fast-polygon-merging-in-jts-using.html (last accessed 15 July 2014).

Davis, M., and J. Aquino. 2003. *JTS Topology Suite Technical Specifications* (Reference No. 1.4.1). Available at http://www.vividsolutions.com/jts/bin/JTS%20Technical%20Specs.pdf

Foerster, T., B. Baranski, and H. Borsutzky. 2012. *Live Geoinformation with Standardized Geoprocessing Services*. In J. Gensel, D. Josselin, and D. Vandenbroucke (Eds.), Bridging the Geographic Information Sciences (pp. 99–118). Berlin, Heidelberg: Springer Berlin Heidelberg. Available at http://www.springerlink.com/index/10.1007/978-3-642-29063-3_6

Gensel, J., D. Josselin, and D. Vandenbroucke. 2012. *Bridging the Geographic Information Sciences: International AGILE'2012 Conference, Avignon (France), April, 24-27, 2012*. Berlin: Springer.

Germán, C. 2012. *52°North Blog: Full Streaming WPS: Near Real-time Geoprocessing with WPS*. 52°North Blog: Full Streaming WPS: Near Real-time Geoprocessing with WPS. Available at http://blog.52north.org/2012/10/04/full-streaming-wps-near-real-time-geoprocessing-with-wps/ (last accessed 16 April 2014).

Grigorik, Ilya. 2013. *High-performance browser networking*. Sebastopol, CA: O'Reilly Media, Inc.

Guo, Y. 2014. *Compiling in the background for a smoother user experience*. Mountain View, CA: Google, Chromium Blog: News and development from the open source browser project. Available at http://blog.chromium.org/2014/02/compiling-in-background-for-smoother.html (last accessed 23 April 2014).

Harrower, M. and M. Bloch. 2006. *MapShaper.org: A map generalization web service*. IEEE Computer Graphics and Applications 26, no. 4 (12 2006): 22-27. doi:10.1109/MCG.2006.85.

Helm, C. 2013. *Shapely.js*. Available at https://github.com/chelm/shapely.js/ (last accessed 15 July 2014).

Herlocker, M. 2014. *Turf.js*. Available at https://github.com/Turfjs/turf (last accessed 17 July 2014).

Herring, J. 2011. *OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 1: Common architecture* (Reference No. OGC 06-103r4). Available at http://portal.opengeospatial.org/files/?artifact_id=25355

Huang, C.H., T.R. Chuang, D.P. Deng, and H.M. Lee. 2009. *Building GML-native Web-based Geographic Information Systems*. Computers & Geosciences 35, no. 9 (12 2009): 1802-816. doi:10.1016/j.cageo.2008.11.009.

Huang, H., Y. Li, G. Gartner, and Y. Wang. 2011. *An SVG-based method to support spatial analysis in XML/GML/SVG-based WebGIS*. International Journal of Geographical Information Science 25, no. 10 (12 2011): 1561-574. doi:10.1080/13658816.2010.532133.

Jobs, S. 2010. *Thoughts on Flash*. Cupertino, CA: Apple, Hot News. Available at http://www.apple.com/hotnews/thoughts-on-flash/ (last accessed 20 July 2014).

Harrtell, B. 2013. *JSTS Topology Suite*. Available at https://github.com/bjornharrtell/jsts (last accessed 25 May 2014).

Kay, R. 2009. *QuickStudy: Rich internet applications*. Computer World. Available at http://www.computerworld.com/s/article/335519/Rich_Internet_Applications (last accessed 20 July 2014).

Koch, P. 2006. *Introduction to JavaScript*. Quirksmode. Available at http://www.quirksmode.org/js/intro.html (last accessed 17 July 2014).

Kuuskeri, J. and T. Mikkonen. 2009. *Partitioning web applications between the server and the client*. Proceedings of the 2009 ACM symposium on Applied Computing - SAC '09 (pp.647-652). New York, New York, USA: ACM Press. Doi:10.1145/1529282.1529416

LA County Address Points [computer file]. 2012. Los Angeles, CA: Countywide Address Management System Program. Available via LA County GIS Portal FTP: http://egis3.lacounty.gov/dataportal/wp-content/uploads/2012/06/lacounty_address_points.zip [January 6, 2014]

Miadowicz, A. 2012 *Advances in JavaScript performance in IE10 and Windows 8*. IEBlog. Available at http://blogs.msdn.com/b/ie/archive/2012/06/13/advances-in-javascript-performance-in-ie10-and-windows-8.aspx (last accessed 10 May 2014).

Mozilla Development Network. 2014. *HTML5*. Available at https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5 (last accessed 15 May 2014).

Natural Earth. 2014. *Natural Earth Data Download.* Available at http://www.naturalearthdata.com/downloads/ (last accessed 17 July 2014).

Peng, Z.R., and M.H. Tsou. *Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks*. Hoboken, NJ: Wiley, 2003.

Powell, J. 2014. *Esri's roadmap for web developers*. Redlands, CA: ESRI, ArcGIS Resources. Available at http://blogs.esri.com/esri/arcgis/2014/02/21/esris-roadmap-for-web-developers/ (last accessed 15 July 2014).

Primate Labs Inc. 2013. *Geekbench 3*. Toronto, Ontario. Available at http://www.primatelabs.com/geekbench/ (last accessed 4 August 2014).

QGIS Development Team. 2013. *QGIS Geographic Information System*. Open Source Geospatial Foundation Project. Available at http://qgis.osgeo.org (last accessed 15 May 2014).

Sveen, A.F. 2014. *Njord.js*. https://github.com/atlefren/njord.js (last accessed 17 July 2014).

Vivid Solutions. 2013. *JTS Topology Suite*. Available at http://tsusiatsoftware.net/jts/main.html (last accessed 15 May 2014).

Wade, T. and S. Sommer. 2006. *A to Z GIS: An illustrated dictionary of geographic information systems*. Redlands, CA: ESRI Press.

Warren, C. 2012. *The life, death, and rebirth of Adobe Flash*. Available at http://mashable.com/2012/11/19/history-of-flash/ (last accessed 15 July 2014).

Westphal, R. 2013. *jQuery Geo*. Available at http://jquerygeo.com/ (last accessed 17 July 2014).

Wolf, E. B. and K. Howe. 2009. *Web-Client based distributed generalization and geoprocessing*. 2009 International Conference on Advanced Geographic Information Systems & Web Services (pp. 123-128). Ieee. doi:10.1109/GEOWS.2009.32

Yang, B. and R. Weibel. *Editorial: Some thoughts on progressive transmission of spatial datasets in the web environment*. Computers & Geosciences 35, no. 11 (12 2009): 2175-176. doi:10.1016/j.cageo.2009.07.001.

Yang, C., D.W. Wong, R. Yang, M. Kafatos, and Q. Li. 2005. *Performance-improving techniques in Web-based GIS*. International Journal of Geographical Information Science 19, no. 3 (12 2005): 319-42. doi:10.1080/13658810412331280202.

2010 Tiger Roads [computer file]. 2010. Washington, DC: US Census Bureau Geography Division. Available via LA County GIS Portal FTP: http://egis3.lacounty.gov/dataportal/wp-content/uploads/2011/04/tigerroads.zip [January 5, 2014]