Tile-based Methods for Online Choropleth Mapping: A Scalability Evaluation

by

Myung-Hwa Hwang

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved September 2013 by the
Graduate Supervisory Committee:

Luc Anselin, Chair
Sergio J. Rey
Elizabeth A. Wentz

ARIZONA STATE UNIVERSITY

December 2013

ABSTRACT

Choropleth maps are a common form of online cartographic visualization. They reveal patterns in spatial distributions of a variable by associating colors with data values measured at areal units. Although this capability of pattern revelation has popularized the use of choropleth maps, existing methods for their online delivery are limited in supporting dynamic map generation from large areal data. This limitation has become increasingly problematic in online choropleth mapping as access to small area statistics, such as high-resolution census data and real-time aggregates of geospatial data streams, has never been easier due to advances in geospatial web technologies. The current literature shows that the challenge of large areal data can be mitigated through tiled maps where pre-processed map data are hierarchically partitioned into tiny rectangular images or map chunks for efficient data transmission. Various approaches have emerged lately to enable this tile-based choropleth mapping, yet little empirical evidence exists on their ability to handle spatial data with large numbers of areal units, thus complicating technical decision making in the development of online choropleth mapping applications. To fill this knowledge gap, this dissertation study conducts a scalability evaluation of three tile-based methods discussed in the literature: raster, scalable vector graphics (SVG), and HTML5 Canvas. For the evaluation, the study develops two test applications, generates map tiles from five different boundaries of the United States, and measures the response times of the applications under multiple test operations. While specific to the experimental setups of the study, the evaluation results show that the raster method scales better across various types of user interaction than the other methods. Empirical evidence also points to the superior scalability of Canvas to SVG in dynamic rendering of vector tiles, but not necessarily for partial updates of the tiles. These findings indicate that the raster method is better suited for dynamic choropleth rendering from large areal data, while Canvas would be more suitable than SVG when such rendering frequently involves complete updates of vector shapes.

ACKNOWLEDGEMENTS

This dissertation would have been impossible without the support of many people. First of all, I would like to thank my advisor, Luc Anselin. He has taught research fundamentals, has encouraged me to explore new fields, and has helped me find and stay on the right path. Without the incessant opportunities he has given, I would not be able to reach this far along my doctoral study. The members of my committee, Sergio J. Rey and Elizabeth A. Wentz provided valuable information and comments that helped improve the design and quality of this research. Dr. Rey, together with Dr. Anselin, allowed me to develop my professional expertise in spatial analysis and programming. Thanks to Dr. Wentz, I could enhance my perspectives to the process of scientific research and publications. Ming-Hsiang Tsou at San Diego State University and Shaowen Wang at University of Illinos at Urbana-Champaign (UIUC) have also shared great insights into the fields of Internet and Cyber GIS.

I also would like to thank the Graduate College and the Graduate and Professional Student Association at Arizona State University (ASU) for the fund and grant provided by a Dissertation Completion Fellowship and Graduate Research Support Program. These financial supports were essential for writing this dissertation and completing some parts of my study.

Julia Koschinsky, David C. Folch, Charles R. Schmidt, and Yan Liu deserve my special thanks. Julia has helped me in many different ways along these years. She was an insightful co-worker, a knowledgeable tutor, and sometimes my sister and mom in the United States (US). David has shared many of my joys and concerns that arose from my research and life in Arizona. Charles, a born programmer, has always been my first resort for troubleshooting multiple technical and programming problems. Yan Liu has provided a lot of guidance in seeing through the dark and bright sides of my life and research.

My sincere appreciation is in order to other graduate students and scholars I met at UIUC and ASU. Jong-Gun, Na-Young, Won-Kyung, and Min-Jo, you washed away my loneliness when I was homesick and in depression. Melinda and Stephanie, thank you for sharing the burden of my doctoral dissertation. Jae-Won, I appreciate your patience of hearing me out when I talked about nonsense out of despair. All hard workers at GeoDa Center of ASU and CyberInfrastructure and Geospatial Information Laboratory (CIGI) of UIUC, your passion have been very contagious. And I could learn how to pursue new things, ideas, and perspectives through diverse forms of collaboration with you guys.

Finally, I want to express my heartfelt gratitude to my family and friends in and from Korea. To my father, when I wanted to give up things you were always beside me. I could hear your conforting and encouraging voice. To my mother, you raised me to pursue dreams. I lived one and will continue to do so. To my sisters and brothers, without my belief in you, I neither could live away from you and mom nor cloud go through my life in US. To Enki, thanks for reaching out to me in difficult times. From you, I've learned how to live and think of our life. To Jung-Phil and Hae-Young, even an one-minute chat with you cheered me up completely. Your care and affection have been and will be indispensable to continuing my life journey.

TABLE OF CONTENTS

Page

Chapter Page

Chapter Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS and ABBREVIATIONS

AJAX Asynchronous Javascript And XML

API Application Programming Interface

DynTM Dynamic Tile Mapper

ESDA Exploratory Spatial Data Analysis

ESRI Environmental Systems Research Institute

FHL Frequency Histogram Legend

GeoJSON Geographic JavaScript Object Notation

GeoRSS Geographic Really Simple Syndication

GIF Graphics Interchange Format

GIS Geographic Information Systems

GIScience Geographic Information Science

GIServices Geographic Information Services

GML Geography Markup Language

GPS Global Positioning Systems

HTML HyperText Markup Language

JPEG Joint Photographic Experts Group

KML Keyhole Markup Language

LOD Level Of Detail

NASA National Aeronautics and Space Administration

OGC Open Geospatial Consortium

PARC Palo Alto Research Center

PCSA Primary Care Service Area

PNG Portable Network Graphic

PySAL Python Spatial Analysis Library

REST REpresentational State Transfer

RSS Really Simple Syndication

SE Symbology Encoding

SLD Styled Layer Description

SOAP Simple Object Access Protocol

SVG Scalable Vector Graphics

TIFF Tagged Image File Format

U.S. United States

VGI Volunteered Geographic Information

VML Vector Markup Language

W3C World Wide Web Consortium

WCS Web Coverage Service

WebGL Web Graphics Library

WFS Web Feature Service

WMS Web Map Service

WMTS Web Map Tile Service

XML Extensible Markup Language

ZCTA Zip Code Tabulation Area

CHAPTER 1

INTRODUCTION

1.1 Online Choropleth Mapping and the Challenge of Large Areal Data

The recent decade has witnessed dramatic changes in the production, provision, and use of online maps. Advances in information technologies, such as global positioning systems (GPS) and web 2.0 technologies, have allowed even casual users to partake in spatial data collection (Goodchild, 2007) and produce maps with a few mouse clicks (Plewe, 2007; Tsou, 2011). Maps from disparate providers can be easily shared through standardized web interfaces and mashed up to new value-added applications (Haklay et al., 2008; Batty et al., 2010). The primary use of online maps does not differ from that of paper maps, i.e., the communication of messages based on geographic information. However, many novel or renewed approaches to map use have also appeared on the web. For example, people use online maps to build social networks based on their current locations (Zheng et al., 2010). Online maps also have been utilized to disseminate georeferenced statistical information through web sites and mass media. Interactive analytical methods and advanced visualization techniques have been increasingly added to these statistical maps (Andrienko et al., 2010).

The progress in the analytical capabilities of online statistical maps is to a substantial degree the result of many research efforts in geographic information science (GIScience). For example, early studies such as those by Dykes (1998), Andrienko et al. (1999), Takatsuka and Gahegan (2001), and Anselin et al. (2004) enabled a wider range of users to easily access interactive tools over the Internet for exploratory spatial data analysis (ESDA) and geovisualization. More recently, researchers have incorporated these types of analytical tools into electronic atlases to improve the spatial exploration of statistical data that governments and public institutions share online (Tsoulos, 2005;

MacEachren et al., 2008; Rinner et al., 2011). User-centered design approaches in GIScience have also contributed to understanding how users work with interactive analytical tools available with online statistical maps (Cinnamon et al., 2009) and how the usability of such tools could be enhanced through changes in user interface and map design (MacEachren et al., 2008).

Among the many analytical methods discussed in the GIScience literature, choropleth mapping is arguably the most widely used technique for online statistical mapping (Armstrong et al., 2003). A choropleth map associates a color code with each geographic observation according to its attribute value. While "classless" methods for choropleth mapping directly map an attribute value to a color code (Tobler, 1973), most other approaches classify observations into several groups by considering the distribution of their attribute values (Slocum et al., 2009). This combination of data grouping and color shading, despite the need for careful decision making on these parameters (Brewer and Pickle, 2002), facilitates the identification of patterns in geographic variations of the mapped variable (Cromley and Cromley, 1996). In addition, most statistical data collected by governments and public institutions are aggregated by predefined enumeration units, making them amenable to choropleth mapping (Armstrong et al., 2003). Many online statistical mapping applications provide the capability of choropleth mapping. In particular, the recent generation of these applications supports dynamic choropleth mapping where output maps are not pre-generated, but are created on the fly as users interactively change design parameters related to data classification and color shading (Anselin et al., 2004; MacEachren et al., 2008; Jern, 2009).

Despite the continued growth of online applications that can support dynamic choropleth mapping, the applications today are limited in that they cannot deal with large areal data containing more than several thousand areal units (Zhao and Shneiderman, 2005; Schmidt and Dev, 2008). In general, it is difficult to dynamically create and display interactive geographic visuals from large spatial data (Gahegan, 1999). This is because the

time for processing and rendering data increases with the number of spatial objects included in the data (Guo et al., 2006). The challenge posed by large spatial data intensifies on the web, particularly because dynamic generation of geographic visuals and user interaction involve data transmission over the Internet (Rohrer and Swing, 1997). Despite this difficulty, there has been a growing demand for enabling dynamic choropleth mapping with large areal data (Gibin et al., 2008): high-resolution areal data, such as fine-grained census and cadastral databases, have become increasingly accessible and available for spatial analysis, at least in developed countries (Kwakkel et al., 2012); spatial problem-solving engines often produce analytical results measured at a type of areal unit; and real-time data from GPS and geospatial sensors are frequently aggregated into various forms of areal units for visual sense making of the data. To better address this demand, our understanding needs to be improved of existing approaches to online mapping of large spatial data and their applications to choropleth mapping.

1.2 Tile-based Choropleth Mapping and the Need for Scalability Evaluation

The first generation of online mapping applications, such as Yahoo! Maps and MapQuest, appeared in the mid 1990s and completely altered how spatial data were shared with the general public (Peng and Tsou, 2003). Despite the novelty, these applications were considered rudimentary due to their slow performance and limited support for user interaction (Sample and Ioup, 2010). In 2005, Google Maps employed tile-based mapping to overcome these drawbacks of online mapping applications. It provided "slippy map" by dividing pre-rendered maps into tiny tiled images, caching them on the server-side storage, and delivering them only if they became visible within the current map view (Quinn and Gahegan, 2010). Because this map tiling substantially reduces the amount of map data to be downloaded at a time, "slippy map" has improved user experience with map navigation (Haklay et al., 2008) and has gained popularity as a solution for efficient online delivery of large spatial data (Peterson, 2012b).

The latest literature on tile-based mapping suggests that map tiling can be used not only for fast delivery of static map images, but also for dynamic yet rapid creation of online choropleth maps from large areal data (Gibin et al., 2008; Mateos and O'Brien, 2011). While the general map-tiling approach is still applicable to online choropleth mapping, it is inappropriate for dynamic choropleth mapping since on-demand changes in map design parameters (e.g., data classification and color scheme) would yield large collections of output maps, even from one data variable (Armstrong et al., 2003). Recently, raster- and vector-based variants of the tile-based mapping method have been discussed in the literature to enable dynamic choropleth mapping of large areal data. In the raster-based approach proposed by Schmidt and Dev (2008), raster tiles are created after assigning unique color codes to individual areal units. New choropleth maps are then generated on the fly by changing the color codes assigned to the pixels of each tile. In contrast, in the vector-based approach geometric data are pre-processed to build vector tiles of varying levels of detail (Antoniou et al., 2009). Then advanced web graphics technologies such as Scalable Vector Graphics (SVG)[1] and HTML5 Canvas[2] are utilized for client-side rendering of the vector tiles (Gaffuri, 2012).

While these tile-based approaches seem to be promising solutions for addressing the data challenge in online choropleth mapping, one key question remains unanswered: *To what extent can they handle large areal data when supporting the dynamic rendering of and user interaction with online choropleth maps?* Answering this question is important in order to help researchers in GIScience select an appropriate tile-mapping method that matches the data requirements for their choropleth mapping applications. The wide use of choropleth maps in electronic atlases also suggests that answers to this question would

[1]SVG (Scalable Vector Graphics) is a language for describing 2D vector graphics that need to be displayed on web browsers (Neumann and Winter, 2005). It is based on extensible markup language (XML).

[2]"HTML5 is being developed as the next major revision of HTML (Hypertext markup language)" (Boulos et al., 2010, p.1). Canvas is a component of HTML5 that is key in the dynamic rendering of 2D vector shapes and bitmap images on web browsers (Boulos et al., 2010).

improve the effectiveness of choropleth mapping in these atlases, especially when they are used to visualize global spatial distributions of statistics measured at small areal units.

1.3 Research Overview and Significance

The purpose of this dissertation study is to empirically evaluate the scalability of three existing approaches to tile-based choropleth mapping: raster, SVG, and Canvas. Adopting the concept of visual scalability formulated by Eick and Karr (2002), the study defines scalability as the capability of tile-mapping methods to effectively render and interact with large areal data sets in terms of the number of areal units. For the evaluation, the study employs two test applications: one for the raster approach and another for the SVG and Canvas approaches. These test applications support an identical set of user interaction for choropleth mapping but utilize different map-rendering methods. Specific types of user interaction that are tested in the evaluation include dynamic choropleth map generation, map juxtaposition, dynamic data query, and map navigation. The study compares the scalability of the three tile-mapping approaches by measuring the time that the test applications took to update and render maps.

The evaluation in this study contributes to the literature of GIScience and cartography by providing empirical evidence for the scalability of the raster, SVG, and Canvas approaches in supporting dynamic, interactive choropleth mapping. To the best of this author's knowledge, the scalability of different tile-based choropleth-mapping approaches has not been benchmarked through empirical experiments. While specific to the test setups in the study, the results from the experiments demonstrate that the raster approach scales better than the other approaches and that HTML5 Canvas is more efficient than SVG for dynamic choropleth rendering of complete map tiles, but not necessarily for interactive updates of a subset of areal units. These findings suggest that although the tile-based approaches indeed facilitate dynamic choropleth mapping from large areal data, their scalability varies to a substantial extent. Empirical knowledge like this would allow map providers to make informed decisions on technological aspects of their mapping

applications. It also would help researchers in online mapping and geovisualization develop a clear understanding of the potential of different web graphics technologies.

The remainder of this dissertation is organized as follows: Chapter 2 provides a review of literature that concerns the concept and methods of choropleth mapping, the evolution of online mapping technologies, and traditional and tile-based approaches to online choropleth mapping. Chapter 3 presents the methodology used for the study. In particular, it elaborates on the implementation of test applications, the characteristics of test data, the process of tile generation, and the framework for the evaluation. Chapter 4 reports the results of the evaluation. Chapter 5 discusses the implications of the results for the research of online choropleth mapping in particular and web-based cartographic visualization in general. Chapter 6 presents conclusions, future research directions, and the significance of the study.

CHAPTER 2

LITERATURE REVIEW

The goal of the literature review is two-fold. First, it aims at providing a theoretical background related to tile-based choropleth mapping so as to place this dissertation study in the larger research context of choropleth mapping and online mapping. The first section thus provides an overview of how choropleth maps are designed. The review then moves on to elaborating the developments in online mapping. Because the evaluation of the study focuses on different structures and technologies for online choropleth-mapping applications, the second section discusses various issues concerning the design and implementation of online mapping applications such as architecture, map data models, geographic information services (GIServices), web 2.0 technologies, and map tiling. The second goal of the review is to identify research gaps in the literature of online choropleth mapping. To this end, the third section of the review examines connections between choropleth mapping and online mapping. It starts with conventional approaches to online choropleth mapping and contrasts them with emerging studies that apply map tiling to choropleth mapping.

2.1 Choropleth Mapping

Choropleth mapping is “a method of cartographic representation which employs distinctive color or shading applied to other than those bounded by isolines” (International Cartographic Association, Commission II, 1973, p.123). These are usually enumeration units at which the data are aggregated or computed. Examples include statistical or administrative areas, such as counties, states, census tracts, and block groups. In choropleth maps, the color value for an individual enumeration unit is associated with the magnitude of its data value that is considered to occur in the enumeration unit “typically” rather than “uniformly” (Slocum et al., 2009). The resulting cartographic representation

easily reveals visual patterns in the spatial distribution of the data values as a histogram provides a visual summary of a data distribution in one-dimensional space (Dent et al., 2009). This power of choropleth maps in pattern revelation is the main reason for their current popularity, along with the increasing accessibility of social and economic statistics aggregated by predefined jurisdictions (Armstrong et al., 2003). Advances in computer and web cartography have also contributed to the proliferation of choropleth maps by providing a wide range of users with easy access to map-making capabilities (Plewe, 2007).

2.1.1 Data Classification

Unarguably, one of the most critical concerns in choropleth map design is data classification. It involves the grouping of raw data values into classes where each is represented by a unique cartographic symbol. Since human perception is limited, this reduction of data variation and the consequent visual generalization tend to facilitate human perception of the spatial patterns displayed in choropleth maps (Dent et al., 2009). The use of data classification, however, has been criticized, as it causes some loss in data accuracy and burdens the map designer with the task of deciding appropriate methods and parameters for the data grouping (Slocum et al., 2009). As such, multiple studies have been conducted to answer the questions of whether data need to be classified, and if so, when and how the classification should be done. This subsection provides a brief review of those studies so as to explicate the rationale behind the rise of software applications capable of dynamic choropleth mapping.

When the first choropleth map was released in 1826, it was classless (Robinson, 1982). However, choropleth maps soon utilized data classification to enhance map readability. A scientific argument against the use of classed choropleth maps was initiated in 1973 by Tobler. He argued “It is now technologically feasible to produce virtually continuous shades of grey by using automatic map drawing equipment. It is therefore no

longer necessary for the cartographer to "quantize" data by combining values into class intervals" (p.262). Heated debates have been raised since then. As well reviewed by Slocum et al. (2009) and Stewart and Kennelly (2010), the debates centered on whether or not data classification facilitated information retrieval and pattern identification from choropleth maps (Muller, 1979, 1980) as well as the memorization of the resultant information and patterns (Dobson, 1980). Despite the debates, their findings are mixed and incomparable since existing studies did not account for or controlled differently confounding factors such as map-reading time (Gilmartin and Shelton, 1989) and the number of classes (MacEachren, 1982; Mersey, 1990). In view of these inconclusive findings from prior research, the issue of importance is not whether unclassed choropleth maps are better than classed ones, but under what conditions one may surpass the other, and when they may be used interchangeably. For instance, Cromley (2005) indicated the accuracy of a classed choropleth map could be improved by taking its unclassed design into account. Similarly but in the context of data exploration, Slocum et al. (2009) pointed out that combined applications of data grouping and ungrouping could facilitate exploration of data patterns, while classed choropleth maps would still work better for static delivery of geographic information.

Classed choropleth maps are widely used, despite the challenges from unclassed maps. Methods for data classification, therefore, have continued to draw attention in cartographic research. A theme of particular significance in this research concerns developing classification methods that can capture the statistical characteristics and spatial configurations of data values as accurately as possible (Armstrong et al., 2003). While conventional methods for data classification, such as equal interval and quantiles, are easy to compute and facilitate map interpretation and comparison (Brewer and Pickle, 2002), they often fail to maintain the numerical (e.g., grouping in data in the number line) and spatial structures (e.g., contiguity) in the data values (Slocum et al., 2009). Researchers in cartography have sought to address this problem by maximizing numerical homogeneity

and spatial contiguity of data observations within each class while also maximizing differences between classes both in the number line and spatial boundaries (Traun and Loidi, 2012).

Jenks and Caspall (1971) were the first to provide a comprehensive theoretical foundation on which data classification could be seen as optimization problems. The goal of optimization was to minimize intra-class variations in data values (i.e., tabular error), area-weighted data values (i.e., overview error), and differences in the data values of neighboring units (i.e., boundary error). This theoretical discussion was implemented by Jenks (1977) after Fisher (1958) had laid the mathematical foundation necessary to find optimal solutions. This implementation, however, did not consider the boundary error, thereby leaving the spatial contiguity goal unachieved. Multiple researchers in cartographic studies soon made explicit considerations of the unachieved goal of spatial contiguity in data classification. For example, Monmonier (1972) and Murray and Shyy (2000) used both numerical and spatial distances among data observations to find an optimal set of class intervals. Cromley and Cromley (1996) created virtual network graphs from data observations and used a shortest path approach to apply spatial contiguity constraints to data classification. Armstrong et al. (2003) generalized data classification as a multicriteria decision-making problem, while introducing new optimization goals such as area equivalency between classes and improved spatial association within each class.

The existing approaches to optimization-based classification are exploratory in the sense that the map designer still needs to experiment with classification parameters such as the number of classes and weightings for various optimization criteria. Their application might therefore seem complicated, especially when guidelines for design experiments and decisions are unavailable. Recently, Traun and Loidi (2012) pointed out the lack of clear guidance for determining weights for spatial constraints in data classification. The authors argued that global spatial autocorrelation in the observed data could be used for creating regions of similar data values, thus removing the need for user-chosen weighting for

optimization criteria. In a similar context but without taking an optimization approach, Jiang (2012) provided a method that recursively applied binary grouping to high values in the long tail of the data distribution until the sizes of the resulting two groups became balanced. This method was designed to spatially exhibit any hierarchical structure in the data values that followed a Zipf's-law-like rank-size distribution. Since the data grouping ended when the balanced group size was achieved, the method ostensibly showed breaks in the data values naturally without necessitating a selection of the number of classes.

Overall, the review thus far shows the variety of data classification approaches used for choropleth mapping. This variety indicates that choropleth mapping is by nature an exploratory rather than a deterministic process (Brewer and Pickle, 2002; Armstrong et al., 2003). That is, designing a choropleth map requires multidimensional exploration of the statistical characteristics and spatial context of data values. It also requires a careful comparison of how different classification solutions would affect the resulting map. The increase of software applications capable of dynamic choropleth mapping may be attributable partly to the growing recognition of choropleth map design as an exploratory activity. Additional grounds for this view can be found in prior studies related to the visual design of choropleth maps. The approaches and findings of these studies are discussed below.

2.1.2 Color and Legend Design

The visual design of choropleth maps are concerned with the specification of colors and legends. Colors are a means for the map reader to perceive variations in data values over space, and legends are the information source defining the mapping between numerical and chromatic spaces. The importance of color design in choropleth mapping has long been recognized, but the relevant theoretical principles and practical guidelines have resulted mostly from the research done by Brewer and her colleagues. Guidelines for legend design have been compiled as a part of guidance for general map design (Slocum

et al., 2009). Research in choropleth mapping has focused on designing legends that can inform the statistical aspects of the data values.

Research on color use in choropleth mapping has sought to develop and validate systematic frameworks for color selection with consideration to a wide range of map readers and uses. Emphasizing the importance of "matching the organization of the perceptual dimensions of color (hue, lightness, saturation) to the organization of data being represented" (p.55), Brewer (1999) provided a typology of color schemes that account for data characteristics. This typology consists of sequential, diverging, and spectral color schemes that are suitable for unipolar and ordered data, bipolar data, and scientific visualization, respectively. Brewer (1996, 1997) and Brewer et al. (1997) also developed color sets for these schemes and evaluated the effectiveness of and user preference for different sets of color schemes through controlled experiments. In these studies, it was highlighted that specific color schemes should be designed in such a way that they could accommodate map readers with color-vision impairments as much as possible (Olson and Brewer, 1997). The results from these theoretical and empirical studies produced a well-designed software tool, namely ColorBrewer, to guide novice map designers in color selection (Brewer et al., 2003; Harrower and Brewer, 2003). While not as actively as in the late 1990s, color theories and guidelines have been further developed from the work of Brewer and her colleagues. For instance, Mennis (2006) applied diverging color schemes to improve the visualization accuracy of various outputs from geographically weighted regression. Jenny and Kelso (2007) developed a general typology of common forms of color vision impairment and provided practical tips for color selection in map design.

Legends are the key to reading choropleth maps since they show the relationships between map colors and data values. Traditionally, the legends align the color boxes vertically or horizontally in conjunction with the logical organization of the data values. This conventional approach is often considered insufficient to clearly reveal the statistical

distribution of the data values. To provide more informative legends, Kumar (2004) proposed combining a frequency histogram with the standard color-box legend. This so-called frequency histogram legend (FHL), however, was sensitive to classification parameters (e.g., classification method and number of classes) and did not depict graphically both changes in class intervals and frequencies. Cromley and Ye (2006) resolved this problem by replacing the frequency histogram with an ogive cumulative frequency diagram. In contrast to FHL, the ogive legend was less sensitive to the classification parameters: any change in the classification parameters altered only the length of the color boxes on its horizontal axis and the cumulative frequency of data observations on its vertical axis. Cromley and Cromley (2009) demonstrated the usefulness of the ogive legend in the context of health disparity mapping. The authors enhanced the original univariate cumulative frequency diagram by including covariates of the mapped variable. As argued by Cromley and his colleagues, the ogive legend informed the map reader of the statistical distribution of data values within each class, although its informativeness was still limited to aspatial aspects of intra-class variations.

On the whole, the literature on choropleth map design suggests that the design process is a two-step exploratory activity where data characteristics are captured and generalized as patterns, and the patterns then graphically depicted in a humanly perceptible fashion. Associations in spatial and aspatial organizations of data values are hidden in the data; indeed iterative explorations through data classification may be required for the revealing and clear manifestation of the associations (Brewer and Pickle, 2002). Colors and legends also need careful design so that the uncovered associations can be easily grasped by human eyes. Previously, the exploratory nature of choropleth mapping was thought to interest only professional cartographers. However, the recent growth of software packages capable of dynamic choropleth mapping has encouraged not only expert cartographers but also domain scientists to investigate design parameters that best serve the goals of their mapping projects (Brewer et al., 1997). As will be discussed

below, the rapid developments in online mapping technologies are, once again, expanding the realm of choropleth map design even to citizen scientists and casual users (Andrienko et al., 1999; Tsoulos, 2005; Boulos et al., 2005).

2.2 Online Mapping

Online mapping or web mapping refers to the activities of designing, producing, distributing, and sharing online maps (Neumann, 2012). The advent of the Internet opened the door to exchanging map data, files, and graphics across distributed networks (Slocum et al., 2009). The World Wide Web has provided a virtual space where maps and mapping software programs are accessible to a wide range of users, while allowing for various types and levels of communication and collaboration among map providers and users (Plewe, 2007; Tsou, 2011). Since the first online appearance of maps and mapping applications, relevant technologies have advanced at a rapid speed and substantially altered how maps are produced, distributed, and used online (Peterson, 2012a). These continuing changes also have significant implications for online delivery of choropleth maps. This section examines central aspects of online mapping technologies and relevant research developments. It also discusses implications for choropleth mapping from advances in online mapping technologies and research.

2.2.1 Architectural Concerns: Server-side and Client-side Mapping

The online delivery of maps involves two layers of computing resources: the server and the client. Consisting of a single computer or a group of them acting in concert, the server hosts maps, map-generating applications, geospatial data, and other processes, and provides the user with access to those assets. The client is the web browser or similar software program running on the user's computer. It serves as an interface between the user and server by handling data requests and responses on behalf of the user. Since the distribution of the workload between the server and client has substantial impact on how maps are generated and used, research of online mapping has investigated different forms

of architecture design and evaluated their advantages and disadvantages. This subsection examines three general designs for an online mapping application–server-side, client-side, and hybrid mapping–and discusses how they can be used for choropleth mapping.

Server-side mapping refers to a software architecture in which the client side merely works as a terminal for invoking user requests and presenting maps, while the server side performs hefty tasks such as map generation, data processing, and analytical computation (Tsou and Buttenfield, 2002). This architecture was adopted in the early examples of online mapping applications like the Xerox Palo Alto Research Center (PARC) map viewer (Peng and Tsou, 2003), the first Internet Geographic Information Systems (GIS), and MapQuest, the first online routing system (Neumann, 2012). While not as simple as these precursors, contemporary applications for online mapping are also making intensive use of the server-side mapping architecture. In particular, applications that provide large maps and advanced analytical capabilities employ server-side mapping since high-end computer servers can accommodate the data storage and computational requirements of those applications better than client-side computers owned by individual users (Dent et al., 2009). Providing high-end computing capability for server-side mapping, however, comes at a cost. Online mapping applications may become less reliable since their functional stability is contingent on the availability and workload of the server-side computing resources (Huang et al., 2011). Even simple operations for user interaction, such as panning and zooming, require at least one "round-trip" of data between server and client, thereby lowering the responsiveness of mapping applications and increasing the amount of network traffic (Zaslavsky, 2000).

In contrast to server-side mapping, the architecture of client-side mapping assigns the server side to work merely as a data host, while the client side carries out all the map-associated activities (Tsou, 2004). This often requires a plug-in or native applications to add special functionality beyond the basic capabilities of the web browser (Fu and Sun, 2010). Typical add-ons include Java applet, ActiveX, Java Web Start, Adobe Flash, and

Microsoft Silverlight (Lienert et al., 2012). In client-side mapping, applications can work like a stand-alone software program and support fast user interaction once the necessary add-ons and data are downloaded and installed on the user's computer (Peng and Zhang, 2004). As such, applications designed for interactive analysis of small spatial data have often adopted the client-side mapping architecture. However, the reliance on add-on software and the time cost for application initialization have been criticized for causing usability problems with online mapping applications (Neumann, 2012). The heterogeneity in network bandwidth and client-side computing power also makes it challenging to guarantee consistent performance of the applications using client-side mapping (Huang and Lin, 2002).

An alternative to server-side and client-side mapping is a hybrid approach in which substantial workloads are distributed in a balanced way across the server and client sides (Fu and Sun, 2010). Although the equilibrium point in distributed workload differs among mapping applications, the client side usually undertakes data manipulation and visualization tasks that involve direct user interaction or the handling of user-provided data (Dent et al., 2009). In contrast, the server side performs computation-intensive tasks and the dispatch of commonly used spatial data such as base maps and data tables (Huang and Worboys, 2001; Brown et al., 2008). As will be discussed later, recent developments in service-oriented computing, web standards, and web 2.0 technologies facilitate the adoption of a hybrid architecture for the construction of an online mapping application. Distributed GIServices provide access to spatial data and operations that are hosted on remote servers (Li, 2008). The widespread support for standard-based service interfaces makes it easy to integrate various types of remote resources into a mapping application (Batty et al., 2010). Because of the popularization of web 2.0 technologies, many data providers are now offering mapping APIs for their geospatial data and services (Chow, 2008). Mapping APIs usually accompany client-side interface frameworks, but third-party

online mapping frameworks that provide advanced client-side functions for data handling, analysis, and visualization also abound on the web (Neumann, 2012).

The review has discussed three general approaches to architecture design for online mapping applications. While this broad categorization clarifies the basic structure of mapping applications, many other factors affect an appropriate architecture solution, such as goal, target users and data, anticipated user environments (e.g., bandwidth and client-side computer specifications), functional and nonfunctional requirements, and budget of a particular mapping application (Zaslavsky, 2000). In the context of choropleth mapping, server-side mapping is more suitable than the other architecture solutions for the online dissemination of static maps, on-demand generation of real-time maps, and integration of computation-intensive analytics with choropleth maps. If application developers want to enable the interactive design of choropleth maps, client-side or hybrid mapping will be of more interest than the server-side mapping approach. Another issue of importance in the choice of an application architecture is which map data model to use. This is because certain technologies geared for a map data model restrict the ranges of matching software architectures. The following subsection examines different map data models, relevant technologies, and their relation to architecture design.

2.2.2 Map Data Models: Raster and Vector Mapping

The encoding of map data is a crucial concern in online mapping. Two data encoding models have been employed: raster and vector. The raster model uses rows and columns of pixels to encode map data, while the vector model represents geographic features as discrete objects, encoding them as simple points, lines, polygons, and the like (Dent et al., 2009). Research in online mapping has focused on evaluating and improving the potential of these data models for web-based map visualization. In particular, it has investigated the feasibilities and capabilities of evolving web graphics technologies, and has proposed frameworks for harnessing those technologies for online mapping.

The most popular conventional approach to online map dissemination is to generate raster images from a digital or paper map and distribute them over the Internet (Esri, 2006). Data of scanned paper maps or automated mapping systems are usually installed on the server side, and the client side simply retrieves the map images from the server. Various formats for raster images exist, and those optimized for network transmission are intensively utilized for online mapping (Slocum et al., 2009). Examples include Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF), Portable Network Graphic (PNG), and Tagged Image File Format (TIFF). Using raster image formats for online map delivery has at least three advantages (Dent et al., 2009). First, raster images can be easily viewed in modern web browsers and embedded into web pages. Second, most mapping engines can export maps in a raster image format. Third, many software frameworks for online mapping support the handling of raster image formats; thus, the entry barriers to implementing a raster-based online mapping application are relatively lower than those for a vector-based application. Along with these advantages, the disadvantages of raster image maps are also well known. In comparison to vector map data, the graphical quality of raster image maps varies with the resolution of the image files. Low-resolution images are small in data size but tend to be poor in graphical quality (Cecconi and Galanda, 2002). On the other hand, high-resolution images can provide high-quality graphics but may increase the network traffic and response times of a mapping application (Peng and Zhang, 2004). Because geographic features cannot be identified in raster images, and any small changes to the images induce data exchanges with the server, user interactivity tends to be limited with raster image maps (Andrienko and Andrienko, 1999; Zhao and Shneiderman, 2005).

Dissatisfaction with raster-based mapping has resulted in various research endeavors to operationalize vector-based online mapping. The main idea underpinning vector mapping is to render geographic features directly on web browsers (Zaslavsky, 2000). Since geographic features in vector maps are represented as discrete objects with

their own geometric coordinates and attributes, the maps can afford a wider range of user interaction and higher levels of graphical quality when compared to raster images of similar resolutions (Neumann and Winter, 2005; Neumann, 2012). In addition, vector maps are free from resolution-related problems (e.g., pixelation) and can respond to user inputs, such as changes in cartographic symbols, without generating a lot of network traffic (Peng and Zhang, 2004; Boulos et al., 2005). Despite these advanced capabilities of vector maps, they have not been used as widely as raster maps, primarily for three reasons. First, vector graphics have not been well supported by web browsers (Dent et al., 2009). Special add-ons such as Adobe Flash and Java Web Start are often required to enable vector-based interactive mapping on web browsers (Steiner et al., 2002). Second, such add-ons are usually operational only on certain web browsers. This limitation may force users to work with specific web browsers regardless of their preference, thereby causing serious usability problems (Lienert et al., 2012). Finally, vector data tend to be voluminous when text-based encoding is used for data transmission. Although vector data can be downsized through precision reduction and compression, they usually take a longer time to download and render than raster map images of a similar resolution due to additional overheads caused by decompression, parsing, and drawing (Peng and Zhang, 2004).

Despite the downsides, vector mapping has continued to draw research attention in cartography and geovisualization. Early studies often utilized Java applets and Java Web Start to enable interactive vector mapping on the web (Andrienko et al., 1999; Andrienko and Andrienko, 1999; Takatsuka and Gahegan, 2001; Anselin et al., 2004). As new web technologies and standards became available for the description and rendering of vector graphics, multiple studies investigated their applicability to online mapping. For example, Zaslavsky (2000) developed a web map publishing kit called AXIOMAP where vector shapes encoded in extensible markup language (XML) were downloaded and transformed into graphic objects represented in vector markup language (VML) on web browsers. Steiner et al. (2002) used XML and Macromedia Flash to enable advanced techniques of

user interaction, such as linking and brushing, in online maps. By integrating open standards like geography markup language (GML), web feature service (WFS), and scalable vector graphics (SVG), rather than employing proprietary technologies, Peng and Zhang (2004) and Yao and Zou (2008) reported that they could improve both the interoperability of online mapping applications and the graphical quality of map products. Pavlicko and Peterson (2005) and Boulos et al. (2005) also demonstrated that SVG is appropriate for topographic mapping and interactive visualization of health data (e.g., dynamic choropleth mapping). Recently, a new vector graphics technology, HTML5 Canvas, has started to attract research attention as an enabler for client-side interactive vector mapping because of cross-browser support for the HTML5 standard (Boulos et al., 2010). Details of SVG, HTML5, and Canvas, the focus of this dissertation study, are further discussed in Section 2.3.2.

To summarize, the representation and encoding of map data are technical issues requiring multicriteria decision making. Raster map data may be efficient to produce and deliver, but impose limits on map interactivity, graphical quality, and application architecture (i.e., the use of server-side mapping). In contrast, vector map data may feature high levels of interactivity, graphical quality, and flexibility in the choice of application architecture (i.e., support for both server- and client-side mapping). However, practical technologies for vector web graphics are not as mature and widespread as those for raster graphics. Prior research suggests vector map data would be better than raster map data for dynamic choropleth mapping because they are superior in supporting interactive map symbolization. The evolving standards and technologies for vector web graphics indicate that the inefficiency issues with vector mapping, such as dependence on third-party add-ons and voluminous data size, may not serve as substantial hindrance in the near future as they have been until now. This potential, however, still needs to be proved through empirical evaluations of typical use cases of interactive choropleth mapping. This

study addresses this research need by conducting such an evaluation with special attention to the data capacities of various technologies for raster and vector mapping.

2.2.3 GIServices and Web 2.0

The software architecture and map data model constitute two foundational elements of any online mapping application. Early studies focusing on these core elements paved the way for recent innovations in online mapping in which maps, geographic data, and geospatial operations are shared in interoperable ways, remixed creatively into value-added applications even by amateurs, and augmented with volunteered geographical information. As discussed below, these innovations have brought dramatic changes in the production, use, and sharing of online maps, map data, and mapping applications. Underlying the changes is a new class of technologies that facilitate standardized interfaces between distributed geospatial data/software, end-user development of and pleasant user experience with online mapping applications, and user contribution of geographic data. This subsection examines this new generation of online mapping technologies and discusses its impacts on online choropleth mapping.

2.2.3.1 GIServices

Online mapping applications deliver raster or vector map data from a server to clients. While this traditional client-server architecture suffices for online map delivery, it does not allow maps or map-making capabilities of one server to be reused and shared by other software programs than its companion clients (Peng and Tsou, 2003). Provided the goal of online mapping is map sharing, it is obvious that the conventional client-server architecture restricts the scope and potential of reuse of online maps and even causes unnecessary redundancy in online maps and mapping applications (Fu and Sun, 2010).

Although not designed specifically for online mapping, geographic information services or GIServices have been adopted to overcome this limitation of the client-server architecture (Tsou and Buttenfield, 2002). In this study, GIServices refer to a type of web

services that provide online access to geospatial data, maps, and operations (Longley et al., 2005). GIServices allow remote users, either human or machine users, to access their core contents and functions by providing standardized interfaces and communication protocols (Tu and Abdelguerfi, 2006). In general, two approaches are taken to define access protocols for GIServices: representational state transfer (REST) and simple object access protocol (SOAP). The REST approach views GIServices as web resources that can be retrieved, modified, created, and deleted through HyperText Transfer Protocol (HTTP) request methods (Fu and Sun, 2010). In the SOAP approach, GIServices need to provide formal metadata of their capabilities and exchange messages with special XML encoding applied (Erl, 2005). Whichever access protocol is used, map servers can use GIServices to share maps and map-making capabilities with a broad range of users and achieve improved interoperability with various software programs. Of particular importance in enabling such interoperable use of online maps are standards for GIServices.

Open Geospatial Consortium (OGC) is a non-profit, international organization playing an active role in establishing standards for GIServices (Alonso et al., 2004; Tu et al., 2004; Barclay et al., 2006). OGC standards related to mapping GIServices include Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS), Styled Layer Description (SLD), and Web Map Tile Service (WMTS). The first three standards–WMS, WFS, and WCS–define interface rules for requesting map data in the form of raster images (OGC, 2006b), vector data (OGC, 2004), and raw coverage data (OGC, 2010), respectively. The fourth standard, SLD, includes a set of specifications for describing cartographic symbolization that needs to be applied to output maps (OGC, 2007b). GIServices in support of WMS, WFS, and WCS should implement this standard in order to allow dynamic cartographic symbolization. The last standard, WMTS, contains interface specifications related to the description, request, and delivery of map tiles, i.e., tiny rectangular pre-rendered map images cached on a map server (Batty et al., 2010). As will be detailed in Section 2.2.4, map tiles have gained popularity since Google started to

use them for fast and scalable map delivery on the web in 2005 (Peterson, 2012b). Since then, various definitions and interfaces for map tiles have been developed by commercial vendors and open-source development teams. WMTS is OGC's response to this new need and aims to harmonize diverse interface specifications for tiled map services (Masó et al., 2010).

GIServices for online mapping communicate with client-side programs or other services by exchanging data. For map data and their representations, OGC established four standards: GML, Keyhole Markup Language (KML), Geographic Really Simple Syndication (GeoRSS), and Symbology Encoding (SE). GML is a specialized XML vocabulary for describing properties of geographic information such as geometry, data attributes, coordinate reference system, topology, and coverage (OGC, 2007a). KML is also an XML-based language for describing spatial data. Unlike GML, it includes constructs for specifying both data and visualization properties (OGC, 2008). Really Simple Syndication (RSS) is a set of data feed formats for publishing frequently updated information such as blog entries and news headlines (Fu and Sun, 2010). GeoRSS is a standard for encoding geographic location information included in RSS and other XML data feeds (OGC, 2006a). Finally, SE is a standard language used to express grammars for styling map data (OGC, 2006c). It extends XML and is used to formulate SLD documents that describe cartographic symbolization for output maps.

OGC standards for GIServices, including those described above, have gradually increased and diversified. This growth of standards may indicate expanding acceptance of their usefulness in both software markets and real-world practices. Indeed, a good portion of open-source GIS has been developed to enable online mapping capabilities in compliance with OGC standards (Li, 2008; Steiniger and Hunter, 2013). Recently, the uptake of those standards has been improved considerably even in commercial domains. Well-known commercial server products such as ArcGIS Server (Esri, 2010) have begun

to support a core set of OGC standards parallel to their open source counterparts (e.g., MapServer and GeoServer).

Along with the increasing software support for the standards, numerous accounts have been published to report the benefits of standards-based online-mapping applications. For instance, Zhang and Li (2005) reported that WMS and WFS could facilitate real-time geospatial data sharing in time-critical applications, such as emergency response and traffic management, by simplifying data integration. In fact, the latest online systems for disaster management, such as the Sahana system (Careem et al., 2007) and an Indian flood monitoring system (Karnatak et al., 2012), use standards-based GIServices extensively for the rapid collection and integration of heterogeneous spatial data. Similar endeavors can easily be found in public health, as demonstrated by online mapping applications for infectious disease surveillance (Gao et al., 2008), health exposure assessment (Evans and Sabel, 2012), and thematic explorations of health data (Moncrieff et al., 2013).

The standards and technologies of GIServices have advanced substantially over the past fifteen years or so. Online maps and the geospatial data behind them are now accessible from a variety of client programs through GIServices, yet the access protocols and data representations are standardized, varying little from one client to another. This service-oriented, standards-based approach has catalyzed the online sharing and integration of spatial data and maps across various domains and is gaining momentum in wide-ranging software markets. In the context of choropleth mapping, the developments in GIServices have multiple impacts. Access to online choropleth maps and spatial data amenable to choropleth mapping have improved much since the introduction of GIServices. As distributed spatial data can be easily integrated even in on-demand fashions, dynamic choropleth mapping from the integrated data is increasingly being requested. OGC standards for cartographic symbolization (e.g., SE and SLD) now allow for the standardized application of interactive choroplethic symbolization in a limited range of data classification methods. However, fundamental knowledge still remains to be

gained about what kinds of technological solutions are available to address these rising needs for dynamic online choropleth mapping and how suitable they are in different contexts of choropleth map use.

2.2.3.2 Web 2.0 and User-centered Geospatial Technologies

Advances in mainstream web technologies have significant influence on developments in online mapping. An obvious example of this is recent innovations brought about by the web 2.0 paradigm. Coined by Tim O'Reilly, the term web 2.0 refers to a collection of strategies for better harnessing the new nature of the Internet, i.e., being a platform for many things (O'Reilly, 2005). Online applications in this web 2.0 era differ from their precursors in that they invite the user to participate in content design, generation, and sharing (Goodchild, 2007). The applications are also distinctive as they provide end users a suite of relatively easy-to-use development toolkits by which the users can integrate various data and information to create their own novel applications (Batty et al., 2010). Online mapping technologies and applications have served as key drivers in the spread of the web 2.0 paradigm. This in turn has led to profound changes in the development, distribution, and use of online maps through improved user experiences, the involvement of amateur geographers (or neogeographers), and the massive production of user-generated geographic information (Tsou, 2011).

Three technologies underpin the web 2.0 revolution in online mapping: Asynchronous Javascript And XML (AJAX), mapping API, and tiled mapping. AJAX refers to a suite of technologies for seamless data exchange between a server and clients without refreshing the entire web page (Zucker, 2007). AJAX allows the user to interact with a web application with little interruption due to webpage reloads. The resulting reduction of wait time helps improve the user experience in online mapping (Tsou, 2005). The provision of mapping APIs is another feature of online mapping services in the web 2.0 era (Chow, 2008; Haklay et al., 2008). When private companies such as Google and

Microsoft started to launch online mapping services, they also provided programming APIs in order for end users to develop custom applications. These APIs are relatively easy to learn and use, and facilitate end-user development by providing prebuilt graphical user interfaces and free access to a multitude of high-quality spatial data and efficient geoprocessing services such as data overlay, geocoding, and routing (Gibin et al., 2008; Peterson, 2012a). Tiled mapping, the focus of this dissertation study, refers to a technique which improves the efficiency of online mapping by using small rectangular map images that are pre-split hierarchically along a fixed set of map scales (Sample and Ioup, 2010). As discussed in Section 2.2.4, tiled mapping enhances the performance of online mapping applications by returning only the tile images in current viewport from a remote cache (Adnan et al., 2010). Combined with AJAX, it improves the responsiveness of mapping applications, thus contributing to pleasurable user experiences in online mapping (Tsou, 2005).

Encompassing the enabling technologies for web 2.0, open source and commercial GIS communities have provided a wide range of mapping APIs and software frameworks for client-side map interface and server-side data provision. For example, the Open Source Geospatial Foundation has been supporting multiple online mapping projects such as OpenLayers and MapFish,[1] creating a bridge between the web 2.0 and GIServices approaches. ESRI and CloudMade are now providing APIs and services to access their private data repositories and public geospatial data sources. CloudMade offers several developer tools to access OpenStreetMap data,[2] and ESRI provides online mapping platforms where user contents are hosted and curated as web maps and applications,[3] not to mention APIs for Flex, Javascript, Silverlight, and mobile native applications.[4] These

[1] http://www.osgeo.org/

[2] http://cloudmade.com/products

[3] http://www.esri.com/software/arcgis/arcgisonline

[4] http://www.esri.com/getting-started/developers/get-started

new tools for online mapping have enabled end users to create novel mapping applications by integrating diverse types of distributed geographic data and services with high-quality base maps (Batty et al., 2010). Called map or spatial mashups, these user-developed applications have gained popularity in recent years and advanced from simple overlays of data layers to interactive virtual spaces for data transactions, geovisualization, geocollaboration, and geosocial activities (Elwood et al., 2012).

Two other technologies that relate strongly to web 2.0 online mapping are virtual globes (or geobrowsers) and global positioning systems (GPS). Virtual globes are software environments where the user can interactively explore 3D representations of the Earth (Butler, 2006). Examples include Google Earth and World Wind from the National Aeronautics and Space Administration (NASA). In these environments, users can freely travel across the Earth's surface, change their viewpoints flexibly, and overlay distributed geographic data layers on the surface (Haklay et al., 2008). Virtual globes deliver realistic displays of geographic environments, provide immersive user experiences, and can afford 3D geovisualization (Schultz et al., 2008). These unique features have made virtual globes a major consumer of interactive online maps of various geospatial themes. *GPS*, another crucial technology in the web 2.0 era, refer to navigational systems that use satellite signals to determine the location of a radio receiver on or above the Earth's surface (Grewal et al., 2007). Nowadays, GPS are embedded in most smartphones that are capable of attaching geographic coordinates to various contents they create, such as photos, videos, blogs, and social media messages. This availability of GPS and standard data formats have allowed citizens to contribute digital spatial data voluntarily (Turner, 2006) and opt to provide spatial data on their activities (Elwood et al., 2012). The resulting new data, termed *volunteered geographic information* (VGI) by Goodchild (2007), have been used for various purposes, such as digital map creation (e.g., OpenStreetsMap), place-based knowledge acquisition (e.g., WikiMapia), geo-marketing, and geosocial networking (e.g., FourSquare).

The recent proliferation of VGI has impacted the practice and research of online mapping significantly. One apparent effect of VGI pertains to the increasing mass-production of user-centered online-mapping applications (Elwood et al., 2012). Online services for VGI collection often employ computing platforms where user-contributed data are automatically mashed up with other geographic data and methods. For example, Flickr allows the user to upload and create online maps of photos geotagged with references to framework geographic data such as satellite images and road maps. Google Map Maker provides the user with GIS-like tools for adding and editing the digital data of geographic features of interest. The rapid increase of these consumer- or citizen-oriented mapping applications initially raised concerns about the lack of opportunity to interlink the expertise of professional cartographers to user-centered map production (Plewe, 2007). These concerns are still valid in the sense that new frameworks are needed to better inform and guide the design and development of mapping applications for producing, sharing, analyzing, and utilizing VGI (Tsou, 2011; Jones and Weber, 2012). Some efforts to address these concerns have been made in the latest services for user-centered online mapping: cartographic theories and practices have been implemented as software tools that can aid lay users in the design of map styles and cartographic symbols (Schmidt and Weiser, 2012).

The rise of VGI-driven mapping has also called attention to the issues of data quality, ownership, and confidentiality. The amateur-driven, participatory, and asserted nature of VGI indicates that data products may lack accuracy, completeness, consistency, or reliability (Batty et al., 2010). While VGI data are indeed subject to a wide range of uncertainties, recent studies suggest that volunteered framework data such as OpenStreetsMap are actually as accurate as datasets from Ordnance Survey (Haklay, 2010); moreover, the currency and timeliness of such volunteered data could render them more useful than conventional spatial databases in special cases like disaster response and recovery (Goodchild and Glennon, 2010). Delicate legal and ethical issues of ownership

and confidentiality related to VGI remain largely unresolved and under-investigated (Elwood et al., 2012). As VGI production may rely heavily on spatial data from authoritative or private institutions, the legal rights of VGI need to be carefully established. Some VGI may concern individual-level activities and could raise issues of privacy invasion and confidentiality violation (Goodchild, 2007). In the context of online mapping, these concerns of data quality and safety present challenges of how the relevant information should be represented or addressed in the design, production, and dissemination of VGI-based online maps without imposing many restrictions on the creative use of the volunteered information.

In summary, the key innovation in web 2.0 online mapping is user-centered design of maps and applications. User-centered design here exceeds user-friendly design or usability improvements. It means the construction of virtual map-based spaces where users can play with, work with, contribute, share, and create geographic data, information, and knowledge. Advances in and the seamless integration of geospatial web technologies have enabled the transformation of online maps from one-directional media of geographic information to multi-directional web platforms where any user can be a *prosumer* of geographic information. This paradigm shift in online mapping has multiple implications for web-based delivery of choropleth maps. Although early versions of mapping APIs lacked capabilities for advanced analytical mapping (Gibin et al., 2008), choropleth mapping has been increasingly supported in their recent releases (Schmidt and Weiser, 2012). One anticipated result of this would be the increased engagement of end users in choropleth map design by means of interactive spatial mashups. To help users design quality choropleth maps, cartographic knowledge needs to be shared in the form of a software tool that can serve as a guide. Technological solutions for dynamic online choropleth mapping also need to be further developed to accommodate a broader range of use cases for direct map design by end users.

2.2.4 Tile Mapping

Improved usability is one of the most frequently noted features of web 2.0 online mapping applications (Haklay et al., 2008). As discussed earlier, two enabling technologies underly this advancement: tile mapping and AJAX. In tile mapping, maps are first pre-rendered for a fixed set of map scales and then divided into a series of small map tiles in a hierarchical fashion (Peterson, 2012b). The resulting map tiles are stored on the server-side cache and indexed for fast tile searches (Sample and Ioup, 2010). When the user requests a map, tiles for the current viewport are retrieved from the cache and re-assembled on the web browser. This approach of tile-based data partitioning has long been used for the efficient management, storage, and retrieval of large map data (Quinn and Gahegan, 2010). Its recent combination with AJAX, however, has brought dramatic changes to the user experience in online mapping by allowing for fast map updates and continuous user interaction by reducing the need for on-demand map rendering and complete page reloading (Tsou, 2005). Providing some background for the tile mapping approaches, this section prepares the way for the discussion of tile-based choropleth mapping.

Tiling is a common strategy for partitioning paper maps of large areas and voluminous spatial databases (Goodchild, 1989). Its application to online mapping, however, is a recent phenomenon. Microsoft TerraServer used a pyramid-like tiling scheme to deliver high-resolution imagery data on the web around 2000 (Barclay et al., 2000, 2006). In this scheme, the number of tiles grows with the map scale, often quadratically. The details of the map diminish as the map scale decreases, because an upper layer in the map pyramid is usually constructed by generalizing lower ones. While this initial effort had drawn the attention of GIS and remote sensing professionals, it was after 2005 when Google launched its online mapping service that tile mapping gained popularity in a wider range of communities (Sample and Ioup, 2010). Three reasons exist for the success of Google's tile mapping approach. First, tiled maps were developed not

only from imagery data but also from vector databases containing complex road data. Instead of pyramid-based resampling of raster images, careful map generalization was applied to create images of pre-rendered road maps at different map scales (Quinn and Gahegan, 2010). Second, Google used AJAX for online tile delivery so that the user could obtain updated maps in a responsive manner without experiencing interruptions in map interactions (Tsou, 2005). Third, Google provided mapping APIs through which the user could reuse map tiles to develop custom applications (Chow, 2008).

Most online-mapping services, including Google Maps, currently provide map tiles generated from raster images of pre-rendered maps. Map rendering starts with a projection of source data. The spherical Mercator projection is widely used because it facilitates map partitioning by creating a 2D map of the Earth with a 1-to-1 horizontal to vertical aspect ratio (Sample and Ioup, 2010). Maps are usually pre-rendered since map rendering takes a substantial amount of time, especially for remotely sensed data (Tsou, 2005). The time cost for map rendering is smaller for vector data. However, on-demand tiling is often used only once when the tiles are initialized from rasterized vector features (Quinn and Gahegan, 2010). Then they are cached to prevent repeated map rendering. The number of map scales at which maps are pre-rendered is often limited to about 20 for the world map. This is because as the number of map scales increases, so does the number of map tiles, which could in turn incur tremendous costs for data storage infrastructures (Peterson, 2012b). Maps, once rendered, are partitioned into tile images, generally by using a quadtree scheme. Sample and Ioup (2010) recommended JPEG and PNG (among others) for the image file format and 256×256 or 512×512 pixels for the tile size. JPEG is proper for tiles of remotely sensed or very colorful imagery, while PNG is better for tiles with a smaller number of colors or when map transparency needs to be adjusted. When the recommended tile sizes are used, an optimality can be achieved between wasted tile pixels and increased network traffic due to the use of tiles.

Map tiling has been employed not only in raster-based mapping but also in vector-based mapping. Use of tiled vector data for online mapping has been studied at least since 1999 when Wei et al. (1999) introduced a scheme for tile-based data partitioning and transmission. Since then, research on client-side vector mapping has occasionally discussed the application of tiling to vector databases as a performance improvement strategy (Tu et al., 2001; Neumann and Winter, 2005; Campin, 2005). Recently, research interest in vector tiling has been rekindled due to the advancements in vector web graphics (Gaffuri, 2012), as elaborated in Section 2.2.2.

Conceptually, the tiling of vector data does not differ from its raster counterpart. Vector data are processed to have an adequate level of detail (LOD) at a given map scale. The resulting data are then divided into tiles for rapid data transmission over the Internet. Operationally, vector tiling differs from raster tiling in four aspects: control of LOD, data partitioning, content, and client-side rendering (see Figure 2.1 for illustration). The LOD of a map varies with the map scale so that the map can maintain a similar level of information density across different map scales (Gaffuri, 2012). Control of the LOD is accomplished by decreasing the resolution in maps of raster images. Since the resolution concept does not exist in vector data, detailedness in vector tiles is adjusted through generalization (Neumann and Winter, 2005). Map generalization here can be purely geometric, i.e., the reduction of geometric detailedness through vertex removal (Cecconi and Galanda, 2002; Yang et al., 2007); or, it can change both geometric detailedness and the geographic phenomenon represented by the data (Bertolotto and Egenhofer, 2001). For example, points representing trees can be transformed into polygons representing forest boundaries when the map scale decreases by one level. To support this kind of scale-sensitive map representation, vector tiling often utilizes multiscale databases where vector data are stored in such a way that incremental changes in data representation across map scales can be easily computed (Kilpelainen, 2000; Ramos et al., 2009).

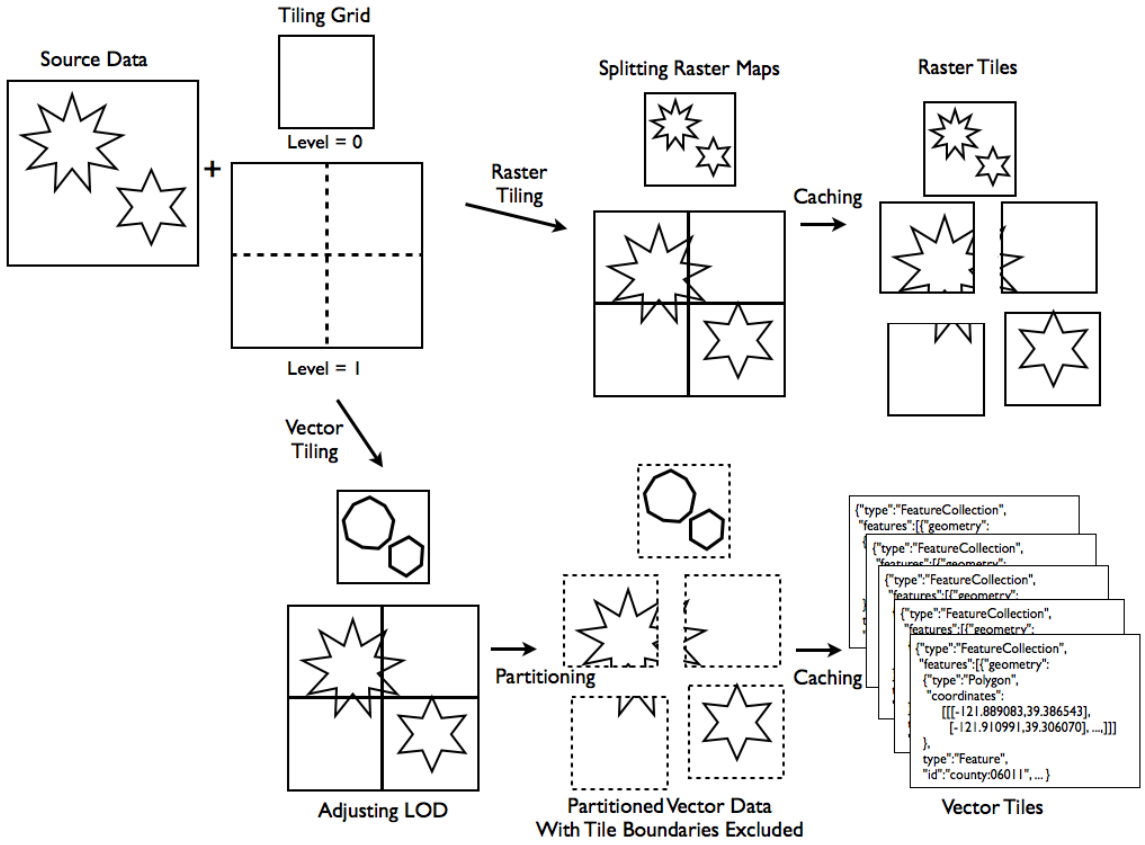


Figure 2.1. Illustration of raster and vector tiling

Two approaches can be used to partition vector data. First, vector data can be divided into tiles as in raster tiling. In this case, geographic features that belong to multiple tiles are split into several parts (Campin, 2005; Langfeld et al., 2008). An alternative approach is to use tiles as a spatial index. Here, a tile becomes a bounding box for querying geographic features (Gaffuri, 2012). Queries can be designed in such a way that each geographic feature belongs to only one tile at a given map scale.

In terms of content, vector and raster tiles differ in that the former contain data about geometric representation, attributes, and rendering information of geographic features while the latter do not (Gaffuri, 2012). As elaborated in Section 2.2.2, this difference in content has substantial influence on the size and file format of map data as well as on how user interaction is supported in the final map.

Lastly, vector and raster tiling are dissimilar as to how tiles are rendered on the client side. Raster tiles can be easily stitched on a web page simply by aligning them in an appropriate order. In contrast, geometries in vector tiles need to be drawn on the web browser, and if they are partitioned across multiple tiles, union operations need to be performed on the downloaded geometries in order to maintain the topological relationships in the original data (Antoniou et al., 2009). Downloading and drawing vector graphics may take more than one or two seconds, a threshold online users are known to tolerate (Nah, 2004). Progressive transmission and rendering of vector graphics can be used to reduce negative user experience (Cecconi and Galanda, 2002; Yang et al., 2007). Unlike raster graphics, for which many progressive rendering techniques are available, custom client-side map interfaces need to be developed to support the incremental drawing of vector geometries (Gaffuri, 2012).

Tiles, either in raster or vector format, are often cached in a server-side data store, since generating tiles for multiple map scales takes substantial time, computing power, and disk space (Peterson, 2012b). File-based storage, relational databases, and memory-based cache can be used with an efficient scheme for tile indexing (Sample and Ioup, 2010). When it is expected that only some portion of a data layer will attract user attention, selective tiling can be employed so that only frequently viewed tiles are cached in advance for fast response, while infrequently requested map tiles are created on demand. To inform selective tiling, tile usage can be monitored (Fisher, 2007) or predicted based on the geographic configuration of a study area, such as the spatial distribution of a residential population and road network (Quinn and Gahegan, 2010). Techniques for data compression are also of interest in server-side caching since they help to reduce the size of tile storage and network traffic (Yang et al., 2005). Lossy or lossless techniques for image compression can be used to reduce the size of raster tiles (Dent et al., 2009), while techniques for XML and payload compression (e.g., binary XML and gzip) can be employed for vector tiles (Lienert et al., 2012; Gaffuri, 2012).

In addition to server-side caching, client-side caching of downloaded map tiles can enhance the responsiveness of a mapping application and decrease the usage of the network bandwidth for tile delivery (Zhao and Shneiderman, 2005). Since most web browsers are designed to cache image files, client-side caching of raster tiles is straightforward. In contrast, client-side caching of vector tiles requires careful schemes of data management, since the size of the browser cache is limited and vector maps tend to be larger than raster maps of a similar resolution (Ramos et al., 2009). New technologies for client-side caching, such as local storage and browser-embedded databases, present promising avenues for the distributed storage of map tiles between the client and the server, which would bring performance improvement in online mapping (Boulos et al., 2010).

To summarize, tile mapping has contributed to enabling efficient online delivery of large map data, thus improving the user experience with interactive online-mapping applications. Raster tiles are predominant in the current web, but advances in vector web graphics have increased the interest in vector tiles. It is common in raster and vector tiling to use cached tiles that are hierarchically partitioned across multiple map scales. However, the two tiling methods differ in operational aspects of tile generation and representation, such as the control of LOD and tile content. The developments in tile mapping suggest that online applications for choropleth mapping could also benefit from tile-based map delivery. Tile mapping would be particularly useful when on-demand rendering of choropleth maps takes non-trivial time, thus hindering interactive map exploration. Tile-based choropleth mapping, however, presents challenges when choropleth maps need to be generated dynamically in accordance with user-defined design criteria, since dynamic adjustments in map design prohibit map pre-rendering. As will be discussed below, various approaches have recently been proposed to enable dynamic choropleth mapping with map tiles.

2.2.5 Summary

The review in this section focused on past developments in online mapping technologies and shows a paradigm shift from enabling online delivery of geographic information to empowering end users to engage in the entire life cycle of geographic information development and use. Architecture design and the map data model have been critical concerns in online mapping and will continue to be such. Recent advances in GIServices and web 2.0 technologies have provided various means for end users to produce, contribute, consume, share, and integrate geographic information such as standards, mapping APIs, and software frameworks for spatial mashups. Tile mapping has served as a key catalyst in this evolution by allowing online mapping applications to respond to user interactions quickly, leading to on improved user experience. These advances in online mapping imply that interactive design of choropleth maps will be increasingly requested by end users and that dynamic yet responsive choropleth mapping will be frequently required for the visualization of geographic information that is integrated on demand. The progress in tile mapping suggests that tiled maps may also result in enhanced responsiveness in online choropleth-mapping applications, especially when the areal data to be mapped are large and thus difficult to render in real time. The research in this dissertation investigates arising technological approaches for addressing these new demands in online choropleth mapping and evaluates their relative strengths in handling large areal data.

2.3 Online Choropleth Mapping

Online choropleth mapping refers to the activities of designing, producing, distributing, and sharing choropleth maps over the Internet. Since the advent of online mapping technologies, online choropleth mapping has drawn substantial attention in the cartography literature due to the popular use of choropleth maps across various domains. Research initially focused on the application of different architecture designs to

choropleth mapping. Lately, the goal has been to increase data capacity by using map tiles for the dynamic delivery of choropleth maps while supporting high levels of user interaction. This section first reviews the traditional approaches to online choropleth mapping. It then moves on to examine tile-based solutions for scalable choropleth mapping and highlights the need for empirical assessments of their actual scalability.

2.3.1 Traditional Approaches

Choropleth maps are effective means for revealing patterns in geographic variations measured at areal units (Slocum et al., 2009). The online delivery of choropleth maps has gained momentum as governments and public institutions have provided citizen users intuitive spatial interfaces to the statistical databases that are often collected for administrative jurisdictions (Armstrong et al., 2003). An early example includes the online thematic mapping interface developed for the 1991 United Kingdom Census of Population (Andrienko et al., 1999). In this interface, interactive choropleth maps provided the basis for for spatial exploration of population distributions and advanced visualization capabilities such as dynamic querying and linking with statistical charts. Similar approaches have been used until recently to develop electronic atlases and interactive interfaces for special types of statistics. For instance, Tsoulos (2005) added choropleth mapping capabilities to an electronic version of the Statistical Atlas of the European Union, and Boulos et al. (2005) employed interactive choropleth maps to facilitate geovisualization of health statistics. In the latter study, the authors emphasized the importance of interactive choropleth mapping by asserting that the online sharing of such a capability could help to better inform decision makers of the accessibility and effectiveness of health facilities and programs.

Underlying the popularization of online choropleth maps are developments in online mapping technologies. Advances in server-side, client-side, and hybrid mapping as well as in raster and vector graphics have strongly influenced how online choropleth maps

are distributed on the web. Early applications for online choropleth mapping could be categorized largely into two groups: those using raster-based server-side mapping and those using vector-based client-side mapping. As discussed in Section 2.2, both approaches have strengths and weaknesses: raster mapping applications are efficient in map generation but limited in supporting high levels of user interaction, while vector mapping applications allow for various types of user interaction but often require initial downloading and the installation of third-party add-ons and map data. Recent developments in GIServices and mapping APIs have remediated some of the weaknesses in each approach. For example, OGC SLD and related software tools now allow for dynamic cartographic symbolization in raster mapping applications (Evans and Sabel, 2012; Moncrieff et al., 2013), while mapping APIs and web mapping frameworks provide map interfaces that can render vector graphics directly on web browsers without using third-party extensions (Neumann, 2012).

Despite the advances in online choropleth mapping, it has been frequently reported that interactive choropleth mapping of large areal data poses challenges for both mapping approaches (Steiner et al., 2002; Zhao and Shneiderman, 2005). In interactive mapping, choropleth maps need to be updated on demand to respond to user interactions. When the maps are created from large areal data, such dynamic map generation often incurs a time cost that online users are unwilling to tolerate (Gahegan, 1999; Guo et al., 2006). This drawback of the conventional approaches has become increasingly problematic in the recent past as areal data amenable to online choropleth mapping have grown rapidly across various spatial scales. Governments and public institutions have made considerable contributions to this data increase by establishing or improving information systems for collecting and sharing geo-referenced statistics (Kwakkel et al., 2012). Big data coming from diverse sources such as social media and search engines often include geotags and are aggregated to various geographic units for spatial sense making of the data (Crampton et al., 2013). When such aggregation is done at a fine spatial resolution, the resulting area

data become large both in data size and number of records. These days, data aggregation and subsequent visualization are often requested in real-time fashion, but the conventional approaches to online choropleth mapping have difficulty addressing such requests due to their limits in handling large areal data.

In response to the challenge, Zhao and Shneiderman (2005) developed a variant of the deferred shading method in which all geometries are rendered first and color shading is applied later to image pixels through computations (Deering et al., 1988; McReynolds and Blythe, 2005). In particular, the authors proposed to use a cache of color-coded raster maps that were pre-rendered on the server side in such a way that the color code for each pixel represented the identifying number of a spatial object in the source data. A Java applet containing the cached raster maps was then downloaded to the client side, where they were dynamically recolored as the user specified the parameters for choropleth mapping (Zhao and Shneiderman, 2005). This approach allowed for client-side real-time updates of the choropleth maps and could deal with census data as large as United States (U.S.) counties with little increase in the size of the maps. Nonetheless, the proposed approach was limited. The performance of client-side map updates relies on the computing power of the client-side machines (Schmidt and Dev, 2008). In addition, downloading raster maps may still increase users' wait time if the maps cover a large spatial extent and contain multiple map layers designed for different map scales.

2.3.2 Tile-based Approaches

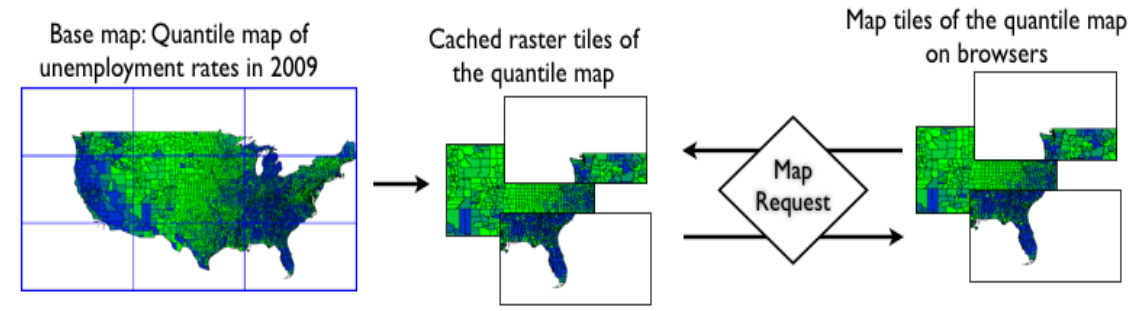
Lately, several studies have adopted tile-based mapping to enable choropleth mapping of large areal data. For example, Gibin et al. (2008) created a collection of raster tiles from pre-rendered choropleth maps by adopting the raster tiling approach (Figure 2.2a). This approach, however, was ineffective for dynamic choropleth mapping. Unlike base maps that are mostly static, choropleth maps can have numerous representations according to user-defined parameter settings such as the variable to be mapped, data classification

method, number of classes, and color scheme (Armstrong et al., 2003). Thus, it is difficult to pre-generate tiles for all possible choropleth maps unless map providers delimit the range of the supported parameters. Aware of this problem, Mateos and O'Brien (2011) chose to have both rendering and tiling performed on demand (Figure 2.2b). This way, the authors could avoid caching numerous tiles in advance but could still allow the user to interactively create new choropleth maps.

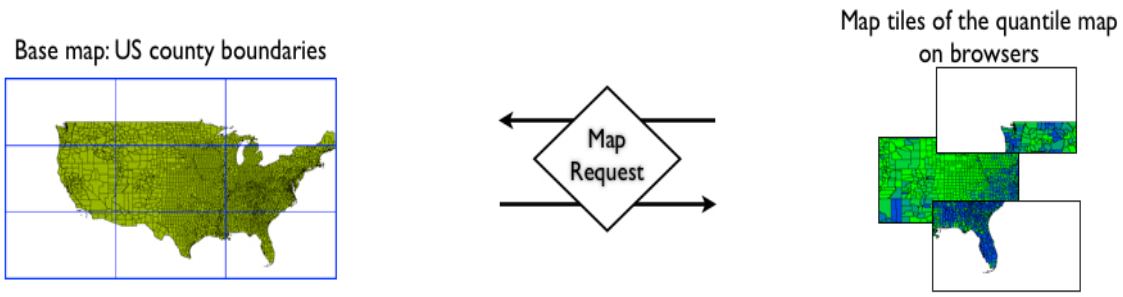
In spite of its flexibility, the approach of Mateos and O'Brien (2011) was computationally intensive. Schmidt and Dev (2008) proposed a solution for this problem by combining the ideas of tile mapping and color-coded raster maps that were initially put forward by Zhao and Shneiderman (2005). In this new proposal, color-coded raster maps were pre-generated, tiled, and cached on the server (Figure 2.2c). As the user specified the parameters for choropleth mapping, the tiles were recolored on the server side and transmitted to the client side (Schmidt and Dev, 2008). Because the tiles were small and had a fixed dimension, they could be dynamically recolored and downloaded in a short amount of time. While this dynamic re-rendering of color-coded raster tiles had the potential to enable scalable choropleth mapping, the approach was still subject to several drawbacks such as limited support for direct user interaction with geographic features and poor graphical quality of output maps in relation to vector maps of equivalent LODs.

An emerging alternative to raster-tile-based choropleth mapping is the use of vector tiles. For example, the Polymaps library[5] supports the dynamic creation of choropleth maps by rendering vector tiles as SVG and Canvas elements on the client side (Figure 2.2d). This vector-based approach aims to provide highly interactive and visually appealing choropleth maps from large areal data (Azzi et al., 2011; Yau, 2011). To do so, it seeks to harness the latest developments in web graphics technologies and map generalization (e.g., geometry simplification). Clearly, this vector-based approach can help

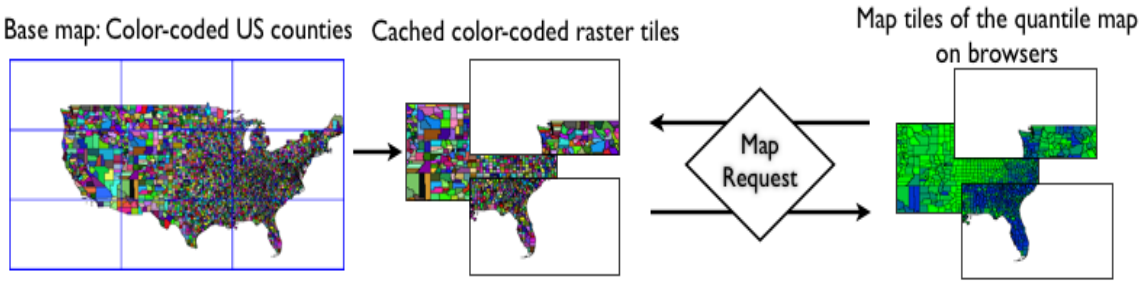
[5]http://www.polymaps.org



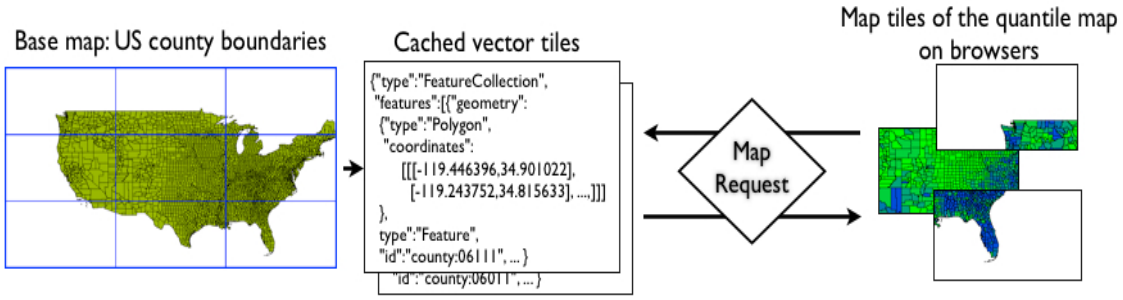
a. The approach of Gibin et al. (2008)



b. The approach of Mateos and O'Brien (2011)



c. The approach of Schmidt and Dev (2008)



d. The approach of the Polymaps library

Figure 2.2. Different approaches to tile-based choropleth mapping

increase the user interactivity and visual aesthetics of output maps since it renders areal units as vector shapes. However, its effectiveness is contingent upon multiple factors, such as the efficiency and quality of the map generalization, the type of and browser support for the web graphics technology, and the computing power of the client-side computers.

The type of web graphics technology, among the factors listed above, has attracted much attention in cartographic research as web technologies for vector rendering have undergone many changes. The most recent innovation in this regard is the advent of HTML5 Canvas and its competition with SVG, an established method for web-based vector rendering. SVG is "an XML-based graphics format integrating vector graphics elements, text, raster graphics, audio and video" (Neumann, 2012, p.575). It treats individual vector objects as independently accessible document elements, each having its own geometric and stylistic attributes as well as interactive behaviors (Huang et al., 2011). In contrast, Canvas is a drawing API of HTML5, a candidate for the next version of the HTML standard (W3C, 2011). The Canvas API allows for pixel-level drawing of vector and raster graphics through executions of Javascript code (Lienert et al., 2012). Unlike SVG, the Canvas API provides no means to access individual vector objects once it completes rendering the entire image. These differences between SVG and Canvas would have varying effects on the display of and user interaction with vector-tile-based choropleth maps. Empirical investigations of such effects, however, have often been conducted in ad-hoc or incomparable manners, thereby resulting in inconsistent results; for example, Johnson and Jankun-Kelly (2008) reported Canvas performed less efficiently than SVG in generating and supporting user interaction with parallel coordinate plots and treemaps, while Boulos et al. (2010) indicated Canvas would be more efficient than SVG in graphics rendering.

Overall, the literature reviewed thus far indicates that tile-based approaches to choropleth mapping would scale better when dynamic, interactive mapping needs to be supported. Despite the nascent nature of the literature, it provides encouraging

demonstrations, which raises another interesting question: *How scalable are the tile-based approaches in supporting the real-time rendering of and high-level user interaction with online choropleth maps?* This question remains under-investigated but deserves empirical investigation for three reasons. First, large spatial data of small areal units have been increasingly produced and distributed online, sometimes in real time. This increase in data supply has also led to growing demands for dynamic and scalable choropleth mapping. Yet, there is little knowledge of how existing and emerging technological solutions can address the rising demands. Second and in relation to the first reason, researchers and practitioners need solid benchmarks for making the technological choices that best suit the requirements of their choropleth mapping applications with respect to scalability and user interaction. Third, the use of tile-mapping methods whose scalability is empirically proven would allow users to easily explore global spatial distributions of data values measured at small areal units and remove the need for region-by-region exploration due to the current data limits of choropleth mapping applications. Motivated by these reasons, this dissertation study evaluates the scalability of three tile-based approaches: the raster approach of Schmidt and Dev (2008), and the SVG and Canvas approaches of the Polymaps library.

2.4 Summary

The literature review in this chapter provides a snapshot of the evolution of choropleth mapping, online mapping, and their interaction, i.e., online choropleth mapping. The literature on choropleth mapping clearly shows that choropleth map design is an exploratory process requiring decision making concerning multiple design criteria, such as data classification method, color scheme, and legend organization. Developments in online mapping technologies have not only increased access to choropleth maps, but also have increasingly allowed end users to participate in the exploratory process of choropleth map design with large areal data. Tile-based mapping has been widely used for efficient online delivery of large map data and has recently been employed to scale up the data

capacity of dynamic choropleth mapping. Although the emerging approaches to tile-based choropleth mapping each claim to better support dynamic yet scalable choropleth mapping, actual empirical benchmarks of their scalability do not exist in the current literature, to the best of this author's knowledge. The lack of such information impedes the development and provision of online mapping applications that can allow end users to interactively explore the spatial aspects of the fast-growing geospatial data by means of choropleth mapping. As such, this dissertation study empirically assesses how different approaches to tile-based choropleth mapping scale with the number of spatial objects in an areal data set, in typical scenarios concerning the interactive use of choropleth maps.

CHAPTER 3

METHOD

The literature review in the previous chapter pointed out the lack of research on the scalability aspects of the emerging tile-based approaches to online choropleth mapping. To fill this gap in the literature, this study aims at assessing how scalable three tile-based methods are in enabling dynamic choropleth mapping and supporting various types of map interaction. In particular, the three methods under evaluation include the raster tiling approach proposed by Schmidt and Dev (2008) and the SVG- and Canvas-based approaches that use vector tiles, as in the Polymaps library. For the evaluation, the study develops two test applications, one for the raster and one for the vector tiling approaches. The vector application supports both the SVG- and Canvas-based tile-mapping approaches without substantial changes in its architecture. The evaluation measured and compared the response times of these applications in supporting multiple types of user interaction. This chapter describes the functionality and implementation of the applications, the characteristics of the test data sets, the procedure of map tile generation, and the framework for the evaluation.

3.1 Test Applications

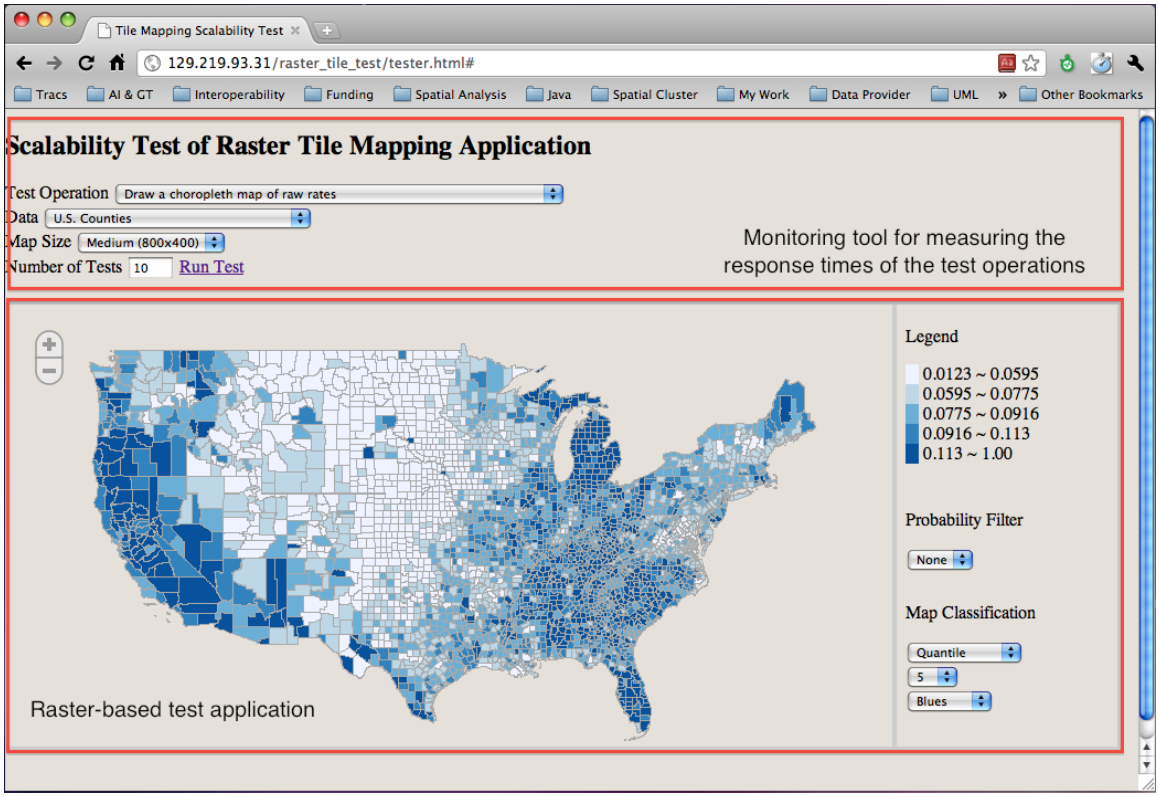
Both test applications (Figure 3.1) support a common task: the spatial exploration of a rate variable. This functionality was chosen because choropleth maps are frequently used to examine the spatial distribution of rates that are obtained by standardizing the counts of disease events by the numbers of population at risk (Slocum et al., 2009). When raw rates are used as the estimates for the risks of rare events, such as disease rates, they become unreliable if the population at risk of the events is small (Waller and Gotway, 2004). To address this problem, raw rates are often smoothed by accounting for the number of events and population at risk at nearby locations or in the overall study area (Wang, 2006).

Estimates of risks based on rates in areas with small populations at risk can thus be improved by "borrowing strength" from nearby populations at risk (Anselin et al., 2006).

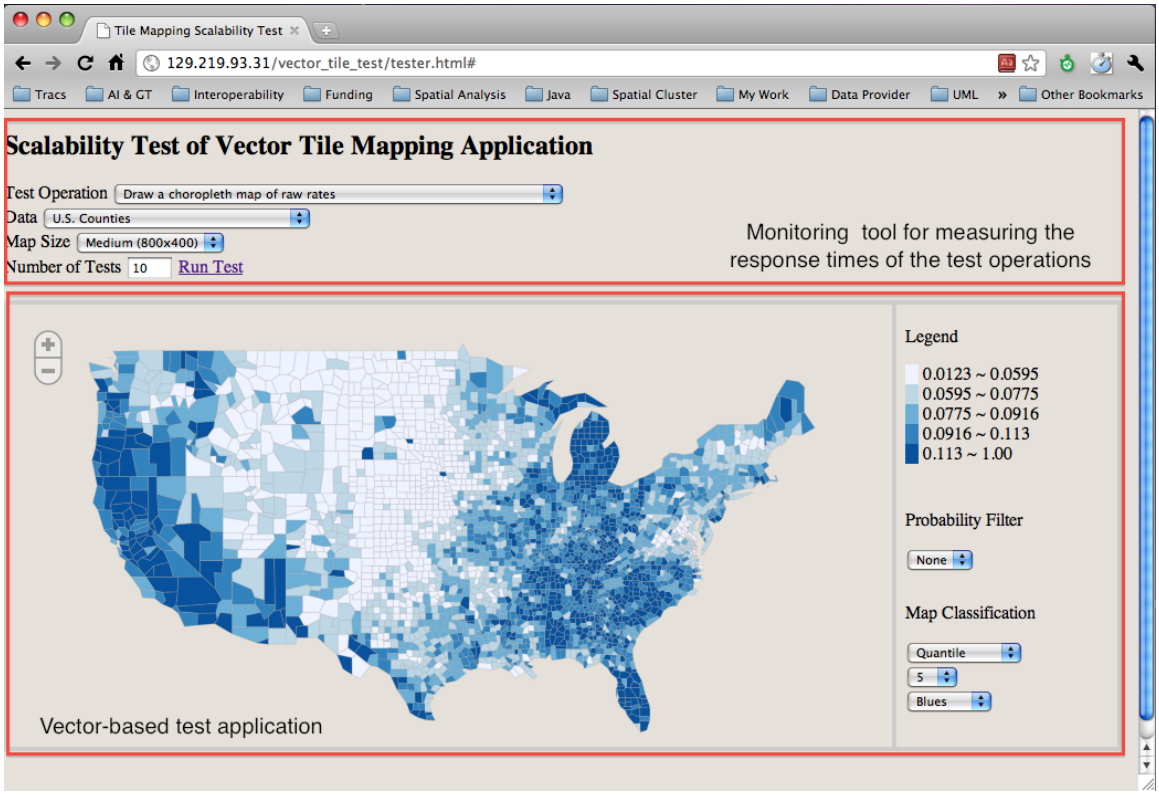
The test applications allowed for data exploration by providing a set of interactive functions that many choropleth mapping tools commonly support (Andrienko et al., 2002; Zhao and Shneiderman, 2005). These functions are as follows:

- Dynamic choropleth map generation: Choropleth maps were created on the fly as the user specified design parameters, such as the variable to be mapped, method for data classification, number of classes, and type of color scheme.
- Map juxtaposition: Two choropleth maps of raw and smoothed rates were created simultaneously and displayed side by side. This function helps the user assess whether the smoothed rates reveal spatial patterns differently than the raw ones.
- Dynamic data query: Choropleth maps could be dynamically updated to answer questions such as, "Where are the rates significantly different from their expected values at a statistical confidence level of 95%?" By using the information of areal units where a rate is likely to differ from its expected value and by permitting flexible selection on a desired confidence level, this query tool allows the user to carry out statistical inferences, in addition to visual inferences from a choropleth map (Waller and Gotway, 2004).
- Map navigation: As with general reference maps, the user was able to pan and zoom choropleth maps. These functions allowed the user to examine spatial patterns of rates across different spatial scales and extents.

Both applications employed a client-server architecture. In the raster application (Figure 3.2a), the server-side components served four functions: 1) providing access to rates and their probabilities, 2) providing access to tiles, 3) rendering tiles, and 4) caching the results of data classification. These functions needed to be carried out on the server



a. The raster application and its monitoring tool



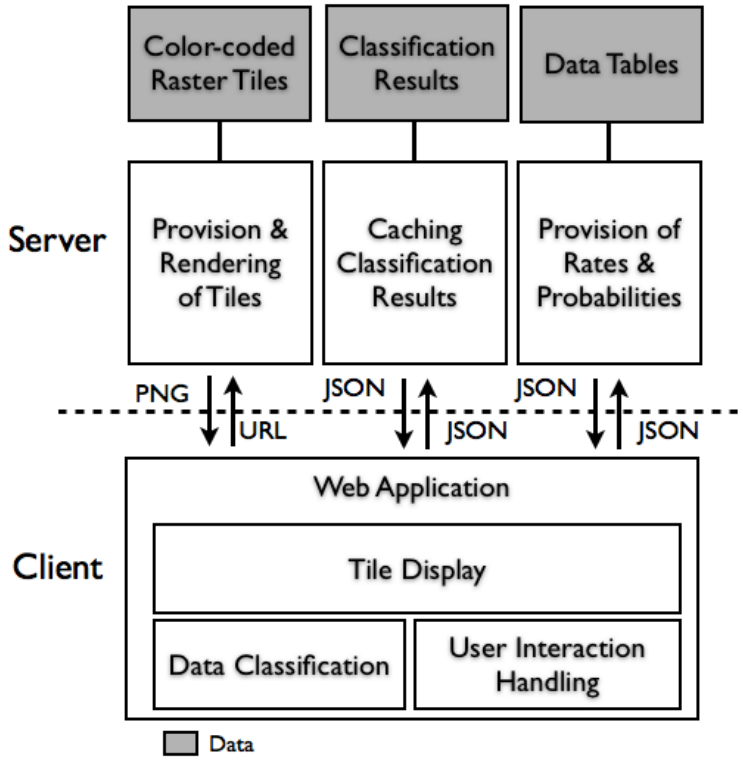
b. The vector application and its monitoring tool

Figure 3.1. Snapshots of test applications

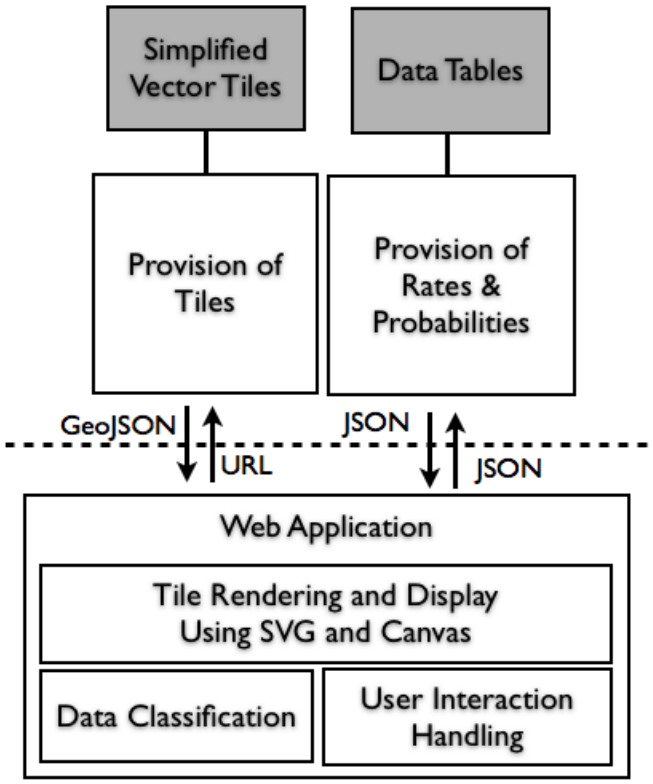
side since they required access to data, including map tiles and data tables. On the other hand, the client-side components performed three tasks: 1) classifying the rates, 2) handling user interaction with choropleth maps, and 3) displaying map tiles. Classification of rates could have occurred on the server side; however, it was done on the client side in order to make the implementation of the two test applications similar.

When compared to the raster application, the architecture of the vector application (Figure 3.2b) had one distinctive feature: tiles were downloaded and rendered on the client side. Because of this feature, the server side of the vector application only provided access to map tiles. It neither carried out tile rendering nor cached the results of data classification. On the other hand, the client-side components of the vector application were similar to their raster counterparts, with the exception of the additional component for tile rendering. Geographic coordinates of polygons in each vector tile were transformed into screen coordinates. Polygons were then drawn as lines on an inline SVG or Canvas element. In the Canvas-based version of the vector application, Canvas elements with the 2D context were used with the hardware acceleration option disabled in the browsers.

The test applications were implemented with a combination of software tools. In both applications, the server-side components were created on top of the Python Django framework and the Python spatial analysis library (PySAL). PySAL is a well-known open source spatial analysis toolkit developed by the GeoDa Center at Arizona State University (Rey and Anselin, 2007). In this study, it provided statistical computing capabilities for rate estimation and spatial smoothing, while the Django framework enabled web deployment of such analytical components and remote access to tile and source data. In addition to these packages, the raster application employed the dynamic tile mapper DynTM to cache the results of data classification and re-render raster tiles. On the client side, both applications used the Polymaps library. This library not only supports the display of raster tiles, but also the dynamic rendering of vector tiles. Additional client-side



a. The raster application



b. The vector application

Figure 3.2. Architectures of test applications

tools for data classification and user interaction handling were written in Javascript. The source code used to develop the test applications is listed in Appendix 1.

3.2 Test Data

In this study, scalability is defined as the capability of tile-mapping approaches to effectively render and interact with large areal data, in terms of the number of areal units (Eick and Karr, 2002). For a valid assessment of this scalability aspect, the study created choropleth maps of a fixed spatial extent while increasing the number of areal units within that extent in a nearly linear fashion. The study operationalized this evaluation setup by utilizing five different administrative boundaries of the conterminous U.S. in 2000. The source data for those boundaries comprised Environmental Systems Research Institute (ESRI) shape files with polygon geometry.

In particular, the test data of the study included the boundaries of counties, primary care service areas (PCSAs),[1] school districts, zip code areas, and census tracts. These boundaries were chosen first, because they are frequently used for choropleth mapping in practice; second, U.S. counties can be a good starting point for the evaluation, since they are the largest data set for which the performance of an online choropleth-mapping application has been reported (Steiner et al., 2002; Zhao and Shneiderman, 2005); and third, because the numbers of areal units included in those boundaries increase approximately by a factor of two in the order they are listed: 3,109; 6,469; 11,680; 29,885; and 64,919 polygons. This linear relationship in the numbers of areal units makes it easy to interpret the evaluation results.

[1] PCSAs are a relatively new type of geographic unit defined by Health Resource and Services Administration to better reflect flows of Medicare patients to primary care physicians (Mobley et al., 2006).

3.3 Tile Generation

Tile generation in this study was completed in two steps. First, some of the test data sets were pre-processed to facilitate the tile generation. Second, raster and vector tiles were created from each test data set. This subsection explains this two-step process in detail.

3.3.1 Data Pre-processing

Some of the test data sets, i.e., the boundaries of PCSAs and school districts, underwent a pre-processing step due to difficulties with geometry simplification. As will be discussed next, polygons in the test data sets were simplified for vector tiling, but this simplification produced problems with the original boundary data of PCSAs and school districts mainly because of topological errors in the source data. To address these problems, the study artificially recreated those boundaries by spatially aggregating census tracts to PCSAs and school districts in which the centroids of the tracts were completely contained. Although this data processing was an arbitrary measure, it had little impact on the evaluation since it did not change the numbers of the PCSAs or school districts.

3.3.2 Common Configurations for Map Tiling

Except for the map data model, the tiles used for this study were generated in similar manners so as to enable a valid comparison between the raster and vector tiling approaches. Both types of map tile were pre-generated in order not to introduce confounding factors related to dynamic tile generation. The configurations for tile generation were identical for both the raster and vector tiles. Before tiling, the test data sets were projected on the spherical Mercator coordinate system since it is widely used for tile-based online maps (Sample and Ioup, 2010). The typical quadtree-based tile grid was employed for three zoom levels ranging from four to seven. The study limited tile generation to this range of zoom level because the entire conterminous U.S. becomes visible at zoom level four, and choropleth maps were zoomed in three times in a row

Table 3.1. The number and data size of raster tiles

Data set	Number of Areal units	Number of Tiles	Data size of Tiles (Mb)	Minimum Tile size (B)	Maximum Tile size (KB)
Counties	3,109	271	4.6	283	16.1
PCSAs	6,469	271	9.1	290	22.5
School districts	11,680	272	9.1	267	23.3
Zip code areas	29,885	271	15.0	302	27.5
Census tracts	64,919	272	12.2	267	29.4

during the evaluation as discussed in Section 3.4.1. Individual map tiles were indexed by zoom level and the row and column numbers in the corresponding tile grid. They were then stored in disk as separate files.

3.3.3 Raster Tiling

Raster tiling in this study followed the approach of Schmidt and Dev (2008). As such, the tiles were generated in such a way that each areal unit was color-coded with its unique ID. In particular, an unclassed choropleth map was first created for each test data set using the IDs of the areal units. This map was then split into raster tiles by using the tile grid explained above. The size of each tile was 256×256 pixels, and Portal Network Graphics (PNG) formats, 24-bit true-color or 8-bit indexed, were used to store the tiles as image files. The 8-bit PNG stores a palette of the colors included in an image file and utilizes the color index in the palette as the final data values for raster pixels (W3C, 2003). Since the size of the index value is only 8 bit, the indexed PNG can reduce the size of raster tiles to some extent. The study used the 8-bit PNG for the tiles with a small number of colors and the 24-bit PNG for the other cases. A tool included in DynTM was employed to generate, store, and index the raster tiles. Table 3.1 shows the number and data size of the non-blank raster tiles used for the evaluation.

3.3.4 Vector Tiling

The study used multiple steps of data processing for the vector tiling. In general, polygons in the test data sets were first simplified for the various zoom levels and then partitioned

into rectangular tiles. For the simplification, the Douglas-Peucker algorithm (Douglas and Peucker, 1973) was applied to the polygon data multiple times. The degree of simplification or tolerance level was decreased as the zoom level increased.

Two challenges arose during repeated polygon simplification: preservation of topological relationships among the polygons and determination of tolerance levels. Each polygon in a shape file carries its own outline, even though it shares some portion of its boundary with other polygons. This double representation of polygon outlines not only results in topologically inconsistent polygons after simplification but also produces numerous slivers and gaps between simplified polygons. To prevent this, the study first converted polygon data into topologically correct polyline data in which only one line representation existed for the shared boundary of two neighboring polygons. Geometric simplification was then applied to the resulting polyline features, and the new polygon geometries were constructed from the simplified lines. Since this process involved multiple such conversions, the attribute data of the original polygons were lost. The centroids of the original polygons were extracted and then spatially joined to the new polygons in order to link the original attribute data to the new polygon geometries. While the study employed a spatial joining operation for convenience, the identification numbers of the original polygons could have been passed along in the conversions to preserve data accuracy during simplification. The study utilized ET GeoWizards[2] for this topologically consistent polygon simplification, for which Figure 3.3 provides an overall summary.

Another challenge in vector tiling is the determination of multiple tolerance levels for geometry simplification so as to vary the geometric detailedness of polygons across zoom levels. Although multiple software tools exist for geometry simplification, few provide the intelligence to inform a possible range of optimal tolerance values. Therefore, the study used a bespoke tool to compute the minimum and maximum values of tolerance

[2]http://www.ian-ko.com/ET_GeoWizards/gw_main.htm

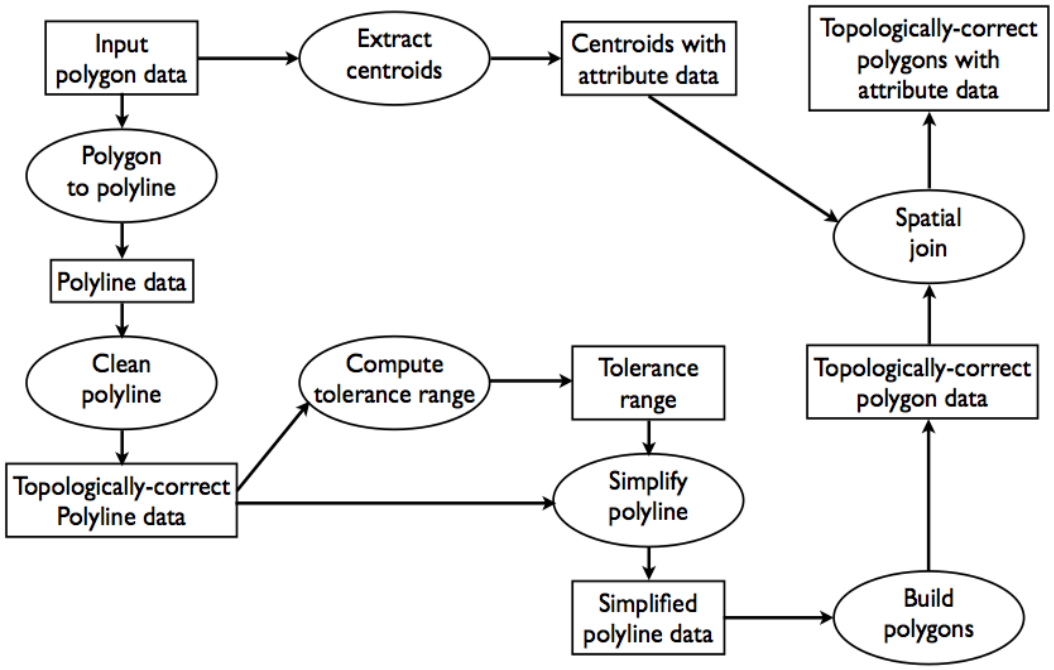


Figure 3.3. The process of vector tile generation

from the topologically correct polyline data (Appendix 2). It achieved this by computing for all polylines the maximum possible distance between the polyline and its simplified version, i.e., the threshold for simplification that is used in the Douglas-Peucker algorithm (Douglas and Peucker, 1973). Through multiple experiments with a test data set, it was found that a tolerance value greater than 20 percent of its allowed maximum tended to induce severe geometric distortion in new boundaries. Thus, the study used 20 percent of the maximum tolerance value for the lowest zoom level and then gradually decreased the tolerance value for the higher zoom levels, by 0.04, the optimum increment chosen from multiple trials with the test data set.

Handling polygons whose boundary crosses multiple cells in a tile grid is also a critical concern in vector tiling. As discussed in Section 2.2.4, such polygons can be partitioned into multiple parts with each being stored in the corresponding map tile. Alternatively, the mapping between a polygon and a tile can be pre-calibrated and used as a spatial index so that one polygon belongs to only one tile. The total size of tile data

Table 3.2. The number and data size of vector tiles

Data set	Number of Areal units	Number of Tiles	Data size of Tiles (MB)	Minimum Tile size (KB)	Maximum Tile size (MB)
Counties	3,109	271	5.6	308	0.4
PCSAs	6,469	271	11.2	311	0.8
School districts	11,680	272	18.7	307	1.0
Zip code areas	29,885	271	40.8	340	3.2
Census tracts	64,919	272	84.0	367	7.6

would be greater in the former approach than the latter since the polygon partitioning process increases the total number of geometry primitives included in the tiled map. Despite this drawback, the polygon partitioning method was employed in this study for two reasons. First, it prevented invisible parts of polygons in a map tile from being downloaded and rendered, incurring little unnecessary costs for rendering the map section corresponding to the user's current viewport (Gaffuri, 2012). Second, map tiles from the data partitioning process facilitated tile-level drawing of vector primitives on an inline SVG or Canvas element by reducing dependencies between tiles. This simplified the client-side rendering and management of vector tiles.

The study stored each vector tile in a text file using GeoJSON encoding. GeoJSON is a format for representing a variety of geographic data structures in Javascript object notation (geojson.org, 2012), which is a lightweight data exchange format that humans can read and write easily (json.org, 2012). This format was chosen since it uses relatively compact representations of geographic features, especially when compared to GML (Lienert et al., 2012). Vector tiles were not compressed at the file level but transmitted with gzip compression enabled at the web server level for efficient data delivery. Once a tile was transmitted, its geometry primitives were drawn as lines in an inline SVG or Canvas element. Table 3.2 shows the number and data size of non-blank vector tiles used for the evaluation. As expected, the total numbers of vector and raster tiles were identical, but the total data size of the vector tiles was about 29 times larger than that of the raster tiles, on average.

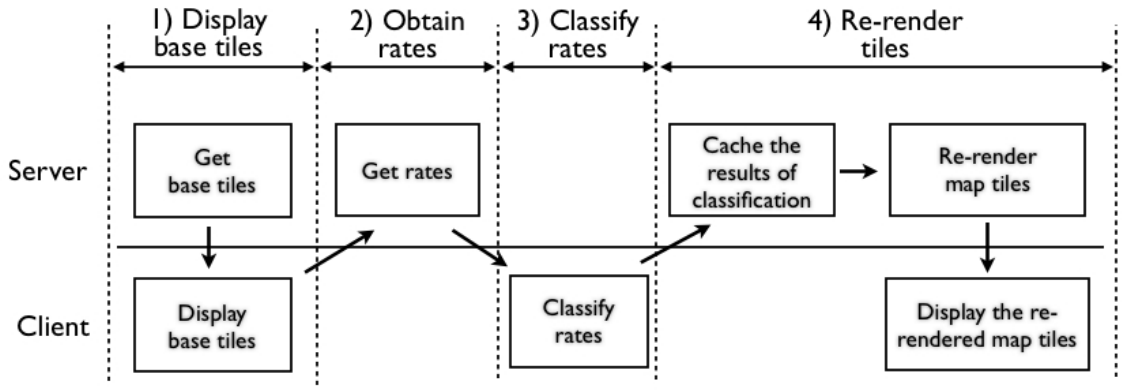
3.3.5 Comparability of Raster and Vector Tiles

The discussion thus far shows that various factors are involved in tile generation. It is worth clarifying how they affect the evaluation in this study. First, pre-generation of raster and vector tiles does not make the evaluation invalid. Although geometries are already rendered in color-coded raster tiles while they are not in vector tiles, the former can be considered to be "partially cached vector tiles" where geometries are rasterized in advance but complete choropleth rendering is deferred (Deering et al., 1988). The study examines whether this partial caching improves the scalability of dynamic choropleth mapping and, if so, to what extent it does so in comparison to the pure vector-tiling approach. Second, it needs to be affirmed that the study controlled for other factors related to the evaluation, such as number of areal units, range of zoom levels, and tiling schemes. A common set of values was used for these factors in both the raster and vector tiling for a valid comparison of the test applications.

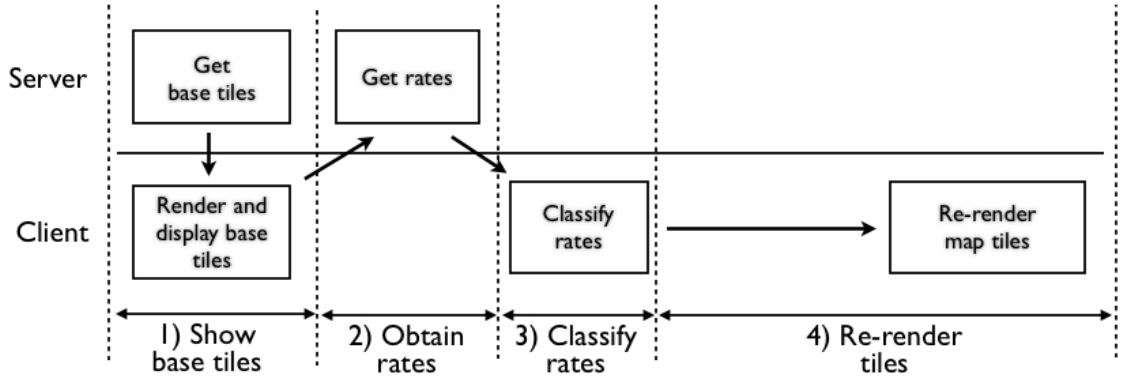
3.4 Evaluation Framework

3.4.1 Test Operations

In this study, five test operations were designed to evaluate the performance of the four functions provided by the test applications. Specifically, the test operations served to: 1) draw an overview choropleth map of the raw rates (dynamic choropleth mapping); 2) draw two choropleth maps of the raw and smoothed rates in separate views simultaneously (map juxtaposition); 3) filter the rates by probability and update the choropleth map accordingly (dynamic data query); 4) zoom in three times sequentially (zoom in); and 5) pan three times sequentially (pan). The study employed these test operations since they are frequently used when the user creates and examines choropleth maps of rate variables (Waller and Gotway, 2004; Zhao and Shneiderman, 2005). The first three of the test operations directly paralleled the functions of the test applications. The zoom-in and panning operations were designed to assess the map navigation functionality of the test



a. Raster application



b. Vector application

Figure 3.4. Workflows for dynamic choropleth mapping in test applications

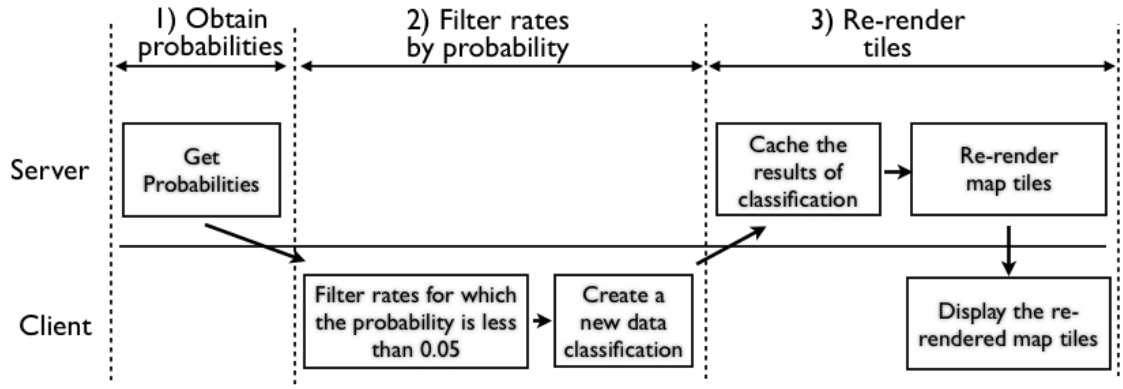
applications. They were repeated three times in a row in order to reflect user interaction spanning multiple map scales and large spatial extents.

Both applications supported all test operations, but the task distributions between the client and server were different for the two applications. The first test operation, dynamic choropleth mapping, was completed after four steps of work in both applications (Figure 3.4): 1) displaying base tiles before choropleth rendering, 2) obtaining rates, 3) classifying rates, and 4) re-rendering map tiles. While the tasks for steps 2 and 3 were identical in both applications, those for steps 1 and 4 differed due to distinctive characteristics of the base tiles: color-coded pixel-based images for the raster application and simplified polygons for the vector application. This difference in tile data caused the

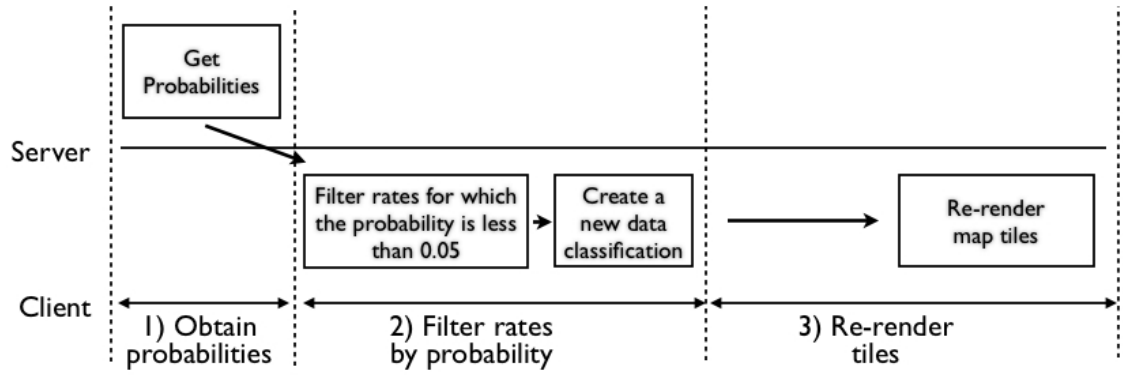
two applications to handle map tiles differently. In step 1, the raster application downloaded and displayed base tiles on the browser while its vector counterpart downloaded tiles and drew polygons for a client-side map display. The rendering of base tiles was introduced to assess the time cost that the rasterization of polygons incurs in the vector application in comparison to the simple display of tile images in the raster application. Step 4 in the test operation was devised to obtain the time costs of tile re-rendering after the initial rasterization of geometries. In this step, the raster application processed tiles on the server side after caching the classification results in the server-side memory for reuse. On the other hand, both the SVG and Canvas versions of the vector application re-rendered map tiles on the client side without caching the data classification results on the server side and downloading updated map tiles. In the SVG-based application, map tiles were re-rendered by re-specifying the color style attribute of polygon elements; thus the rasterization of polygon geometries was not repeated. In contrast, all features in a map tile had to be completely redrawn in the Canvas-based application, since Canvas provides no means to distinguish different geometry primitives once they are drawn (Neumann, 2012).

The purpose of the second test operation, map juxtaposition, was to create two choropleth maps at once: one of raw rates and another of smoothed rates. The test applications undertook this operation by initiating two sessions of dynamic choropleth mapping at the same time. Consequently, the task distribution across the client and server was similar for map juxtaposition and dynamic choropleth mapping. However, the workload of the test applications doubled during map juxtaposition.

The third test operation, dynamic data query, involved a workflow of three steps (Figure 3.5): 1) obtaining probabilities, 2) filtering rates by probability, and 3) re-rendering tiles. As in dynamic choropleth mapping, the tasks for the first two steps were identical in both applications, but those for tile re-rendering were carried out in different places: the server side in the raster application and the client side in the vector



a. Raster application



b. Vector application

Figure 3.5. Workflows for dynamic data query in test applications

application. For the second step of the test operation, this study used a fixed probability threshold for rate filtering, i.e., 0.05. However, in real-world applications the user would change the threshold interactively, and each change would result in a new classification scheme that switched previously filtered polygons to unfiltered ones, or vice versa. To correctly reflect these alternations in tile re-rendering, the vector-based applications had to check the class changes in all geographic features contained in the map tiles and make adjustments to their color values. The SVG-based vector application achieved this by re-defining the color style attribute of the polygon elements. In contrast, the Canvas-based vector application had to redraw the map tiles, since direct access to geometry primitives is impossible in Canvas.

The fourth test operation, zooming in, required the test applications to increase the map scale by one level along the pre-specified set of discrete map scales. Although continuous map scales can be used for tile mapping, step-wise zooming along discrete map scales is widely utilized since it simplifies the generation and retrieval of map tiles (Sample and Ioup, 2010). Adopting the zooming based on discrete scales, the test applications replaced the old tiles of a lower zoom level with the new tiles of the next zoom level whenever the user zoomed in on the map. This study tested the zoom-in operation for zoom levels ranging from four to seven, i.e., map users would first examine an overview choropleth map of the U.S. at zoom level four and then start to zoom in on the map. The operation zoomed in three times automatically for testing purposes. In each zoom-in operation, the test applications obtained and rendered new tiles. Once all events of tile loading and rendering were completed, the applications automatically initiated a new zoom-in operation.

The last test operation, panning, involved three sequential changes in the map extent as well. Each pan operation moved the map center by 256×256 pixels on the screen in the northwestern direction at zoom level seven. Since this operation required the acquisition and rendering of new map tiles for the panned area, with most of the new tiles becoming visible only partially visible, the time cost due to the wasted real estates of the map tiles was examined for each of the three tile-based approaches. As in the other test operations, the workflow for completing the panning operation differed in the test applications: tiles were rendered on the server and then displayed on the client side in the raster application, while the rendering and display of tiles both occurred on the client side in the vector application. Three sequential panning operations were executed automatically; a new pan started once all tiles of the previous map were rendered.

So far, the discussion indicates that the workflows and implementations of the test operations were different for the two test applications. Despite the differences, the performances of the operations in these test applications can be compared for two reasons.

First, as asserted by Luotsinen et al. (2007) two methods are comparable if both of them can be used to implement the same functionality or requirement. Second, as long as the methods implemented in the test applications are what researchers and practitioners need to choose from, a systematic investigation of their pros and cons is worthwhile as it informs their potential users. Despite these justifications, the results of the comparison in this study should be interpreted with careful consideration to the differences that exist in the two applications.

3.4.2 Metric

This study assessed the scalability of the three tile-mapping approaches by measuring the response times of the test applications when each test operation was executed. Response time refers to "the amount of time that elapses from when a user submits a request until the result is returned from the system" (Lilja, 2000, p.19). The study used response time because an online choropleth-mapping system not only needs to handle large areal data but also needs to render a complete choropleth map from them in a short time. Response times can serve as estimates for the user's wait time (Laird and Brennan, 2006).

The study adjusted the general definition of response time above so that it could better reflect the user's wait time. From the user's perspective, each test operation starts when the user initiates it, and the operation ends when the resulting choropleth map is completely displayed on the browser. To reflect this life span of the test operations, the study defines response time as the amount of time that elapses from when a user initiates a test operation until the resulting choropleth map is completely visualized on the user's screen. The main difference of this definition from the generic one is that it takes into account the time the browser spends on rendering and/or displaying the map tiles after the tiles are returned from the server.

To measure response times, the study developed one Javascript-based client-side monitoring tool for each test application (Figure 3.1 and Appendix 1). This monitoring

tool first changed the state of the test application so that a test operation could be invoked. For example, before the test operation for dynamic data query was assessed, the monitoring tool prepared a choropleth map to which the query was applied. Next, the monitoring tool automatically initiated the test operation and recorded the initiation time. Then, the monitoring tool waited for the test application to complete the test operation, obtain map tiles, and display them as a complete choropleth map. The time interval between the test operation being initiated and the map display being completed was measured in milliseconds and recorded as the response time. The monitoring tool could automatically repeat this measurement of response time and avoided the potential side effect of browser-side caching on the measurement by removing all cached data related to map tiles, rate data values, and classification results whenever a new measurement begun.

To increase the reliability of the measured response times, the study repeated the measurement fifty times for each pair of test operation and data set, and used the average for the evaluation. The study chose fifty as the sample size for two reasons. First, it exceeded the generally known upper limit on the size of a small sample drawn from normally distributed data, which is thirty (Lilja, 2000). Second, the time cost for obtaining the fifty measurements for each pair set was affordable, ranging from fifty seconds to more than two hours.

Generally, it is known that measuring the performance of a web mapping application is complicated by multiple factors such as internet connection, network traffic intensity, browser type, map size, operating system, and specifications of the client- and server-side hardware (Kraak, 2004). This study could not control for all of the various conditions of these factors. In terms of Internet connection, only a local area network with 1 Gb/s of bandwidth was used for the evaluation. Similarly, the response times were measured only for the case that one client (i.e., one test application) accessed the server. As will be shown in Section 4.2, a unit increase in network traffic intensity or workload would increase the response times of the test applications. However, it would have little

impact on how many areal units the applications could handle. Therefore, the study limited the evaluation to the case of one client access.

In the study, the factor of browser type was controlled for to avoid confounding effects that might be caused by potential associations between browser type and different methods for web graphics rendering. In particular, the study focused on Google Chrome Version 17.0 since it showed the highest level of support for HTML5 and SVG at the time of evaluation (March 20, 2012).[3] According to www.w3schools.com, the proportion of Google Chrome users increased from 37.3 to 52.7% from March 2012 to April 2013 (the time of this writing), which indicates the evaluation results of the study have now become relevant to a majority of online users.[4] Although the study focused on Chrome, the tile mapping approaches evaluated here can be used in a broad range of web browsers that support HTML5 and inline SVG. At present, these browsers include Internet Explorer 9.0 or later, FireFox 4.0 or later, Chrome 7.0 or later, Safari 5.1 or later, and Opera 11.6 or later,[5] which are in total used by at least 85.6% of the visitors to www.w3schools.com.

The operating system used for the study was Mac OS X v.10.6 Snow Leopard on both the client and server sides. Although Johnson and Jankun-Kelly (2008) reported that the SVG- and Canvas-based rendering of vector graphics showed similar scalability in terms of the size of vector data across different client-side operating systems, the results of the study should be interpreted in the context of using Chrome on the Mac OS X system. The map size used in the evaluation was fixed at 800×400 pixels as this size could show the entire conterminous U.S. When the map size is larger than that, only blank tiles will be added as long as map tiles are available for the conterminous U.S. On the other hand, smaller map sizes are ineffective for choropleth mapping of the U.S. since the view limits the spatial extent to be explored. As for browser type, the factor of hardware specification

[3] http://caniuse.com/#agents=desktop&cats=HTML5,SVG

[4] http://www.w3schools.com/browsers/browsers_stats.asp

[5] http://caniuse.com/#cats=HTML5,SVG

was controlled for to prevent erroneous conclusions that might be influenced by potential correlations between specific hardware specifications and different methods for web graphics rendering.

3.4.3 Test Environment

The test environment included two computers that served as server and client on a network with 1 Gb/s of bandwidth. Both computers were Mac OS X v.10.6 Snow Leopard systems with 2.8 GHz Intel Core 2 Duo processors and 4 GB memory.

CHAPTER 4

RESULTS

The previous chapter explained the test materials and evaluation framework that this study used to assess the three tile-based approaches to choropleth mapping. This chapter presents the results of the assessment with its focus on comparing the efficiency and scalability of the three tile-based approaches during each test operation. The rules used for the assessment are that the less time it takes to complete a test operation with a test data set, the more efficient the tile-mapping approach is, and that the more areal units a test operation can handle in a unit time, the more scalable the tile-mapping approach is.

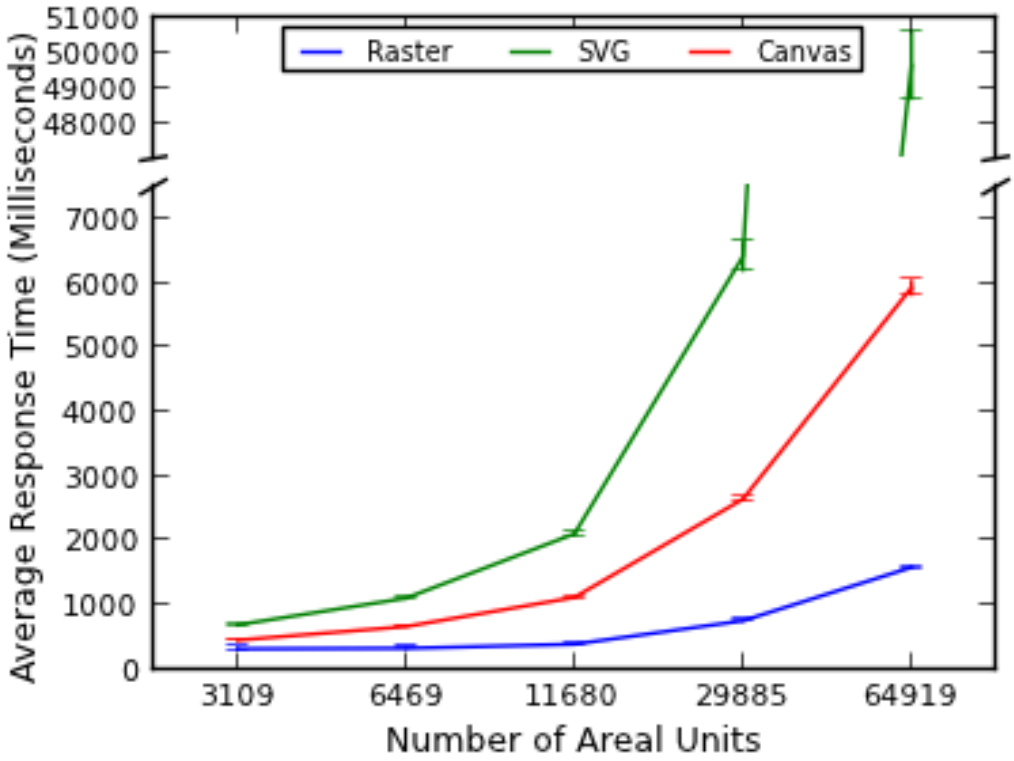
4.1 Dynamic Choropleth Mapping

Table 4.1 and Figure 4.1 show the average response times of the test applications during dynamic choropleth mapping with the three tile-mapping approaches. As evident in these results, the raster approach was the most scalable, since it took the shortest times to complete dynamic choropleth mapping with all test data sets. The Canvas approach was ranked next in terms of scalability: as the number of areal units increased from 3,109 to 64,919, the time for dynamic choropleth mapping increased 4.07 and 12.15 times in the raster and Canvas approaches, respectively. The SVG approach was the least scalable:

Table 4.1. Average response times during dynamic choropleth mapping

	Raster		SVG		Canvas	
Number of areal units	Average (milliseconds)	Margin of error ($\alpha = 0.05$*)	Average	Margin of error	Average	Margin of error
3109	311.80	28.39	684.14	16.49	447.42	10.38
6469	320.24	19.67	1100.66	25.06	658.86	10.79
11680	384.50	15.30	2103.30	46.10	1112.50	13.36
29885	748.76	18.88	6415.10	235.19	2642.42	27.52
64919	1580.12	16.69	49625.62	976.09	5929.74	127.29

* α represents significance level.



*The bars in the figure represent margins of error at 95% confidence level.

Figure 4.1. Response time chart of test applications (dynamic choropleth mapping)

when the number of areal units increased as before, the time for dynamic choropleth mapping increased 72.54 times from 686.14 milliseconds to 49.62 seconds.

Section 3.4.1 explained that dynamic choropleth mapping was completed in four steps: 1) displaying base tiles, 2) obtaining rates, 3) classifying rates, and 4) re-rendering tiles. Table 4.2 shows the composition of the time spent on each step by the three tile-mapping approaches. In the raster approach, on average 48% of the response time was spent displaying and re-rendering tiles, while in the other approaches more than 80% of the response time was spent handling tiles. A close look at Table 4.2 shows that the percentage of the time spent on tile handling decreased with the number of areal units in the raster approach, while the opposite pattern was observed in the SVG approach.

Table 4.2. Details of time spent on dynamic choropleth mapping

a) Raster approach					
Number of areal units	Display base tiles milliseconds (%)	Obtain rates	Classify rates	Re-render tiles	Total
3109	86.27 (27.67)	61.44 (19.70)	16.12 (5.17)	147.97 (47.46)	311.80 (100)
6469	87.91 (27.45)	109.92 (34.32)	21.12 (6.60)	101.29 (31.63)	320.24 (100)
11680	88.24 (22.95)	168.56 (43.84)	27.78 (7.22)	99.92 (25.99)	384.50 (100)
29885	97.31 (13.00)	433.96 (57.96)	68.70 (9.18)	148.79 (19.87)	748.76 (100)
64919	112.67 (07.13)	1064.26 (67.35)	132.88 (8.41)	270.31 (17.11)	1580.12 (100)
b) SVG approach					
Number of areal units	Display base tiles milliseconds (%)	Obtain rates	Classify rates	Re-render tiles	Total
3109	372.36 (54.27)	72.12 (10.51)	27.36 (3.99)	214.30 (31.23)	686.14 (100)
6469	677.32 (61.54)	113.22 (10.29)	30.48 (2.77)	279.64 (25.41)	1100.66 (100)
11680	1135.08 (64.43)	169.50 (08.06)	46.32 (2.20)	532.40 (25.31)	2103.30 (100)
29885	4665.50 (72.73)	431.80 (06.73)	91.94 (1.43)	1225.86 (19.11)	6415.10 (100)
64919	45246.06 (91.17)	1089.62 (02.20)	335.74 (0.68)	2054.20 (05.95)	49625.62 (100)
c) Canvas approach					
Number of areal units	Display base tiles milliseconds (%)	Obtain rates	Classify rates	Re-render tiles	Total
3109	244.49 (54.64)	60.64 (13.55)	22.30 (4.98)	119.99 (26.82)	447.42 (100)
6469	326.60 (49.57)	103.46 (15.70)	27.64 (4.20)	201.16 (30.53)	658.86 (100)
11680	543.73 (48.87)	163.94 (14.74)	45.12 (4.06)	359.71 (32.33)	1112.50 (100)
29885	1308.06 (49.50)	435.70 (16.49)	105.64 (4.00)	793.02 (30.01)	2642.42 (100)
64919	3061.31 (51.63)	1091.32 (18.40)	205.40 (3.46)	1571.71 (26.51)	5929.74 (100)

The time cost for displaying base tiles varied little in the raster approach but increased with the number of areal units in the vector-based approaches. For an identical set of vector base tiles to be downloaded and drawn on the browser, the time cost for the geometry rendering was, on average, 4.8 times larger in the SVG approach than in the Canvas approach. The growth of the time cost was also much greater in the SVG approach; the time cost increased 121.51 and 12.52 times in the SVG and Canvas approaches, respectively, while the number of areal units changed from 3,109 to 64,919. This result indicates that geometry rendering would hinder vector-tile-based choropleth mapping and that the effects would be more severe in the SVG approach than in the Canvas approach.

In the step for tile re-rendering, the average response time of the raster application was affected little by an increase in the number of areal units. In contrast, the vector application re-rendered map tiles slower as more areal units were shown on the map; the time for tile re-rendering grew about 9.59 and 13.10 times in the SVG and Canvas approaches, respectively, when the test data set altered from U.S. counties to census tracts. Although vector tiles were re-rendered, on average,1.50 times faster in the Canvas approach than in the SVG approach, this result shows that the former is more sensitive than the latter to the increase of input areal units.

Altogether, the results in this section suggest that tile-based dynamic rendering of choropleth maps is more efficient and scalable in the raster approach than in the vector approaches. They also indicate that geometry rendering would present substantial challenges for scalable choropleth mapping in both the SVG and Canvas approaches. In the SVG approach, its negative effects would be greater than those of tile re-rendering.

4.2 Map Juxtaposition

The test operation for map juxtaposition involved two function calls for generating choropleth maps of raw and smoothed rates (Section 3.4.1). Since these function calls

Table 4.3. Average response times during map juxtaposition

	Raster		SVG		Canvas	
Number of areal units	Average	Margin of error ($\alpha = 0.05$*)	Average	Margin of error	Average	Margin of error
3109	461.44	15.62	1332.94	35.01	776.58	12.34
6469	527.20	13.70	2464.68	153.30	1195.42	11.94
11680	596.76	17.30	4618.12	224.50	2033.90	16.26
29885	1158.56	20.87	12559.58	55.20	4833.78	25.16
64919	2242.62	24.98	97214.52	422.91	10577.70	262.31

were made simultaneously, the workload of the test applications doubled in all three of the tile-mapping approaches. Tables 4.1 and 4.3 show that when the response times for choropleth mapping and map juxtaposition were compared, the doubled workload slowed down the speed of map generation across all tile-mapping approaches. However, it was apparent with all test data sets that the raster approach was the fastest in completing map juxtaposition, the Canvas approach the second fastest, and the SVG approach the slowest.

In addition to the speed of map generation, the degree of its slowdown was also analyzed (Figure 4.2). In the raster approach, the time cost for map juxtaposition (T_M) was 1.42 (=2242.62/1580.12) ~1.65 (=527.20/320.34) times greater than that for dynamic choropleth mapping (T_D) (Tables 4.1 and 4.3). In the SVG approach, the ratio of T_M over T_D varied from 1.94 (=1332.94/684.14) to 2.24 (=2464.68/1100.66) while in the Canvas approach it ranged from 1.74 (=776.58/447.42) to 1.83 (=4833.78/2642.42). These results show that the impact of the doubled workload was the largest in the SVG approach, the second largest in the Canvas approach, and the smallest in the raster approach. This pattern in the slowdown indicates that the increase in the workload does not change the earlier observation that the SVG, Canvas, and raster approaches scale better in the order they are listed.

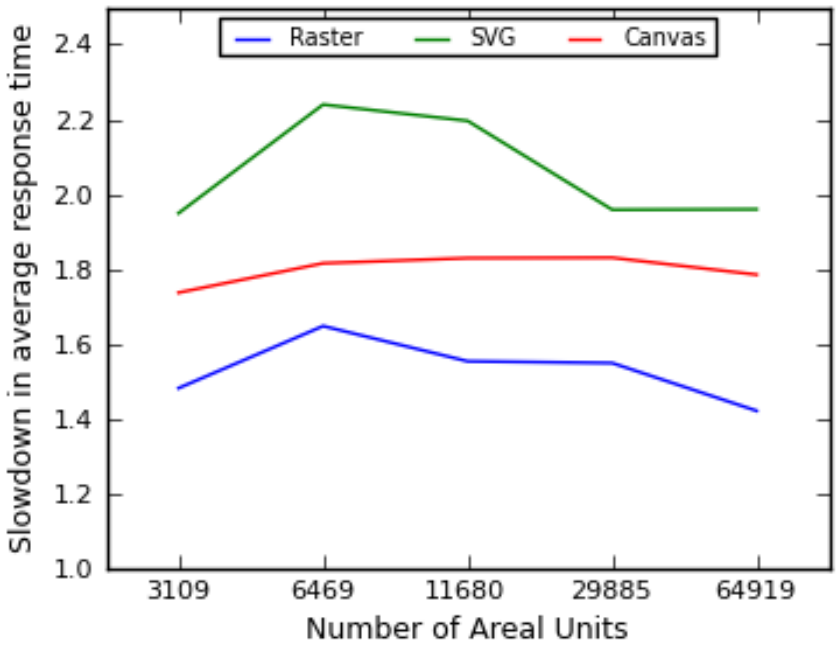


Figure 4.2. Response time chart of test applications (map juxtaposition)

4.3 Dynamic Data Query

Table 4.4 and Figure 4.3 show that the time cost for dynamic data queries was always largest in the Canvas approach while it was smaller and more similar in the other approaches. In addition, while the time cost increased with the number of areal units in all tile-mapping approaches, the amount of the increase was largest in the Canvas approach (2902.64 milliseconds), second largest in the SVG approach (1525.60 milliseconds), and smallest in the raster approach (1478.46 milliseconds). These results suggest that, unlike the cases of dynamic choropleth mapping and map juxtaposition, dynamic data querying was less efficient and scalable in the Canvas approach than in the other approaches.

The differences in the speed of dynamic data querying are attributable to the time spent on tile re-rendering. As discussed in Section 3.4.1, the test applications completed dynamic data querying by undertaking three tasks: 1) obtaining the probabilities of the observed rates, 2) filtering the rates for which the probability was lower than 0.05, and 3) re-rendering the map tiles to reflect the filtering results. Table 4.5 shows that while the

Table 4.4. Average response times during dynamic data query

	Raster		SVG		Canvas	
Number of areal units	Average	Margin of error ($\alpha = 0.05$*)	Average	Margin of error	Average	Margin of error
3109	156.14	5.78	100.30	2.18	195.02	3.04
6469	212.04	5.27	153.92	7.00	333.66	7.52
11680	318.30	9.14	267.56	7.95	570.82	10.50
29885	747.38	10.83	684.00	14.77	1384.90	22.55
64919	1634.60	13.96	1625.90	22.39	3097.66	29.44

time costs for the first two tasks were similar in the three tile-mapping approaches, it took more time to re-render tiles in the Canvas approach than in the other approaches. This result relates to the different processes of tile re-rendering in the tile-mapping approaches. In the raster approach, the entire set of tiles constituting the current map had to be completely re-rendered on the server side to visualize the effects of rate filtering. In the Canvas approach, the polygons in the map tiles also had to be completely redrawn and recolored to reflect new classification results. This was necessary since individual geometry primitives become inaccessible in Canvas once they are drawn. In contrast, the SVG approach allowed for direct access to geometry primitives after rendering them since they are converted into identifiable elements in a web document. Because of this feature, the effects of rate filtering could be achieved in the SVG approach by merely redefining the color attributes of the polygons. These differences in the tile re-rendering processes were unavoidable due to the unique properties of raster map tiles, Canvas, and SVG. They were the main cause of the better performance of the SVG approach observed here.

4.4 Zoom in

In the tile-mapping approaches, the zoom-in operation involved dynamic rendering of new tiles for a higher zoom level (Section 3.4.1). Because of this, the response times of the zoom-in operation showed a similar pattern to those of dynamic choropleth mapping: across all test data sets, zooming in three times took the shortest time in the raster approach, the second shortest time in the Canvas approach, and the longest time in the

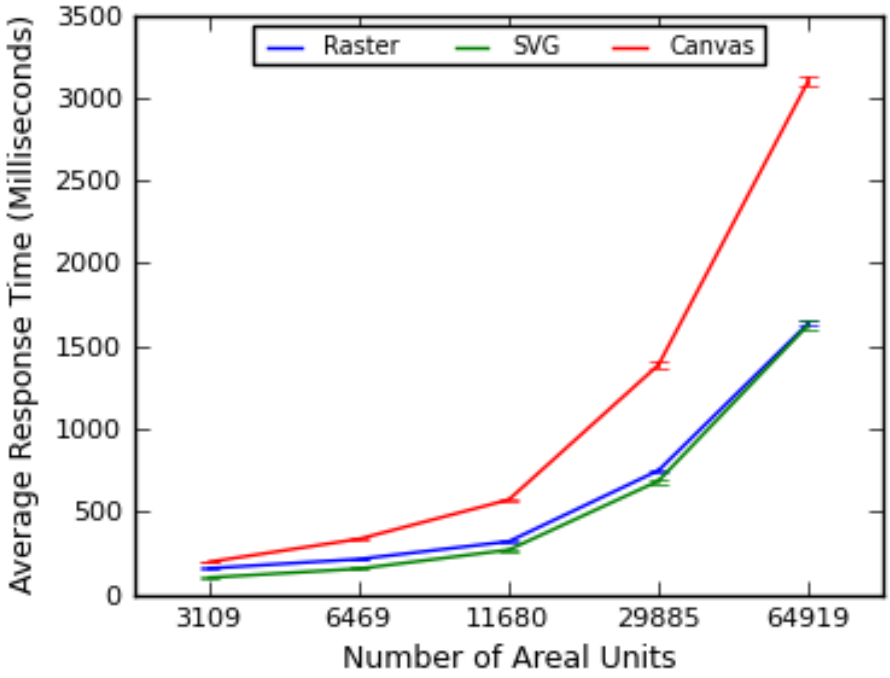


Figure 4.3. Response time chart of test applications (dynamic data query)

Table 4.5. Details of time spent on dynamic data query

Raster approach				
Number of areal units	Obtain probabilities	Filter rates	Re-render tiles	Total
3109	93.50	3.64	59.00	156.14
6469	143.70	7.48	60.86	212.04
11680	237.58	10.34	70.38	318.30
29885	586.32	28.96	132.10	747.38
64919	1320.74	70.84	243.02	1634.60
SVG approach				
3109	61.42	7.44	31.44	100.30
6469	100.20	10.42	43.30	153.92
11680	172.00	20.74	74.82	267.56
29885	445.50	52.72	185.78	684.00
64919	1097.82	119.08	409.00	1625.90
Canvas Approach				
3109	61.94	7.10	125.98	195.02
6469	102.60	10.42	220.64	333.66
11680	164.22	16.80	389.80	570.82
29885	444.70	46.46	893.74	1384.90
64919	1082.36	112.16	1903.14	3097.56

* Values in this table are in milliseconds.

Table 4.6. Average response times during zoom-in operation

	Raster		SVG		Canvas	
Number of areal units	Average	Margin of error ($\alpha = 0.05$*)	Average	Margin of error	Average	Margin of error
3109	356.44	7.20	1514.34	10.95	944.96	36.63
6469	496.78	13.64	2768.30	61.02	1340.86	14.63
11680	614.70	18.15	4735.64	71.64	2137.40	22.41
29885	1235.12	19.63	11346.96	30.90	4543.06	79.26
64919	1900.90	20.86	42223.56	1156.23	9574.22	130.28

SVG approach (Table 4.4). The time cost for the zoom-in operation also increased the least in the raster approach (1544.46 milliseconds) as the number of areal units grew from 3,109 to 64,919; the amount of the time increase was the second least in the Canvas approach (8629.26 milliseconds) and the most in the SVG approach (40709.22 milliseconds). Combined, these results indicate: 1) the zoom-in operation scaled better in the raster approach than in the other approaches; and 2) while both vector approaches were relatively inefficient, the zoom-in operation performed better in the Canvas approach than in the SVG approach.

4.5 Pan

Figure 4.5 and Table 4.7 show the average response times of the panning operation in the tile-mapping approaches. As with the zoom-in operation, the panning operation was the slowest in the SVG approach across all test data sets. In contrast, it was not always the case that the panning operation performed more efficiently in the raster approach than in the Canvas approach: when the number of areal units was in the range of 2,227 to 2,414, the time cost for the panning operation was smaller in the Canvas approach than in the raster. In addition, the time cost increased the most in the raster approach (230.84 milliseconds) as the number of areal units grew from 385 to 2,414. For the same amount of increase in the number of areal units, the time cost for the panning operation grew by 209.46 and 96.20 milliseconds in the SVG and Canvas approaches, respectively. These results suggest that when the number of areal units is small (e.g., less than 2,414 units in

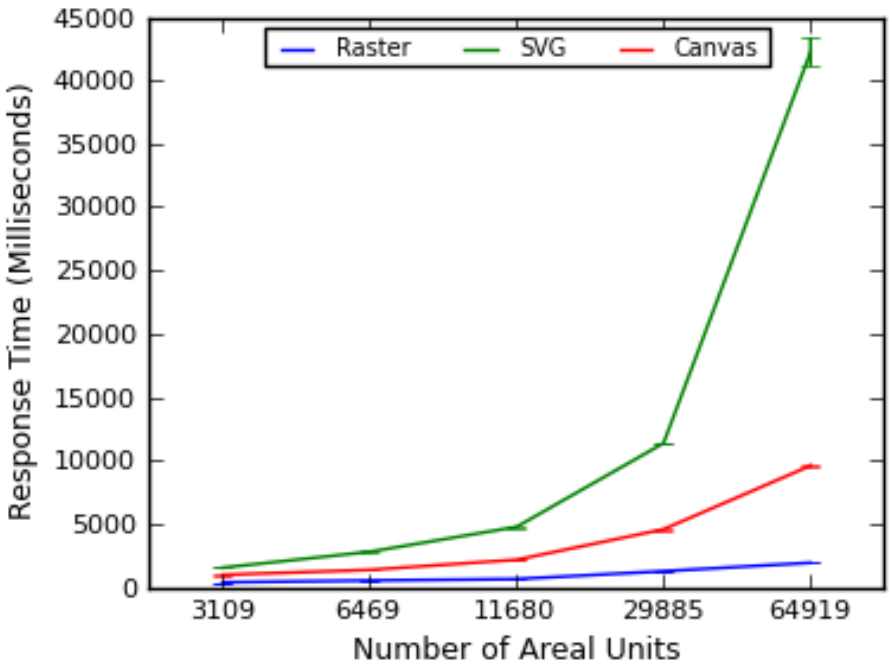


Figure 4.4. Response time chart of test applications (zoom in)

the particular case of the study), the panning operation could show a similar level of efficiency in the raster and Canvas approaches but scale better in the latter approach. Provided that new tiles obtained in each panning operation were mostly partially visible, another indication of the results is that the time cost for rendering wasted parts of map tiles would be similar between the raster and Canvas approaches but increase faster in the former than in the latter, when the number of areal units stays within a relatively small number like the 2,414 used in the study.

4.6 Summary

Overall, the results in this chapter indicate that the raster approach was more efficient and scalable in most test operations than the SVG and Canvas approaches. Particularly, the raster approach provided the best performance and scalability during dynamic choropleth mapping, map juxtaposition, and zoom-in. For the same operations, Canvas performed and scaled better than SVG. In contrast, for dynamic data query the performance of the raster approach was similar to that of the SVG approach, while the Canvas approach responded

Table 4.7. Average response times during pan operation

	Raster		SVG		Canvas	
Number of areal units**	Average	Margin of error ($\alpha = 0.05$*)	Average	Margin of error	Average	Margin of error
385	120.76	6.70	185.64	4.41	172.72	3.97
557	133.54	11.15	211.30	9.50	150.72	6.44
875	138.42	20.79	205.78	7.43	164.28	8.71
2227	198.66	10.77	309.92	6.55	192.60	3.61
2414	351.60	15.53	395.10	11.80	246.92	9.55

* α represents significance level.
** Values in this data column represent numbers of areal units within panned map extents.

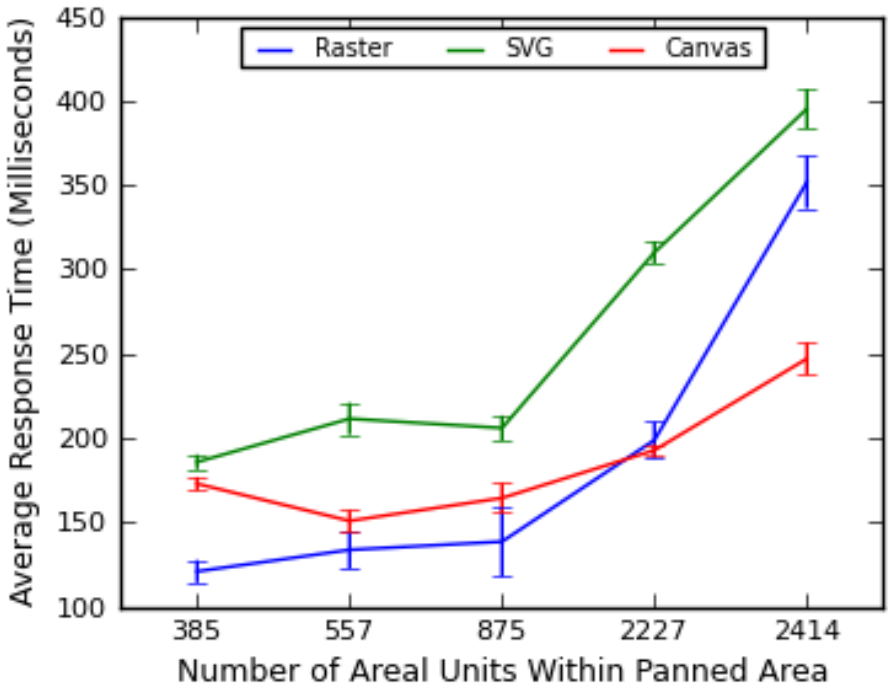


Figure 4.5. Response time chart of test applications (pan)

the most slowly, due to the need for the complete re-drawing of the polygon geometries as well as for re-shading of their colors. When the panning operation involved the addition of new map tiles that included a relatively small number of polygons (e.g., 2,414 polygons as in the study), the Canvas approach performed as efficiently as the raster approach.

CHAPTER 5

DISCUSSION

The previous chapter presented the results of this study and revealed the pros and cons of the three tile-mapping approaches in supporting various types of user interaction required for choropleth-map-based exploration of a rate variable. This chapter discusses their implications in online choropleth and other forms of online mapping. It also examines the limitations of the study.

The evaluation in this study showed that the raster approach was more efficient and scalable for the dynamic rendering of choropleth maps than the other vector approaches when benchmarked with the test applications and evaluation methods of the study. In all test operations, the raster approach performed better than or equal to the vector approaches. This result is consistent with the discussions in the literature that raster maps are more efficient for online rendering than vector maps (Peng and Zhang, 2004; Zhao and Shneiderman, 2005).

In the context of the study, the better performance of the raster approach related to three factors: tile size, geometry rendering, and the method of tile re-rendering. First, raster tiles were small in their data size while vector tiles were relatively large: the data sizes of the map tiles ranged from 267 B to 29.4 KB in the raster tiles but varied from 307 KB to 7.6 MB in the vector tiles before the gzip compression was applied (Tables 3.1 and 3.2). Consequently, it took more time to download vector tiles than raster tiles. Second, as discussed in Section 3.4.1, the conversion of geometric coordinates to an image space like a screen display was necessary in the vector approaches while it was not in the raster approach. Unsurprisingly, this need for geometry rendering presented increasingly large costs in the vector approaches as the number of areal units grew. Third, color-coded raster

tiles were re-rendered by redefining the color code of each pixel while vector tiles were re-drawn by re-specifying the color style of each polygon. Since the dimensions of a raster tile were fixed at 256×256 pixels, the speed of dynamic tile re-rendering varied little in the raster-based approach. However, the time for tile re-rendering increased with the number of areal units in the vector approaches; the color attributes of pre-rendered geometry elements had to be redefined in the SVG approach, while the entire set of geometry primitives contained in the map tiles had to be redrawn in the Canvas approach.

From the above observations and the results of the evaluation, it can be concluded that the raster approach is still more suitable for online choropleth rendering, especially when the rendering needs to be supported for spatial data with large numbers of areal units, such as statistics measured at small areal units and numerical attributes of buildings and parcels in densely populated regions. Nonetheless, a note should be added that more research is needed to understand how further optimizations would affect the scalability of choropleth mapping in the vector approaches. Such optimizations include use of different simplification algorithms, data compression methods, and file formats for vector tiles.

In terms of the vector approaches, the evaluation results in two findings. The first is stated above: the vector approaches do not perform and scale as well as the raster approach. However, they perform reasonably well when the data to be mapped are similar to the boundaries of U.S. counties and PCSAs in terms of both geometric features and number of areal units, if the computing environment of the online choropleth-mapping application is equivalent to the test environment of the evaluation. This is because for online display in general, “useful content needs to be loaded within 2 seconds” (Laird and Brennan, 2006, p.169), and Tables 4.1 ~4.7 show that most test operations in this study were completed within two seconds when the test data sets contained the boundary data of U.S. counties and PCSAs, in which several thousand areal units resided.

The second finding that concerns the vector approaches and has not been empirically corroborated by other studies in the cartography literature is that the Canvas approach is more effective for online choropleth rendering than the SVG approach. The results of the evaluation (e.g., Tables 4.1 ~4.3) show that the time spent rendering choropleth maps was usually reduced by half in the Canvas approach when compared to the SVG approach. Although this finding is applicable only to the case in which all polygons in a tile need to be (re-)rendered, it suggests that the Canvas approach would be more suitable for vector-based dynamic choropleth mapping than the SVG approach when the data contain a relatively large number of areal units. Typical use cases of such dynamic yet scalable choropleth mapping include the real-time generation of vector-based visualizations of the modeling results of large study areas, and sensor data streams aggregated at small areal units where the base geography of the mapped area is relatively static.

Although tile rendering was more efficient in the Canvas approach than in the SVG approach, one caveat should be added: it would take more effort to enable interactive manipulation of areal units in the former than in the latter. This was demonstrated in the test case of dynamic data query in which with Canvas, the interactive filtering of insignificant rates involved not only the recoloring of polygons but also the redrawing of their geometric shapes. In contrast, color attributes of individual polygons could be modified without geometry rendering in the SVG approach. This observation suggests that if advanced techniques for interactive spatial data analysis, such as linking and brushing, need to be supported in the vector-based choropleth mapping of relatively small data, SVG would be more cost-effective than Canvas, since it has built-in attributes and functions for handling vector geometry shapes. When such visualization methods need to be enabled through Canvas, it would be necessary to develop or extend client-side software frameworks that are carefully optimized for cartographic rendering and interactive data visualization.

Overall, the evaluation in this study demonstrated that the tile-based approaches could indeed handle more than several thousand areal units, beyond which the traditional methods for online choropleth mapping were reported to have performance problems (Steiner et al., 2002; Zhao and Shneiderman, 2005; Schmidt and Dev, 2008). However, the evaluation also revealed several directions for further improvements in the tile mapping approaches. In the raster tiling approach, it is difficult to support user interaction that involves direct manipulation of individual areal units. For example, user interaction such as feature selection and map linking requires continuous updates of choropleth maps as the user makes selections on the client-side screen. Since a map update involves server-side tile re-rendering in the raster tiling method, it is still challenging to provide responsive feedback to the user while he or she performs feature selection and map linking on a raster-tile-based map. A solution would be to use an additional map layer where user selections on the map were traced and selected polygons returned as a raster image or a vector data set, so as to visualize the effects of the user selections on the client side.

In the vector tiling approach, improvements are required in the generation and delivery of the vector tiles. As discussed in Sections 2.2.4 and 3.3, the LOD of a vector-tile-based map is controlled through geometry simplification or map generalization. At present, most algorithms for geometry simplification are computation-intensive and have issues regarding the maintenance of topological relationships among the geographic features of polygon geometry (Yang et al., 2007; Antoniou et al., 2009). These limits in geometry simplification often present barriers to on-demand generation of vector tiles. In addition to LOD control, it is important in vector-tile-based mapping to produce map tiles of comparably small sizes in order to avoid performance issues in tile downloading and client-side map visualization. Various methods for data encoding and compression could be applied to reduce tile size, and the method and degree of geometry simplification chosen in such a way that tile size varied little within a pre-specified threshold value (Gaffuri, 2012).

Although the study focused on online choropleth mapping, the results of the evaluation also have implications in improving the performance of other forms of online mapping through map tiles. In general, both color-coded raster tiles and vector tiles can facilitate the dynamic mapping of new data observed or computed at static geographic configurations. Examples include the visualization of statistics aggregated from live data streams (e.g., location-based social media data) or the results of on-demand spatial data analysis where the study area is fixed but the parameters for the analysis vary according to user selections. When the output data from such aggregation and analysis contain a large number of geographic features and rapid delivery of the related maps is crucial, as in the case of disaster response, the raster tiling approach would be preferable to the vector tiling one because of its scalability. In contrast, when the end user needs to interactively add new measurements for a small study area and examine their spatial distributions dynamically, as in participatory mapping, the vector tiling approach would provide the user with interactive yet responsive experience while supporting dynamic changes in map symbolization locally on the client side.

Along with the implications discussed here, it deserves mention that the results of the evaluation should be interpreted carefully due to the differences in the architectures of the raster and the two vector approaches. As indicated in Section 3.4, the raster approach requires a consistent network connection for dynamic choropleth rendering. Its efficiency also depends on the specification of server-side hardware, as color-coded raster tiles are re-rendered on the server side. In contrast, the performances of the SVG and Canvas approaches rely on network bandwidth more than that of the raster approach because the vector tiles tend to be larger than their raster counterparts. These approaches also perform differently if the specification of the client-side computer changes. For the effective delivery of online choropleth maps, researchers and practitioners need to take into account these architectural differences as well as the efficiency of dynamic choropleth rendering, in order to better satisfy the user requirements of their online-mapping applications.

Despite considerable effort to conduct an unbiased evaluation of the three tile-mapping methods, this study was limited in that it could not control for various factors that may have influenced the performance of the test applications. As discussed in Section 3.4.2, these factors included internet connection methods, network traffic intensity, browser types, hardware specifications, geometry simplification algorithms, and methods for data encoding and compression. It remains for future work to investigate specific impacts of these factors on the efficiency of tile-based choropleth-mapping approaches.

CHAPTER 6

CONCLUSION

This dissertation evaluated the strengths and weaknesses of cutting-edge tile-mapping approaches for enabling dynamic choropleth mapping of and user interaction with areal data that include large numbers of polygons.

Online choropleth mapping of large areal data is an emerging research topic in the literature on cartography, with implications for the web-based delivery of analytical cartographic visualizations and various forms of atlases. The growing interest in this topic pertains to recent developments in online mapping. End users now design and produce online maps interactively, including choropleth maps. Traditional spatial data usable for choropleth mapping have become increasingly available and accessible, especially through GIServices and mapping APIs. In addition, real-time spatial data observed from GPS and geospatial sensors are also being visualized through choropleth maps to facilitate pattern detection in such massive data. From a technological viewpoint, these advances indicate dynamic choropleth mapping will be increasingly needed for large areal data, which existing approaches cannot easily accommodate. To address this need, several tile-based methods have been proposed. For example, Schmidt and Dev (2008) developed a scheme of using color-coded raster tiles in which changing the color codes can result in a new choropleth map. Vector tiles have also been employed for dynamic choropleth mapping in combination with web graphics technologies such as SVG and Canvas (Gaffuri, 2012). While these tile-based methods demonstrated their feasibility, little knowledge is available of their actual scalability, thereby complicating technical decision making by providers of online choropleth maps.

This dissertation study sought to fill this knowledge gap in the literature of online mapping by conducting an empirical evaluation of the scalability of three existing tile-based methods for choropleth mapping: the raster tiling method of Schmidt and Dev (2008), and the SVG- and Canvas-based vector tiling methods that are implemented in the Polymaps library. For the evaluation, the study developed test applications for the raster and vector tiling methods. The evaluation then focused on examining how the response times of these applications varied as the number of areal units increased, within the boundary of the conterminous U.S. This examination was conducted with a set of test operations: dynamic choropleth mapping, map juxtaposition, dynamic data query, zooming in, and panning.

While specific to the test applications and evaluation method of this study, the evaluation resulted in two findings. The first finding is that the raster tiling method performed and scaled better in most test operations when compared to the SVG- and Canvas-based vector tiling methods. In particular, the raster method outperformed its vector equivalents during dynamic choropleth mapping, map juxtaposition, and zoom-in. This superiority of the raster method is attributable to the small size of the raster tiles, pre-rendering of geometric configurations in the tiles, and relatively invariant time cost for tile re-rendering. Given the evidence for the better scalability of the raster method, it is recommended for use cases where dynamic yet rapid choropleth rendering of maps is frequently requested for large areal data.

The second finding is that although neither vector tiling method was as scalable as the raster tiling method, the Canvas-based method was more efficient and scalable than the SVG-based method in supporting the choropleth rendering of vector tiles. While SVG-based map tiles could be recolored by simple changes in the color attributes of the polygons, the re-rendering of Canvas-based map tiles required the additional work of redrawing the polygons. Even so, tile re-rendering usually took less time in the Canvas method than in the SVG method, suggesting that dynamic vector-based rendering of

choropleth maps would become more responsive and scalable with Canvas. Despite this superior performance of the Canvas method in tile re-rendering, it should be noted that when user interactions incurred only partial updates of the tile, the SVG method was more efficient than the Canvas method, since the former allowed for direct manipulation of geometry primitives, while the latter did not. This result indicates that SVG could be more cost-effective than its Canvas counterpart if choropleth maps need to allow direct user interactions with individual geographic features.

The findings of this dissertation show that current technologies for tile mapping indeed facilitate online choropleth mapping of large areal data. The present study can be further extended in future research. The first direction for such an extension is to examine how recent technologies for web-based 3D graphics would enhance the performance of analytical cartographic visualization, including choropleth mapping. In particular, the web graphics library (WebGL) is of interest for such investigations since it allows for direct drawing of 3D graphics on web browsers through a Javascript API; it can also improve the graphics rendering speed by accessing the graphics card of the client-side computer (Lienert et al., 2012). This new generation of web graphics technologies provides opportunities to mitigate the challenge of handling large spatial data in online mapping. Yet, methodological frameworks remain to be established and evaluated to harness the innovative graphics technologies for online mapping and geovisualization.

The second direction for related research is to explore and address the challenge of scalable choropleth mapping in the context of mobile mapping. The primary concern of online choropleth mapping is to allow a wide range of users to examine spatial distributions of data variables across diverse computing environments, such as different web browsers and mobile devices. Since end users increasingly access online maps from mobile web browsers or native applications, scalable choropleth mapping would be an issue in mobile mapping. In comparison to desktop environments, mobile computing platforms are limited in terms of network connectivity, network bandwidth, and hardware

specification. New approaches that can account for such characteristics are necessary for the efficient delivery of choropleth maps across mobile networks.

Overall, this study contributes to the literature of GIScience and cartography by providing a useful benchmark of tile-based methods for online choropleth mapping of large areal data. Researchers and practitioners can benefit from the benchmark when they seek to develop online mapping applications capable of the dynamic choropleth mapping of small area statistics, real-time measurements, and other forms of distributed geospatial data that contain a large number of areal units. The benchmark should also be helpful for geovisualization researchers who want to understand the potential of the state-of-the-art online-mapping technologies for web-based geovisualization.

REFERENCES

Adnan, M., Singleton, A., and Longley, P. A. (2010). Developing efficient web-based GIS applications. Technical report, CASA. CASA Working Papers (153).

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web services: Concepts, architectures and applications*. Springer-Verlag, Berlin and Heidelberg.

Andrienko, G., Andrienko, N., Demsar, U., Dransch, D., Dykes, J., Fabrikant, S. I., Jern, M., Kraak, M. J., Schumann, H., and Tominski, C. (2010). Space, time and visual analytics. *International Journal of Geographic Information Science*, 24(10):1577–1600.

Andrienko, G., Andrienko, N., Voss, H., and Carter, J. (1999). Internet mapping for dissemination of statistical information. *Computers, Environment and Urban Systems*, 23:425–441.

Andrienko, G. L. and Andrienko, N. V. (1999). Interactive maps for visual data exploration. *International Journal of Geographic Information Science*, 13(4):355–374.

Andrienko, N., Andrienko, G., Voss, H., Bernardo, F., Hipolito, J., and Kretchmer, U. (2002). Testing the usability of interactive maps in CommonGIS. *Cartography and Geographic Information Science*, 29(4):325–342.

Anselin, L., Kim, Y.-W., and Syabri, I. (2004). Web-based analytical tools for the exploration of spatial data. *Journal of Geographical Systems*, 6:197–218.

Anselin, L., Syabri, I., and Kho, Y. (2006). GeoDa: An introduction to spatial data analysis. *Geographical Analysis*, 38(1):5–22.

Antoniou, V., Morley, J., and Haklay, M. M. (2009). Tiled vectors: A method for vector transmission over the Web. In Carswell, J. D., Fotheringham, A. S., and McArdle, G., editors, *Web and Wireless Geographical Information Systems*, pages 56–71, Berlin and Heidelberg, Germany. Springer-Verlag.

Armstrong, M. P., Xiao, N., and Bennett, D. A. (2003). Using genetic algorithms to create multicriteria class intervals for choropleth maps. *Annals of the Association of American Geographers*, 93(3):595–623.

Azzi, M., Caviglia, G., Ricci, D., Ciuccarelli, R., Bonetti, E., and Bontempi, L. (2011). Dust: A visualization tool supporting parents' school-choice evaluation process. *Parsons Journal for Information Mapping*, 3(4):1–7.

Barclay, T., Gray, J., Ekblad, S., Strand, E., and Richter, J. (2006). Designing and building TerraService. *IEEE Internet Computing*, 10(5):16–25.

Barclay, T., Gray, J., and Slutz, D. (2000). Microsoft TerraServer: A spatial data warehouse. In *Proceedings of the Nineteenth ACM SIGMOD International Conference on Management of Data*, pages 307–318. Dallas, Texas.

Batty, M., Hudson-Smith, A., Milton, R., and Crooks, A. (2010). Map mashups, Web 2.0, and the GIS revolution. *Annals of GIS*, 16(1):1–13.

Bertolotto, M. and Egenhofer, M. J. (2001). Progressive transmission of vector map data over the World Wide Web. *Geoinfomatica*, 5(4):345–373.

Boulos, M. N. K., Russell, C., and Smith, M. (2005). Web GIS in practice II: Interactive SVG maps of diagnoses of sexually transmitted diseases by Primary Care Trust in London, 1997-2003. *International Journal of Health Geographics*, 4(4):1–12.

Boulos, M. N. K., Warren, J., Gong, J., and Yue, P. (2010). Web GIS in practice VIII: HTML5 and the Canvas element for interactive online mapping. *International Journal of Health Geographics*, 9(14).

Brewer, C. A. (1996). Guidelines for selecting colors for diverging schemes on maps. *The Cartographic Journal*, 33(2):79–86.

Brewer, C. A. (1997). Spectral schemes: controversial color use on maps. *Cartography and Geographic Information Systems*, 24(4):203–220.

Brewer, C. A. (1999). Color use guidelines for data representation. In *Proceedings of the Section on Statistical Graphics, American Statistical Association*, pages 55–60, Alexandria, VA.

Brewer, C. A., Hatchard, G. W., and Harrower, M. A. (2003). ColorBrewer in print: a catalog of color schemes for maps. *Cartography and Geographic Information Science*, 30(1):5–32.

Brewer, C. A., MacEachren, A. M., Pickle, L. W., and Herrmann, D. (1997). Mapping mortality: evaluating color schemes for choropleth maps. *Annals of the Association of American Geographers*, 87(3):411–438.

Brewer, C. A. and Pickle, L. (2002). Evaluation of methods for classifying epidemiological data on choropleth maps in series. *Annals of the Association of American Geographers*, 92(4):662–681.

Brown, C., Bartley, J., Chivite, I., Szukalski, B., and Shanks, R. (2008). ArcGIS in a Web 2.0 world. http: //www.scdhec.gov/gis/presentations/ESRI_Conference_08/tws/workshops/tw_983.pdf.

Butler, D. (2006). Virtual globes: the web-wide world. *Nature*, 439:776–778.

Campin, B. (2005). Use of vector and raster tiles for middle-size Scalable Vector Graphics' mapping applications. http://www.svgopen.org/2005/papers/ VectorAndRasterTilesForMappingApplications/index.html.

Careem, M., Bitner, D., and de Silva, R. (2007). GIS integration in the Sahana Disaster Management System. In Van de Waller, B., Burghardt, P., and Nieuwenhuis, C., editors, *Proceedings of ISCRAM2007*, pages 211–218, Delft, the Netherlands. Academic & Scientific Publishers.

Cecconi, A. and Galanda, M. (2002). Adaptive zooming in web cartography. *Computer Graphics Forum*, 21(4):787–799.

Chow, T. E. (2008). The potential of Maps APIs for Internet GIS applications. *Transactions in GIS*, 12(2):179–191.

Cinnamon, J., Rinner, C., Cusimano, M. D., Marshall, S., Bekele, T., Hernandez, T., Glazier, R. H., and Chipman, M. L. (2009). Evaluating web-based static, animated and interactive maps for injury prevention. *Geospatial Health*, 4(1):3–16.

Crampton, J. W., Graham, M., Poorthuis, A., Shelton, T., Stephens, M., Wilson, M. W., and Zook, M. (2013). Beyond the geotag: situating 'Big Data' and leveraging the potential of the geoweb. *Cartography and Geographic Information Science*, 40(2):130–139.

Cromley, E. K. and Cromley, R. G. (1996). An analysis of alternative classification schemes for medical atlas mapping. *European Journal of Cancer*, 32A(9):1551–1559.

Cromley, R. G. (2005). An alternative to maximum contrast symbolization for classed choropleth mapping. *The Cartographic Journal*, 42(2):137–144.

Cromley, R. G. and Cromley, E. K. (2009). Choropleth map legend design for visualizing community health disparities. *International Journal of Health Geographics*, 8(52):1–11.

Cromley, R. G. and Ye, Y. (2006). Ogive-based legends for choropleth mapping. *Cartography and Geographic Information Science*, 33(4):257–268.

Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. (1988). The triangle processor and normal vector shader: a VLSI system for high performance graphics. *ACM SIGGRAPH Computer Graphics*, 22(4):21–30.

Dent, O. D., Torguson, J. S., and Hodler, T. W. (2009). *Cartography: thematic map design*. McGraw-Hill, New York, NY, USA, sixth edition.

Dobson, M. W. (1980). Commentary: perception of continuously shaded maps. *Annals of the Association of American Geographers*, 70(1):106–107.

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

Dykes, J. (1998). Cartographic visualization: Exploratory spatial data analysis with local indicators of spatial association using Tcl/Tk and cdv. *The Statistician*, 47(3):485–497.

Eick, S. G. and Karr, A. F. (2002). Visual scalability. *Journal of Computational and Graphical Statistics*, 11(1):22–43.

Elwood, S., Goodchild, M. F., and Sui, D. Z. (2012). Researching volunteered geographic information: spatial data, geographic research, and new social practice. *Annals of the Association of American Geographers*, 102(3):571–590.

Erl, T. (2005). *Service-Oriented Architecture (SOA): Concepts, technology, and design*. Prentice Hall, Boston, MA.

Esri (2006). Comparing vector and raster mapping for Internet applications. Technical report, Esri.

Esri (2010). ArcGIS Server 10 functionality matrix. Technical report, Esri.

Evans, B. and Sabel, C. E. (2012). Open-source web-based geographical information system for health exposure assessment. *International Journal of Health Geographics*, 11(2).

Fisher, D. (2007). Hotmap: looking at geographic attention. *IEEE Transactions on Visualization and Computer Graphics*, 13:1184–1191.

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(December):789–798.

Fu, P. and Sun, J. (2010). *Web GIS: Principles and applications*. Esri Press, Redlands, CA.

Gaffuri, J. (2012). Toward web mapping with vector data. In Xiao, N., Kwan, M.-P., Goodchild, M. F., and Shekhar, S., editors, *Geographic Information Science*, Lecture Notes in Computer Science 7478, pages 87–101, Berlin Heidelberg. Springer-Verlag.

Gahegan, M. (1999). Four barriers to the development of effective exploratory visualization tools for the geosciences. *International Journal of Geographic Information Science*, 13(4):289–309.

Gao, S., Mioc, D., Anton, F., Yi, X., and Coleman, D. J. (2008). Online GIS services for mapping and sharing disease information. *International Journal of Health Geographics*, 7(8).

geojson.org (2012). GeoJSON – JSON geometry and feature description. http://geojson.org.

Gibin, M., Singleton, A., Milton, R., Mateos, P., and Longley, P. A. (2008). An exploratory cartographic visualization of London through the Google Maps API. *Applied Spatial Analysis and Policy*, 1:85–97.

Gilmartin, P. and Shelton, E. (1989). Choropleth maps on high-resolution CRTs: the effects of number of classes and hue on communication. *cartographica*, 26(2):40–52.

Goodchild, M. F. (1989). Optimal tiling for large cartographic database. In *Proceedings of the Ninth International Conference on Computer-Assisted Cartography*, pages 444–451, Baltimore, Maryland.

Goodchild, M. F. (2007). Citizens as voluntary sensors: Spatial data infrastructure in the world of Web 2.0. *International Journal of Spatial Data Infrastructure Research*, 2:24–32.

Goodchild, M. F. and Glennon, A. (2010). Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3(3):231–241.

Grewal, M. S., Weill, L. R., and Andrews, A. P. (2007). *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons, Hoboken, New Jersey, 2nd edition edition.

Guo, D., Chen, J., MacEachren, A. M., and Liao, K. (2006). A visualization system for space-time and multivariate patterns (VIS-STAMP). *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1461–1474.

Haklay, M. (2010). How good is volunteered geographical information? A comparative study of openstreetmap and ordnance survey datasets. *Environment and Planning B*, 37:682–703.

Haklay, M., Singleton, A., and Parker, C. (2008). Web mapping 2.0: The neogeography of the geoweb. *Geography Compass*, 2(6):2011–2039.

Harrower, M. and Brewer, C. A. (2003). ColorBrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37.

Huang, B. and Lin, H. (2002). A Java/CGI approach to developing a geographic virtual reality toolkit on the Internet. *Computers and Geosciences*, 28:13–19.

Huang, B. and Worboys, M. F. (2001). Dynamic modelling and visualization on the Internet. *Transactions in GIS*, 5(2):131–139.

Huang, H., Li, Y., Gartner, G., and Wang, Y. (2011). An SVG-based method to support spatial analysis in XML/GML/SVG-based WebGIS. *International Journal of Geographic Information Science*, 25(10):1561–1574.

International Cartographic Association, Commission II (1973). *Multilingual dictionary of technical terms in cartography*. Franz Steiner, Wiesbaden, Germany.

Jenks, G. (1977). *Optimal data classification for choropleth maps*. University of Kansas, Lawrence, KN.

Jenks, G. F. and Caspall, F. C. (1971). Error on choroplethic maps: definitions, measurement, reduction. *Annals of the Association of American Geographers*, 61(2):217–244.

Jenny, B. and Kelso, N. V. (2007). Color design for the color vision impaired. *Cartographic Perspectives*, Spring(57):61–67.

Jern, M. (2009). Collaborative web-enabled geoanalytics applied to OECD regional data. In Luo, Y., editor, *Cooperative Design, Visualization, and Engineering*, volume 5738 of *Lecture Notes in Computer Science*, pages 32–43. Springer, Berlin / Heidelberg.

Jiang, B. (2012). Head/tail breaks: a new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer*, in press.

Johnson, D. W. and Jankun-Kelly, T. J. (2008). A scalability study of web-native information visualization. In *Proceedings of graphics interface 2008*, pages 163–168. Canadian Information Processing Society.

Jones, C. E. and Weber, P. (2012). Towards usability engineering for online editors of volunteered geographic information: a perspective on learnability. *Transactions in GIS*, 16(4):523–544.

json.org (2012). Introducing JSON. http://www.json.org.

Karnatak, H., Shukia, R., Sharma, V. K., Murthy, Y., and Bhanumurthy, V. (2012). Spatial mashup technology and real time data integration in geo-web application using open source GIS - a case study for disaster management. *Geocarto International*, 27(6):499–514.

Kilpelainen, T. (2000). Maintenance of multiple representation database for topographic data. *The Cartographic Journal*, 37(2):101–107.

Kraak, M. J. (2004). The role of the map in a Web-GIS environment. *Journal of Geographical Systems*, 6:83–93.

Kumar, N. (2004). Frequency histogram legend in the choropleth map: a substitute to traditional legends. *Cartography and Geographic Information Science*, 31(4):217–236.

Kwakkel, J. H., Carley, S., Chase, J., and Cunningham, S. W. (2012). Visualizing geo-spatial data in science, technology and innovation. *Technological Forecasting and Social Change*, In Press.

Laird, L. M. and Brennan, M. C. (2006). *Software measurement and estimation: A practical approach*. John Wiley & Sons, Inc., Hoboken, New Jersey.

Langfeld, D. D., Kunze, R., and Vornberger, O. (2008). Four-dimensional visualization of time- and geobased data. http://www.svgopen.org/2008/presentations/115-SVG_Web_Mapping/index.pdf.

Li, S. (2008). Web mapping/GIS services and applications. In Li, Z., Chen, J., and Baltsavias, E., editors, *Advances in Photogrammetry, Remote Sensing and Spatial Information Science: 2008 ISPRS Congress Book*, chapter 25, pages 335–353. Taylor & Francis, London, U.K.

Lienert, C., Jenny, B., Schnabel, O., and Hurni, L. (2012). Current trends in vector-based internet mapping: a technical review. In Peterson, M. P., editor, *Online maps with APIs and WebServices*, chapter 3, pages 23–36. Springer, Berlin and Heidelberg.

Lilja, D. J. (2000). *Measuring computer performance: A practitioner's guide*. Cambridge University Press, Cambridge, UK.

Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (2005). *Geographic information systems and science*. Wiley, West Sussex, England, second edition.

Luotsinen, L. J., Ekblad, J. N., Fitz-Gibbon, T. R., Houchin, C. A., Key, J. L., Khan, M. A., Lyu, J., Nguyen, J., II, R. R. O., Stein, G., Weide, S. A. V., Trinh, V., and Bölöni, L. (2007). Comparing apples with oranges: Evaluating twelve paradigms of agency. In Bordini, R. H., Dastani, M., and Dix, J., editors, *ProMAS 2006*, LNAI 4411, pages 93–112. Springer-Verlag, Berlin and Heidelberg, Germany.

MacEachren, A. M. (1982). Map complexity: comparison and measurement. *The American Cartographer*, 9(1):31–46.

MacEachren, A. M., Crawford, S., Akella, M., and Lengerich, G. (2008). Design and implementation of a model, web-based, GIS-enabled cancer atlas. *The Cartographic Journal*, 45(4):246–260.

Masó, J., Pomakis, K., and Juliá, N. (2010). OpenGIS web map tile service implementation standard. Technical report, Open Geospatial Consortium.

Mateos, P. and O'Brien, O. (2011). CensusProfiler - Creating accessible geovisualizations of the census of population. University College London Working Papers Series Paper 174.

McReynolds, T. and Blythe, D. (2005). *Advanced graphics programming using OpenGL*. Elsevier, San Francisco, CA, USA.

Mennis, J. (2006). Mapping the results of geographically weighted regression. *The Cartographic Journal*, 43(2):171–179.

Mersey, J. E. (1990). *Colour and thematic map design: the role of colour scheme and map complexity in choropleth map communication*. Cartographica Monograph No. 41. University of Toronto Press, Toronto, Canada.

Mobley, L. R., Root, E., Anselin, L., Lozano-Gracia, N., and Koschinsky, J. (2006). Spatial analysis of elderly access to primary care services. *International Journal of Health Geographics*, 5(19).

Moncrieff, S., West, G., Cosford, J., Mullan, N., and Jardine, A. (2013). An open source, server-side framework for analytical web mapping and its application to health. *International Journal of Digital Earth*, DOI:10.1080/17538947.2013.786143.

Monmonier, M. S. (1972). Contiguity-biased class-interval selection: a method for simplifying patterns on statistical maps. *Geographical Review*, 62:203–228.

Muller, J.-C. (1979). Perception of continuously shaded maps. *Annals of the Association of American Geographers*, 69(2):240–249.

Muller, J.-C. (1980). Comment in reply. *Annals of the Association of American Geographers*, 70(1):107–108.

Murray, A. T. and Shyy, T.-K. (2000). Integrating attribute and space characteristics in choropleth display and spatial data mining. *International Journal of Geographic Information Science*, 14(7):649–667.

Nah, F. F.-H. (2004). A study on tolerable waiting time: How long are web users willing to wait? *Behavior & Information Technology*, 23(3):153–163.

Neumann, A. (2012). Web mapping and web cartography. In Shekhar, S. and Xiong, H., editors, *Springer handbook of geographic information*, chapter 14, pages 273–287. Springer, Berlin Heidelberg.

Neumann, A. and Winter, A. M. (2005). Web mapping with Scalable Vector Graphics (SVG): Delivering the promise of high quality and interactive web maps. In Peterson, M. P., editor, *Maps and the Internet*, chapter 12, pages 197–220. Elsevier Ltd., Oxford, GB.

OGC (2004). OpenGIS web feature service implementation specification. Technical report, OGC.

OGC (2006a). An introduction to GeoRSS: A standards-based approach for geoenabling RSS feeds. Technical report, OGC.

OGC (2006b). OpenGIS web map server implementation specification. Technical report, OGC.

OGC (2006c). Symbology encoding implementation specification. Technical report, OGC.

OGC (2007a). OpenGIS Geography Markup Language (GML) encoding standard. Technical report, OGC.

OGC (2007b). Styled layer descriptor profile of the web map service implementation specification. Technical report, OGC.

OGC (2008). OGC KML. Technical report, OGC.

OGC (2010). OGC WCS 2.0 interface standard–core. Technical report, OGC.

Olson, J. M. and Brewer, C. A. (1997). An evaluation of color selections to accommodate map users with color-vision impairments. *Annals of the Association of American Geographers*, 87(1):103–134.

O'Reilly, T. (2005). What is Web 2.0? Design patterns and business models for the next generation of software. http://oreilly.com/web2/archive/what-is-web-20.html.

Pavlicko, P. and Peterson, M. P. (2005). Large-scale topographic web maps using Scalable Vector Graphics. *Cartographic Perspectives*, Winter(50):34–45.

Peng, Z. R. and Tsou, M. (2003). *Internet GIS: Distributed geographic information services for the Internet and wireless networks*. John Wiley & Sons, Hoboken, New Jersey.

Peng, Z. R. and Zhang, C. (2004). The roles of Geography Markup Language (GML), Scalable Vector Graphics (SVG), and Web Feature Service (WFS) specifications in the development of Internet Geographic Information Systems. *Journal of Geographical Systems*, 6:95–116.

Peterson, M. P., editor (2012a). *Online maps with APIs and WebServices*. Springer, Berlin and Heidelberg.

Peterson, M. P. (2012b). The tile-based mapping transition in cartography. In Zentai, L. and Nunez, J. R., editors, *Maps for the future*, Lecture Notes in Geoinformation and Cartography 5, chapter 13, pages 151–163. Springer-Verlag Berlin Heidelberg.

Plewe, B. (2007). Web cartography in the United States. *Cartography and Geographic Information Science*, 34(2):133–136.

Quinn, S. and Gahegan, M. (2010). A predictive model for frequently viewed tiles in a web map. *Transactions in GIS*, 14(2):193–216.

Ramos, J. A. S., Esperança, C., and Clua, E. W. G. (2009). A progressive vector map browser for the Web. *Journal of the Brazillian Computer Society*, 15(1):35–48.

Rey, S. J. and Anselin, L. (2007). PySAL: A Python library for spatial analytical methods. *The Review of Regional Studies*, 37:5–27.

Rinner, C., Moldofsky, B., Cusimano, M. D., Marshall, S., and Hernandez, T. (2011). Exploring the boundaries of web map services: the example of the Online Injury Atlas for Ontario. *Transactions in GIS*, 15(2):129–145.

Robinson, A. H. (1982). *Early thematic mapping in the history of cartography*. University of Chicago, Chicago, US.

Rohrer, R. M. and Swing, E. (1997). Web-based information visualization. *IEEE Computer Graphics and Applications*, 17(4):52–59.

Sample, J. T. and Ioup, E. (2010). *Tile-based geospatial information systems: Principles and practices*. Springer Science-Business Media, LLC, New York, Dordrecht, Heidelberg, London.

Schmidt, C. R. and Dev, B. (2008). A scalable tile map service for distributing dynamic choropleth maps. Working paper, GeoDa Center for Geospatial Analysis and Computation, Arizona State University, USA.

Schmidt, M. and Weiser, P. (2012). Web mapping services: development and trends. In Peterson, M. P., editor, *Online maps with APIs and WebServices*, chapter 2, pages 13–21. Springer, New York, Dordrecht, Heidelberg, London.

Schultz, R. B., Kerski, J. J., and Patterson, T. C. (2008). The use of virtual globes as a spatial teaching tool with suggestions for metadata standards. *Journal of Geography*, 107:27–34.

Slocum, T. A., McMaster, R. B., Kessler, F. C., and Howard, H. H. (2009). *Thematic cartography and geovisualization*. Prentice Hall, Upper Saddle River, NJ, third edition.

Steiner, E., MacEachren, A. M., and Guo, D. (2002). Developing lightweight, data-driven exploratory geo-visualization tools for the web. In Richardson, D. E. and Oosterom, P. V., editors, *Advances in Spatial Data Handling: Proceedings of 10th International Symposium on Spatial Data Handling*, pages 487–500, Berlin, Germany. Springer-Verlag.

Steiniger, S. and Hunter, A. J. (2013). The 2012 free and open source GIS software map - a guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, 39:136–150.

Stewart, J. and Kennelly, P. J. (2010). Illuminated choropleth maps. *Annals of the Association of American Geographers*, 100(3):513–534.

Takatsuka, M. and Gahegan, M. (2001). Sharing exploratory geospatial analysis and decision making using GeoVISTA studio: From a desktop to the Web. *Journal of Geographic Information and Decision Analysis*, 5(2):129–139.

Tobler, W. R. (1973). Choropleth maps without class intervals? *Geographical Analysis*, 5:262–265.

Traun, C. and Loidi, M. (2012). Autocorrelation-based regioclassification - a self-calibrating classification approach for choropleth maps explicitly considering spatial autocorrelation. *International Journal of Geographic Information Science*, 26(5):923–939.

Tsou, M. (2004). Integrating web-based GIS and image processing tools for environmental monitoring and natural resource management. *Journal of Geographical Systems*, 6:155–174.

Tsou, M. (2005). Recent development of Internet GIS. GISdevelopment.net.

Tsou, M. and Buttenfield, B. P. (2002). A dynamic architecture for distributing geographic information services. *Transactions in GIS*, 6(4):355–381.

Tsou, M.-H. (2011). Revisiting web cartography in the United States: the rise of user-centered design. *Cartography and Geographic Information Science*, 38(3):250–257.

Tsoulos, L. (2005). System design considerations for the development of an electronic statistical atlas. *Cartography and Geographic Information Science*, 32(3):181–194.

Tu, S. and Abdelguerfi, M. (2006). Web services for geographic information systems. *IEEE Internet Computing*, September/October:13–15.

Tu, S., Flanagin, M., Wu, Y., Abdelguerfi, M., Normand, E., Mahadevan, V., Ratcliff, J., and Shaw, K. (2004). Design strategies to improve performance of GIS web services. In *ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2*, volume 2, pages 444–448. IEEE Computer Society.

Tu, S. R., He, X., Li, X., and Ratcliff, J. (2001). A systematic approach to reduction of user-perceived response time for GIS web services. In *Proceedings of the Ninth ACM International Symposium on Advances in Geographic Information Systems*, pages 47–52, Atlanta, Georgia.

Turner, A. J. (2006). *Introduction to neogeography*. O'Reilly Media, Sebastopol, CA.

W3C (2003). Portable network graphics (PNG) specification (second edition). http://www.w3.org/TR/PNG/.

W3C (2011). HTML5: A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/.

Waller, L. A. and Gotway, C. A. (2004). *Applied spatial statistics for public health data*. Wiley, Hoboken, New Jersey.

Wang, F. (2006). *Quantitative methods and applications in GIS*. CRC Press, Boca Raton, Fl.

Wei, Z.-K., Oh, Y.-H., Lee, J.-D., Kim, J.-H., Park, D.-S., Lee, Y.-G., and Base, H.-H. (1999). Efficient spatial data transmission in web-based GIS. In *Proceedings of the Second International Workshop on Web Information and Data Management*, pages 38–42, Kansas City, Missouri.

Yang, B. S., Purves, R., and Weibel, R. (2007). Efficient transmission of vector data over the Internet. *International Journal of Geographic Information Science*, 21(2):215–237.

Yang, C., Wong, D., Yang, R., Kafatos, M., and Li, Q. (2005). Performance-improving techniques in web-based GIS. *International Journal of Geographical Information Science*, 19(3):319–342.

Yao, X. and Zou, L. (2008). Interoperable Internet mapping–An open source approach. *Cartography and Geographic Information Science*, 35(4):279–293.

Yau, N. (2011). *Visualizing this: The FlowingData guide to design, visualization, and statistics*. John Wiley & Sons, Indianapolis, IN.

Zaslavsky, I. (2000). A new technology for interactive online mapping with vector markup and XML. *Cartographic Perspectives*, Fall(37):12–24.

Zhang, C. and Li, W. (2005). The roles of web feature and web map services in real-time geospatial data sharing for time-critical applications. *Cartography and Geographic Information Science*, 32(4):269–283.

Zhao, H. and Shneiderman, B. (2005). Colour-coded pixel-based highly interactive web mapping for georeferenced data exploration. *International Journal of Geographical Information Science*, 19(4):413–428.

Zheng, Y., Xie, X., and Ma, W.-Y. (2010). GeoLife: a collaborative social networking service among user, location, and trajectory. *IEEE Data Engineering Bulletin*, 32(2):32–40.

Zucker, D. F. (2007). What does AJAX mean for you? *Interactions*, 14(5):10–12.

APPENDIX A

SOURCE CODE FOR TEST APPLICATIONS

A.1 Common Components

A.1.1 Base web page

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
      <title>Tile Mapping Scalability Test</title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <style type="text/css">
          @import url("example.css");
          @import url("lib/colorbrewer/colorbrewer.css");
          @import url("ratemap.css");
          html, body { height: 100%; }
          body { margin: 0px; }
      </style>
      <script type="text/javascript" src="lib/jquery/jquery.min.js"></script>
      <script type="text/javascript" src="lib/protovis/protodata.min.js?3.2"></
          script>
      <script type="text/javascript" src="lib/colorbrewer/colorbrewer.js"></
          script>
      <script type="text/javascript" src="polymaps.js"></script>
      <script type="text/javascript" src="ratemap.js"></script>
      <script type="text/javascript" src="eventobserver.js"></script>
      <script type="text/javascript" src="mapclassifier.js"></script>
      <script type="text/javascript" src="tester.js"></script>
      <script type="text/javascript">
       function hideSelectors(){
         $('#classification_selector').hide();
         $('#noclass_selector').hide();
         $('#colorscheme_selector').hide();
        }
      </script>
  </head>
  <body onload="hideSelectors();">
    <div class="tester">
    <h2>Scalability Test of Vector Tile Mapping Application</h2>
        <form name="options">
        <label>Test Operation</label>
        <select name="test_operation">
        <option value="op_1">Draw a choropleth map of raw rates</option>
        <option value="op_2">Draw two choropleth maps of raw and
        smoothed rates simultanesouly</option>
        <option value="op_3_1">Change map classification</option>
        <option value="op_3_2">Filter map by probability</option>
        <option value="op_4_1">Zoom in 3x</option>
        <option value="op_4_2">Zoom out 3x</option>
        <option value="op_4_3">Pan 3x</option>
        <option value="op_5_1">Highlight a polygon in a single map</option>
        <option value="op_5_2" selected="selected">Map linking with two maps</
            option>
        </select>
                <br />
        <label>Data</label>
        <select name="data">
        <option value="county">U.S. Counties</option>
        <option value="pcsa">U.S. Primary Care Service Areas (PCSAs)</option>
        <option value="sd">U.S. School Districts</option>
        <option value="zip">USPS Zip Code Areas</option>
        <option value="tract" selected="selected">U.S. Census Tracts</option>
        </select>
        <br />
        <label>Map Size</label>
```

```
58          <select name="mapsize">
59          <option value="">Small (400x200)</option>
60          <option value="" selected="selected">Medium (800x400)</option>
61          <option value="">Large (1200x600)</option>
62          <option value="">Larger (1600x800)</option>
63          </select>
64                  <br />
65                  <label>Number of Tests</label>
66              <input name="no_tests" value="2" size="3" />
67                  <a href="#" onclick="startTest()">Run Test</a>
68                  </form>
69          </div>
70          <select id="classification_selector">
71          <option value="Quantile" selected="selected">Quantile</option>
72          <option value="Percentile">Percentile</option>
73          <option value="Boxmap">Boxmap</option>
74          <option value="Equal_Interval">Equal Interval</option>
75          </select>
76          <select id="noclass_selector">
77          <option value="2">2</option>
78          <option value="3">3</option>
79          <option value="4">4</option>
80          <option value="5" selected="selected">5</option>
81          <option value="6">6</option>
82          <option value="7">7</option>
83          <option value="8">8</option>
84          <option value="9">9</option>
85          <option value="10">10</option>
86          </select>
87          <select id="colorscheme_selector">
88          <option value="Blues" selected="selected">Blues</option>
89          <option value="Purples">Purples</option>
90          <option value="Greens">Greens</option>
91          <option value="Oranges">Oranges</option>
92          <option value="Reds">Reds</option>
93          <option value="Greys">Greys</option>
94          </select>
95    </body>
96  </html>
```

A.1.2 Map Classifier

```
1
2   pv.Scale.percentile = function(){
3
4       var n = -1,
5           j = -1,
6           q = [],
7           d = [],
8           y = pv.Scale.linear();
9
10      function scale(x){
11          return y(Math.max(0, Math.min(j, pv.search.index(q, x) - 1)) / j);
12      };
13
14      scale.set_n = function(v){
15          n = v;
16      };
17
18      scale.get_n = function(){
19          return n;
20      };
21
22      scale.set_j = function(v){
```

```
23          j = v;
24      };
25
26      scale.get_j = function(){
27          return j;
28      };
29
30      scale.set_q = function(v){
31          q = v;
32      };
33
34      scale.get_q = function(){
35          return q;
36      };
37
38      scale.set_d = function(v){
39          d = v;
40      };
41
42      scale.get_d = function(){
43          return d;
44      };
45
46      scale.intervals = function(){
47          q[0] = d[0];
48          var percentiles = [1,10,50,90,99,100];
49          n = percentiles.length;
50          for (var i = 0; i < percentiles.length; i++){
51              q[i+1] = d[~~(percentiles[i]*1.0*(d.length - 1)/100)];
52          }
53          j = n - 1;
54          return this;
55      };
56
57      scale.domain = function(array, f){
58          if (arguments.length){
59              d = (array instanceof Array)?
60                  pv.map(array, f): Array.prototype.slice.call(arguments);
61              d.sort(pv.naturalOrder);
62              scale.intervals();
63              return this;
64          }
65      };
66
67      scale.range = function(){
68          y.range.apply(y, [0, j]);
69      };
70
71      scale.by = function(f){
72          function by(){return scale(f.apply(this, arguments));}
73          for (var method in scale) by[method] = scale[method];
74          return by;
75      };
76
77      scale.domain.apply(scale, arguments);
78      return scale;
79
80  };
81
82  pv.Scale.equal_interval = function(){
83
84      var scale = pv.Scale.percentile();
85
86      scale.intervals = function(no_class){
```

```
87          if (no_class != null)
88              scale.set_n(no_class);
89          var d = scale.get_d(),
90              n = scale.get_n(),
91              d_max = pv.max(d),
92              d_min = pv.min(d),
93              interval = (d_max - d_min)*1.0/n,
94              bps = pv.range(d_min, d_max, interval);
95          bps.push(d_max);
96          scale.set_q(bps);
97          scale.set_j(bps.length - 1);
98          return this;
99      };
100
101     scale.domain.apply(scale, arguments);
102     return scale;
103
104 };
105
106 pv.Scale.mapquantile = function(){
107
108     var scale = pv.Scale.percentile();
109
110     scale.intervals = function(no_class){
111         if (no_class != null)
112             scale.set_n(no_class);
113         var q = [],
114             n = scale.get_n(),
115             d = scale.get_d(),
116             w = 100.0/n,
117             quantiles = pv.range(0, 100+w, w);
118         if (quantiles[quantiles.length - 1] > 100.0)
119             quantiles[quantiles.length - 1] = 100.0
120         for (var i = 0; i < quantiles.length; i++){
121             q.push(d[~~(quantiles[i]*1.0*(d.length - 1)/100)]);
122         }
123
124         scale.set_q(q);
125         scale.set_j(n - 1);
126         return this;
127     };
128
129     scale.domain.apply(scale, arguments);
130     return scale;
131
132 };
133
134 pv.Scale.boxmap = function(){
135
136     var scale = pv.Scale.percentile();
137
138     scale.intervals = function(){
139         var q = [],
140             d = scale.get_d(),
141             uq = d[~~(75.0*(d.length - 1)/100)],
142             lq = d[~~(25.0*(d.length - 1)/100)],
143             md = pv.median(d),
144             ub = uq + 1.5*(uq - lq),
145             lb = lq - 1.5*(uq - lq);
146         scale.set_q([Number.NEGATIVE_INFINITY, lb, lq, md, uq, ub, Number.
                POSITIVE_INFINITY]);
147         scale.set_n(6);
148         scale.set_j(5);
149         return this;
```

```
150     };
151
152     scale.domain.apply(scale, arguments);
153     return scale;
154
155 };
```

A.2 Components for raster-based test application

A.2.1 Test tool

```
1  var rate_maps = {};
2  var myEventObserver;
3  var testResults = [];
4  var rateTimes = {"Raw_Map":[],"Spatial_Rate_Map":[]};
5  var eventLog = [];
6  var rateStartTimes = {'Raw':[], 'Spatial Rate':[]};
7  var rateEndTimes = {'Raw':[], 'Spatial Rate':[]};
8  var rateTypes = [];
9  var noPans = 0;
10 var renderedMaps = [];
11 var proxy = "PROXY-HOST";
12 var selectObserver = po.select();
13 var tileset = '';
14 var test_feats = null;
15
16 var currentTestOp;
17 var eventHeaders = {
18     0:["StartToDisplayBaseTiles","EndToDisplayBaseTiles","StartToGetRates",
19     "EndToGetRates","StartToClassify","EndToClassify","StartToRender","
           EndToRender"],
20 };
21 var timeHeaders = {
22     0:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles"],
23     3:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","GetProbs",
24     "FilterRates","CacheClasses","RerenderTiles"],
25     4:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","ZoomIn",
26     "ZoomIn","ZoomIn"],
27     6:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","Pan","Pan","
           Pan"],
28 };
29 var preZoom;
30
31 //
32 // test operations
33 //
34 function drawRateMap(map, data, map_id){
35    var sm_method = {"Raw_Map":"Raw", "Spatial_Rate_Map":"Spatial Rate"};
36    map.init(data.data, data.evt, data.pop, sm_method[map_id], data.wgt);
37 }
38
39 function changeClassification(map){
40    map.initTime = (new Date()).getTime();
41    var classifier = $('#' + map.container + '_clsf');
42    classifier.val("Percentile");
43    classifier.change();
44 }
45
46 function filterByProb(map){
47    map.initTime = (new Date()).getTime();
48    var probFilter = $('#' + map.container + '_prob_filter');
49    probFilter.val("0.05");
50    probFilter.change();
51 }
```

```
52
53  function zoomIn3x(map){
54     map.initTime = (new Date()).getTime();
55     eventLog.push(["Raw_Map", "Starting to zoom in", map.initTime]);
56     //console.log((new Date()).getTime());
57     map.map.zoomBy(1);
58  }
59
60  function zoomOut3x(map){
61     map.initTime = (new Date()).getTime();
62     map.map.zoomBy(-1);
63  }
64
65  function pan3x(map){
66     noPans = 0;
67     map.initTime = (new Date()).getTime();
68     eventLog.push(["Raw_Map", "Starting to pan", map.initTime]);
69     map.map.panBy({x:256, y:256});
70  }
71
72  function centroid(coords){
73     var area = 0.0,
74         cx = 0.0,
75         cy = 0.0;
76     for (var i = 0; i < (coords.length - 1); i++){
77         var p1 = coords[i],
78             p2 = coords[i+1];
79         var areaInc = (p1[0]*p2[1] - p2[0]*p1[1]);
80         area += areaInc;
81         cx += (p1[0] + p2[0])*areaInc;
82         cy += (p1[1] + p2[1])*areaInc;
83     }
84     cx = cx/(3*area);
85     cy = cy/(3*area);
86     return {lon:cx, lat:cy};
87  }
88
89  function getTargetTile(tiles){
90     var tileKeys = [];
91     for (var tile in tiles) tileKeys.push(tile);
92     var targetTile = null;
93     while (targetTile == null){
94         var tKey = tileKeys[Math.floor(tileKeys.length*Math.random())];
95         targetTile = tiles[tKey];
96     }
97     return targetTile;
98  }
99
100 function getTargetFeature(tiles){
101    var targetTile = getTargetTile(tiles);
102    var targetFeat = null;
103    while (targetFeat == null){
104        var aFeat = targetTile[Math.floor(targetTile.length*Math.random())];
105        if (aFeat && aFeat.element) targetFeat = aFeat;
106    }
107    return targetFeat;
108 }
109
110 function getFeatForSmallMap(){
111    var feat = [-360, -360];
112    while (!((feat[0] >= -113.578125 && feat[0] <= -78.421875) &&
113             (feat[1] >= 31.847258664621535 && feat[1] <= 45.49674343466975)))
114        feat = test_feats.features[Math.floor(100*Math.random())].geometry.
               coordinates;
```

```
115     return feat;
116 }
117
118 function highlightPolyInOneMap(map){
119     var feat = test_feats.features[Math.floor(100*Math.random())].geometry.
            coordinates;
120     var point = map.map.locationPoint({lon:feat[0],lat:feat[1]});
121     var evt = document.createEvent("MouseEvents");
122     evt.initMouseEvent("click", true, true, window, 0, 0, 0, point.x, point.y,
123     false, false, false, false, 0, null);
124     map.initTime = (new Date()).getTime();
125     map.map.container().dispatchEvent(evt);
126 }
127
128 function linkMaps(maps){
129     var feat = test_feats.features[Math.floor(100*Math.random())].geometry.
            coordinates;
130     var point = maps["Raw_Map"].map.locationPoint({lon:feat[0],lat:feat[1]});
131     var evt = document.createEvent("MouseEvents");
132     evt.initMouseEvent("click", true, true, window, 0, 0, 0, point.x, point.y,
133     false, false, false, false, 0, null);
134     maps["Spatial_Rate_Map"].initTime = (new Date()).getTime();
135     maps["Raw_Map"].map.container().dispatchEvent(evt);
136 };
137
138 //
139 // utility functions
140 //
141 function debug(logTxt){
142     if (window.console != undefined) {
143         console.log(logTxt);
144     } else if (opera) {
145         opera.postError(logTxt);
146     }
147 }
148
149 function avg(results) {
150     var sum = 0;
151     for (i = 0; i < results.length; i++) {
152         sum += results[i];
153     }
154     return sum/results.length;
155 }
156
157 function sleep(milliseconds){
158     var start = new Date().getTime();
159     while (new Date().getTime() < start + milliseconds);
160 }
161
162 function destroyMap(map){
163     var cont = $("#" + map.container);
164     if (cont) {
165         cont.prev().remove();
166         cont.next().remove();
167         cont.remove();
168     }
169     if (map.container in rate_maps){
170         map.layer.cache.size(0);
171         delete rate_maps[map.container];
172     }
173     delete map.container;
174 }
175
176 function resetEvtObserver(){
```

```
177     myEventObserver = null;
178 }
179
180 //
181 // evaluation functions
182 //
183 function startTest(){
184     debug('test started');
185     var testOp = getTestOp();
186     currentTestOp = testOp;
187     var data = getTestData();
188     var mapSize = getMapSize();
189     var noTests = getNoTests();
190     setupAndRunTest(testOp, mapSize, data, noTests);
191 }
192
193 function setupAndRunTest(testOp, mapSize, data, noTests){
194     testResults = [];
195     prepareTest(testOp, mapSize, data, noTests);
196     prepareTestRun(testOp, mapSize, data);
197     if (testOp < 2) runTest(testOp, data);
198 }
199
200 function addResult0(edata){
201     testResults.push(parseFloat(edata[2]) - rate_maps["Raw_Map"].initTime);
202     console.log('time diff:' + rate_maps["Raw_Map"].initTime + "-" + edata[2] +
        "=" +
203     testResults[testResults.length -1]);
204 }
205
206 function addResult1(edata){
207     addResult0(edata);
208 }
209
210 function addResult2(edata){
211     testResults.push(parseFloat(edata[2]) - rate_maps["Spatial_Rate_Map"].
        initTime);
212     console.log('time diff:' + rate_maps["Spatial_Rate_Map"].initTime + "-" +
        edata[2] +
213     "=" + testResults[testResults.length -1]);
214 }
215
216 function nextTest0(noTests, testOp, mapSize, data, srcMap){
217     if (testResults.length < noTests){
218         window.setTimeout(function(){
219             prepareTestRun(testOp, mapSize, data);
220             runTest(testOp, data);
221             }, 2);
222     }
223     else if (testResults.length == noTests && currentTestOp == 1) {
224         debug("Avg time: " + (avg(testResults) + 2.0)  + " ms");
225         var eHeader = eventHeaders[currentTestOp - 1];
226         var tHeader = timeHeaders[currentTestOp - 1];
227         var eHeaderLen = eHeader.length*2;
228         var tHeaderLen = tHeader.length;
229         debug("Response times");
230         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
231         for (var i = 0; i < testResults.length; i++) {
232             debug("Test" + i);
233             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
234             for (var j = 0; j < l.length; j++){
235                 debug(l[j].join(","));
236             }
237         }
```

```
238         debug("Raw_Map");
239         debug(tHeader.join(","));
240         for (var i = 0; i < testResults.length; i++) {
241             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
242             debug(l.join(","));
243         }
244         debug("Spatial_Rate_Map");
245         debug(tHeader.join(","));
246         for (var i = 0; i < testResults.length; i++) {
247             var l = rateTimes["Spatial_Rate_Map"].slice(i*tHeaderLen, (i+1)*
                    tHeaderLen);
248             debug(l.join(","));
249         }
250     }
251     else if (testResults.length == noTests) {
252         debug("Avg time: " + (avg(testResults) + 2.0)  + " ms");
253         var eHeader = eventHeaders[currentTestOp];
254         var tHeader = timeHeaders[currentTestOp];
255         var eHeaderLen = eHeader.length;
256         var tHeaderLen = tHeader.length;
257         debug("Response times");
258         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
259         for (var i = 0; i < testResults.length; i++) {
260             debug("Test" + i);
261             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
262             for (var j = 0; j < l.length; j++){
263                 debug(l[j].join(","));
264             }
265         }
266         debug(tHeader.join(","));
267         for (var i = 0; i < testResults.length; i++) {
268             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
269             debug(l.join(","));
270         }
271     }
272 }
273
274 function nextTest1(noTests, testOp, mapSize, data, srcMap){
275     console.log('nextTest1');
276     nextTest0(noTests, testOp, mapSize, data, srcMap);
277 }
278
279 function nextTest2(noTests, testOp, mapSize, data, srcMap){
280     console.log('nextTest2');
281     if (testResults.length < noTests){
282         window.setTimeout(function(){
283             prepareTestRun(testOp, mapSize, data);
284             }, 1);
285     }
286     else if (testResults.length == noTests && currentTestOp == 3) {
287         debug("Avg time: " + (avg(testResults) + 2.0)  + " ms");
288         var tHeader = timeHeaders[currentTestOp];
289         var eHeaderLen = 15;
290         var tHeaderLen = tHeader.length;
291         debug("Response times");
292         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
293         for (var i = 0; i < testResults.length; i++) {
294             debug("Test" + i);
295             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
296             for (var j = 0; j < l.length; j++){
297                 debug(l[j].join(","));
298             }
299         }
300         debug(tHeader.join(","));
```

```
301         for (var i = 0; i < testResults.length; i++) {
302             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
303             debug(l.join(","));
304         }
305     }
306     else if (testResults.length == noTests) {
307         var addition = 2;
308         if (testOp == 4 || testOp == 5 || testOp == 6) addition = (4-1000);
309         else if (testOp == 7 || testOp == 8) addition = 0;
310         debug("Avg time: " + (avg(testResults) + addition)  + " ms");
311         debug("Response times");
312         for (var i = 0; i < testResults.length; i++) debug(testResults[i] +
                addition);
313         var tHeader = timeHeaders[currentTestOp];
314         var eHeaderLen = 13;
315         var tHeaderLen = tHeader.length;
316         for (var i = 0; i < testResults.length; i++) {
317             debug("Test" + i);
318             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
319             for (var j = 0; j < l.length; j++){
320                 debug(l[j].join(","));
321             }
322         }
323         debug(tHeader.join(","));
324         for (var i = 0; i < testResults.length; i++) {
325             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
326             debug(l.join(","));
327         }
328     }
329 }
330
331 function prepareTest(testOp, mapSize, data, noTests){
332     var addResult = {0: addResult0, 1: addResult1};
333     var runNextTest = {0: nextTest0, 1: nextTest1, 2: nextTest2};
334     var runTestOp2_3 = function(evtData){
335         console.log(evtData);
336         var edata = evtData.split('/');
337         addResult0(edata);
338         var pMap = rate_maps["Raw_Map"];
339         $.ajax(
340             {
341                 url: proxy + '/tileservice/raster/service_proxy/reset',
342                 data: {cl_id:pMap.classificationi_id, cs_id:pMap.colorscheme_id
                        },
343                 type: 'POST',
344                 dataType: 'jsonp',
345                 success: function(res, stat, xhr){
346                     if (res.success) nextTest2(noTests, testOp, mapSize, data);
347                 }
348             }
349         );
350     };
351     var runTestOp7 = function(evtData){
352         console.log(evtData);
353         var edata = evtData.split('/');
354         addResult0(edata);
355         var pMap = rate_maps["Raw_Map"];
356         window.setTimeout(function(){nextTest2(noTests, testOp, mapSize, data)
                ;}, 500);
357     };
358     var runTestOp8 = function(evtData){
359         console.log(evtData);
360         var edata = evtData.split('/');
361         addResult2(edata);
```

```
362         var pMap = rate_maps["Spatial_Rate_Map"];
363         window.setTimeout(function(){nextTest2(noTests, testOp, mapSize, data)
                ;}, 500);
364     };
365     if (myEventObserver == null) myEventObserver = new MessageEventHandler();
366     if (testOp == 0) {
367         myEventObserver.addEventListener("layerRendered", function(evtData){
368             console.log(evtData);
369             var edata = evtData.split('/');
370             addResult[testOp](edata);
371             $.ajax(
372                 {
373                     url: proxy + '/tileservice/raster/service_proxy/reset',
374                     data: {cl_id:rate_maps["Raw_Map"].classification_id,
375                     cs_id:rate_maps["Raw_Map"].colorscheme_id},
376                     type: 'POST',
377                     dataType: 'jsonp',
378                     success: function(res, stat, xhr){
379                         if (res.success == 'true')
380                             runNextTest[testOp](noTests, testOp, mapSize, data,
                                edata[1]);
381                     }
382                 }
383             );
384         });
385         myEventObserver.addEventListener("loadEnd", function(evtData){
386             console.log(evtData);
387             var edata = evtData.split('/');
388             var loadEndTime = parseInt(edata[2]);
389             eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
390             base_tile_display = loadEndTime - rate_maps["Raw_Map"].initTime;
391             rateTimes["Raw_Map"].push(base_tile_display);
392             console.log('base tile display (t1-t0):' + base_tile_display);
393             rate_maps["Raw_Map"].startGettingRates();
394         });
395     }
396     else if (testOp == 1) {
397         myEventObserver.addEventListener("layerRendered", function(evtData){
398             console.log(evtData);
399             var edata = evtData.split('/');
400             renderedMaps.push(edata[1]);
401             if (renderedMaps.length == 2){
402                 addResult[testOp](edata);
403                 var qs_data = {};
404                 qs_data["cl_id"] = rate_maps["Raw_Map"].classification_id + ","
                        +
405                 rate_maps["Spatial_Rate_Map"].classification_id;
406                 qs_data["cs_id"] = rate_maps["Raw_Map"].colorscheme_id + "," +
407                 rate_maps["Spatial_Rate_Map"].colorscheme_id;
408                 $.ajax(
409                     {
410                         url: proxy + '/tileservice/raster/service_proxy/reset',
411                         data: qs_data,
412                         type: 'POST',
413                         dataType: 'jsonp',
414                         success: function(res, stat, xhr){
415                             if (res.success == 'true')
416                                 runNextTest[testOp](noTests, testOp, mapSize,
                                        data,
417                                 edata[1]);
418                         }
419                     }
420             );
421             }
```

```
422         });
423         myEventObserver.addEventListener("loadEnd", function(evtData){
424             console.log(evtData);
425             var edata = evtData.split('/');
426             var map_type = edata[1];
427             var loadEndTime = parseInt(edata[2]);
428             eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
429             base_tile_display = loadEndTime - rate_maps[edata[1]].initTime;
430             rateTimes[edata[1]].push(base_tile_display);
431             console.log(edata[1] + ':base tile display (t1-t0):' +
                    base_tile_display);
432             rate_maps[edata[1]].startGettingRates();
433         });
434     }
435     else if (testOp >= 2 && testOp <= 8){
436         if (testOp != 4 && testOp != 5)
437             myEventObserver.addEventListener("loadEnd", function(evtData){
438                 console.log(evtData);
439                 var edata = evtData.split('/');
440                 var loadEndTime = parseInt(edata[2]);
441                 eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
442                 base_tile_display = loadEndTime - rate_maps[edata[1]].initTime;
443                 rateTimes[edata[1]].push(base_tile_display);
444                 console.log('base tile display (t1-t0):' + base_tile_display);
445                 rate_maps[edata[1]].startGettingRates();
446             });
447
448         myEventObserver.addEventListener("readyForTest", function(evtData){
449             console.log(evtData);
450             var edata = evtData.split('/');
451             runTest(parseInt(edata[1]), data);
452         });
453
454         if (testOp == 2 || testOp == 3 || testOp == 7 || testOp == 8){
455             if (testOp != 8){
456                 myEventObserver.addEventListener("layerRendered", function(
                        evtData){
457                     console.log(evtData);
458                     window.setTimeout(function(){
459                         window.postMessage("evt://readyForTest/" + String(testOp
                                ), "*");
460                     },5);
461                 });
462             }
463             else
464                 myEventObserver.addEventListener("layerRendered", function(
                        evtData){
465                     console.log(evtData);
466                     var edata = evtData.split('/');
467                     renderedMaps.push(edata[1]);
468                     if (renderedMaps.length == 2)
469                         window.postMessage("evt://readyForTest/" + String(testOp
                                ), "*");
470                  });
471             if (testOp == 2)
472                 myEventObserver.addEventListener("classificationUpdated",
                        runTestOp2_3);
473             else if (testOp == 3)
474                 myEventObserver.addEventListener("filteredByProb", runTestOp2_3)
                        ;
475             else if (testOp == 7)
476                 myEventObserver.addEventListener("featHighlighted", runTestOp7);
477             else if (testOp == 8)
```

```
478                 myEventObserver.addEventListener("featHighlighted", function(
                        evtData){
479                     console.log(evtData);
480                     var edata = evtData.split('/');
481                     if (edata[1] == "Spatial_Rate_Map")
482                         runTestOp8(evtData);
483                 });
484         }
485         else if (testOp == 4 || testOp == 5){
486             var zoomInterval = testOp == 5 ? -1: 1;
487             var evtName = testOp == 5? 'zoomOut3x': 'zoomIn3x';
488             var goalZ = testOp == 5? 4 : 7;
489             myEventObserver.addEventListener("layerRendered", function(evtData){
490                 console.log(evtData);
491                 console.log("zoom level:" + rate_maps["Raw_Map"].map.zoom());
492                 preZoom = (new Date()).getTime();
493                 myEventObserver.addEventListener("loadEnd", function(evtData){
494                     console.log(evtData);
495                     var edata = evtData.split('/');
496                     var curZoom = parseInt(edata[2]);
497                     eventLog.push(["Raw_Map", "Ending to zoom in", curZoom]);
498                     if (rate_maps["Raw_Map"].map.zoom() == 5)
499                         rateTimes["Raw_Map"].push(curZoom - preZoom);
500                     else
501                         rateTimes["Raw_Map"].push(curZoom - preZoom - 500);
502                     preZoom = curZoom;
503                     console.log("zoom level:" + rate_maps["Raw_Map"].map.zoom())
                            ;
504                     var z = rate_maps["Raw_Map"].map.zoom();
505                     if (z != goalZ) window.setTimeout(function(){
506                         rate_maps["Raw_Map"].map.rendered = {};
507                         eventLog.push(["Raw_Map", "Starting to zoom in", (new
                                Date()).getTime()]);
508                         rate_maps["Raw_Map"].map.zoomBy(zoomInterval);
509                         }, 500);
510                     else {
511                         window.postMessage("evt://" + evtName + "/Raw_Map/" +
                                edata[2], "*");
512                     }
513                 });
514                 runTest(testOp, data);
515             });
516             myEventObserver.addEventListener(evtName, runTestOp2_3);
517         }
518         else if (testOp == 6){
519             myEventObserver.addEventListener("layerRendered", function(evtData){
520                 console.log(evtData);
521                 preZoom = (new Date()).getTime();
522                 myEventObserver.addEventListener("loadEnd", function(evtData){
523                     console.log(evtData);
524                     var edata = evtData.split('/');
525                     var curZoom = parseInt(edata[2]);
526                     eventLog.push(["Raw_Map", "Ending to pan", curZoom]);
527                     if (noPans == 0)
528                         rateTimes["Raw_Map"].push(curZoom - preZoom);
529                     else
530                         rateTimes["Raw_Map"].push(curZoom - preZoom - 500);
531                     preZoom = curZoom;
532                     noPans += 1;
533                     if (noPans < 3) window.setTimeout(function(){
534                         rate_maps["Raw_Map"].map.rendered = {};
535                         eventLog.push(["Raw_Map", "Starting to pan", (new Date()
                                ).getTime()]);
536                         rate_maps["Raw_Map"].map.panBy({x:256, y:256});
```

```
                        //noPans += 1;
                        }, 500);
                    else {
                        window.postMessage("evt://pan3x/Raw_Map/" + edata[2], "*
                            ");
                    }
                });
                runTest(testOp, data);
            });
            myEventObserver.addEventListener("pan3x", runTestOp2_3);
        }
    }
}

function prepareTestRun(testOp, mapSize, data){
    var origTestOp = testOp;
    if (testOp >= 2 && testOp <= 7) testOp = 0;
    else if (testOp == 8) testOp = 1;
    var z = (origTestOp == 5 || origTestOp == 6)? 7: 4;
    if (testOp == 0 || testOp == 1) {
        var map_ids = ["Raw_Map"];
        if (testOp == 1) map_ids.push("Spatial_Rate_Map");
        for (var i = 0; i < map_ids.length; i++){
            var map_id = map_ids[i];
            if (map_id in rate_maps) destroyMap(rate_maps[map_id]);
            var clsf = {
                method: 'Quantile',
                nocls: 5,
                colorscheme: 'Blues'
            };
            rate_maps[map_id] = new RateMap(map_id, mapSize, null, z, null, clsf
                );
        }
        if (testOp == 1) renderedMaps = [];
    }
    if (origTestOp >= 2 && origTestOp <= 7){
        if (origTestOp == 4 || origTestOp == 5 || origTestOp == 6)
                myEventObserver.removeEventListener("loadEnd");
        drawRateMap(rate_maps["Raw_Map"], data, "Raw_Map");
        if (origTestOp == 4 || origTestOp == 6){
            var t = (new Date()).getTime();
            rateTimes["Raw_Map"].push(t - rate_maps["Raw_Map"].initTime);
            rate_maps["Raw_Map"].startGettingRates();
        }
        if (origTestOp == 7) {
            sessionStorage.clear();
            delete selectObserver;
            selectObserver = po.select();
            selectObserver.add(rate_maps["Raw_Map"].selectionLayer);
        }
    }
    else if (origTestOp == 8){
       renderedMaps = [];
       sessionStorage.clear();
       selectObserver = po.select();
       for (var map_id in rate_maps){
           drawRateMap(rate_maps[map_id], data, map_id);
           selectObserver.add(rate_maps[map_id].selectionLayer);
       }
    }
}

function runTest(testOp, data){
    console.log('runTest' + testOp);
```

```
599     var func = getTestFunction(testOp);
600     if (testOp == 0 || testOp == 1) {
601         var map_ids = ["Raw_Map"];
602         if (testOp == 1) map_ids.push("Spatial_Rate_Map");
603         for (var i = 0; i < map_ids.length; i++){
604             var map_id = map_ids[i];
605             func(rate_maps[map_id], data, map_id);
606         }
607         return true;
608     }
609     else if (testOp >= 2 && testOp <= 7){
610         console.log('test operation ' + testOp + ' starts..');
611         func(rate_maps["Raw_Map"], data, "Raw_Map");
612     }
613     else if (testOp == 8){
614         console.log('test operation ' + testOp + ' starts..');
615         func(rate_maps);
616     }
617 }
618
619 function getTestOp(){
620     return document.forms["options"].test_operation.selectedIndex;
621 }
622
623 function getTestFunction(op){
624     if (op == 0 || op == 1) return drawRateMap;
625     else if (op == 2) return changeClassification;
626     else if (op == 3) return filterByProb;
627     else if (op == 4) return zoomIn3x;
628     else if (op == 5) return zoomOut3x;
629     else if (op == 6) return pan3x;
630     else if (op == 7) return highlightPolyInOneMap;
631     else if (op == 8) return linkMaps;
632 }
633
634 function getTestData(){
635     var mapdata = document.forms["options"].data.selectedIndex;
636     if (mapdata == 0){
637         tileset = 'mhwang4:1F7D3BE8E027C405D8925A66C38A692E';
638         test_feats = test_feats1;
639         return {data: 'unemployment_2009_main_', evt: 'UNEMP', pop: 'POP1',
640         wgt: 'unemployment_2009_main.gal'};
641     }
642     else if (mapdata == 1){
643         tileset = 'mhwang4:175F93390222A7E1BDE019FDA4EDFCA9';
644         test_feats = test_feats2;
645         return {data: 'pcsa_mainland_', evt: 'SUM_Sum_AG', pop: 'SUM_Sum_PO',
646         wgt: 'pcsa_mainland_q1.gal'};
647     }
648     else if (mapdata == 2){
649         tileset = 'mhwang4:0027C0AFE2E03586D6719B96EBEB70A9';
650         test_feats = test_feats3;
651         return {data: 'sd_from_tracts_', evt: 'SUM_AGE_65', pop: 'SUM_POP200',
652         wgt: 'sd_from_tracts_q1.gal'};
653     }
654     else if (mapdata == 3){
655         tileset = 'mhwang4:F4E47B01C0BC68B5E9ED4E8EADADD554';
656         test_feats = test_feats4;
657         return {data: 'usps_zip_', evt: 'Over_65', pop: 'Sum_POP200',
658         wgt: 'zip_mainland_q1.gal'};
659     }
660     else if (mapdata == 4){
661         tileset = 'mhwang4:A1CB5A8404F8051CE46ECE31DFE35EF9';
662         test_feats = test_feats5;
```

```
663         return {data: 'tracts_', evt: 'AGE_65_UP', pop: 'POP2000',
664         wgt: 'tracts_q1.gal'};
665     }
666     return null;
667 }
668
669 function getMapSize(){
670     var res = {width: null, height: null};
671     var mapsize_inx = document.forms["options"].mapsize.selectedIndex;
672     if (mapsize_inx == 0) {
673         res.width = 400;
674         res.height = 200;
675     } else if (mapsize_inx == 1){
676         res.width = 800;
677         res.height = 400;
678     } else if (mapsize_inx == 2){
679         res.width = 1200;
680         res.height = 600;
681     } else {
682         res.width = 1600;
683         res.height = 800;
684     }
685     return res;
686 }
687
688 function getNoTests(){
689     return parseInt(document.forms["options"].no_tests.value);
690 }
691
692 function hex(x){
693     return ("0" + parseInt(x).toString(16)).slice(-2);
694 }
695
696 function rgb2hex(rgb){
697    rgb = rgb.match(/^rgb\((\d+),\s*(\d+),\s*(\d+)\)$/);
698    return "#" + hex(rgb[1]) + hex(rgb[2]) + hex(rgb[3]);
699 }
```

A.2.2 Map component

```
1  var po = org.polymaps;
2
3  function RateMap(container, size, center, zoom, zoomRange, classification){
4
5      this.container = container;
6      this.controller = this.container + '_controller';
7      this.map = null;
8      this.width = 800;
9      this.height = 400;
10     this.mapClass = "map_800_400";
11     if (size != null){
12         this.width = size.width;
13         this.height = size.height;
14         this.mapClass = "map_" + String(this.width) + "_" + String(this.height);
15     }
16     if ($("#" + container).length == 0){
17         $("body").append('<br><div id="' + container + '" class="' +
18         this.mapClass + '"></div><br>');
19     }
20     this.center = {lat: 39, lon: -96};
21     if (center != null) this.center = center;
22     this.zoom = 4;
23     if (zoom != null) this.zoom = zoom;
24     this.zoomRange = [1,18];
```

```
25      if (zoomRange != null) this.zoomRange = zoomRange;
26      this.classification = 'Quantile';
27      if ('method' in classification)
28          this.classification = classification.method;
29      this.nocls = 5;
30      if ('nocls' in classification)
31          this.nocls = classification.nocls;
32      this.colorscheme = "Blues";
33      if ('colorscheme' in classification)
34          this.colorscheme = classification.colorscheme;
35      this.qs = null;
36      this.layer = null;
37      this.rates = null;
38      this.probabilities = null;
39      this.tiles = null;
40      this.classes = null;
41      this.classification_id = '';
42      this.colorscheme_id = '';
43      this.url_base = proxy + "/gws_sqlite/dyntm/t/?ts=" + tileset;
44      this.legend = null;
45      this.prob_threshold = null;
46      this.initTime = null;
47      this.rendered = {};
48  }
49
50  RateMap.prototype = {
51
52      draw_map: function(){
53          if (this.map != null) return;
54
55          this.map = po.map()
56              .size({x:this.width, y:this.height})
57              .container(document.getElementById(this.container).appendChild(po.
                    svg("svg")))
58              .center(this.center)
59              .zoom(this.zoom)
60              .zoomRange(this.zoomRange)
61              .add(po.interact());
62
63          var pMap = this;
64          function tile_url_template(t){
65              var url = (pMap.url_base + '&cl=' + pMap.classification_id);
66              url +=  ('&cs=' + pMap.colorscheme_id);
67              url += ('&x=' + t.column + "&y=" + t.row + "&z=" + t.zoom);
68              url += ('&time=' + String((new Date()).getTime()));
69              return url;
70          };
71
72          this.layer = po.image().url(tile_url_template).log(false)
73                      .on("load", function(t){
74                          //var t1 = (new Date()).getTime();
75                          pMap.rendered[t.tile.key] = true;
76                          var all_rendered = true;
77                          for(var lock in pMap.layer.cache.locks()){
78                              if (!(lock in pMap.rendered)){
79                                  all_rendered = false;
80                                  break;
81                              }
82                          }
83                          if (all_rendered)
84                              window.postMessage("evt://loadEnd/" + pMap.
                                    container +
85                              "/" + String((new Date()).getTime()), "*");
86                      });
```

```
87
88          this.map.add(this.layer);
89
90          this.selectionLayer = po.geoJson().tile(false).log(false).tile_reload(
                true);
91          this.map.add(this.selectionLayer);
92          var pMap = this;
93          function clickEvtDispatcher(e){
94              if (pMap.map.mouseEvt() != null) {
95                  pMap.map.mouseEvt(null);
96                  return;
97              }
98              var p = {x:e.clientX, y:e.clientY},
99                  loc = pMap.map.pointLocation(p),
100                 tile = pMap.map.locationCoordinate(pMap.map.pointLocation({x:p.x
                        , y:p.y})),
101                 evt = {
102                     type: 'click',
103                     tile: {column: Math.floor(tile.column), row: Math.floor(tile
                            .row),
104                     zoom: tile.zoom},
105                     select: e.target.tagName == 'image'?
106                     true: !($(e.target).attr('class').baseVal == ''),
107                     layers: pMap.qs.data,
108                     bbox: [String(loc.lon), String(loc.lat),
109                     String(loc.lon + 0.0000000001), String(loc.lat +
                            0.0000000001)].
110                     join(',')
111                 };
112             selectObserver.dispatch(evt);
113         };
114         this.selectionLayer.container().setAttribute("class", "selection_layer")
                ;
115         this.map.container().addEventListener("click", clickEvtDispatcher, false
                );
116         this.selectionLayer.on('show', function(e){
117             for(var i = 0; i < e.features.length; i++)
118                 e.features[i].element.onclick = clickEvtDispatcher;
119         });
120         this.selectionLayer.on('select', function(e){
121             if (!e.features) return;
122             if (pMap.selectionLayer.features() == undefined)
123                 pMap.selectionLayer.features(e.features);
124             else
125                 pMap.selectionLayer.features(pMap.selectionLayer.features().
126                 concat(e.features));
127             var tiles = pMap.selectionLayer.tiles();
128             for (var tile in tiles){
129                 var t = pMap.selectionLayer.cache.unload(tile, true);
130                 t.element.parentNode.removeChild(t.element);
131             }
132             pMap.map.dispatch({type:'move'});
133             window.postMessage("evt://featHighlighted/" + pMap.container + "/" +
134             String((new Date()).getTime()), "*");
135         });
136         this.selectionLayer.on('unselect', function(e){
137             if (!e.features) return;
138             var feats = pMap.selectionLayer.features(),
139                 ids = {};
140             for (var f in e.features) ids[e.features[f].id] = true;
141             feats = feats.filter(function(f){return !(f.id in ids);})
142             pMap.selectionLayer.features(feats);
143             var tiles = pMap.selectionLayer.tiles();
144             for (var tile in tiles){
```

```
145                 var t = pMap.selectionLayer.cache.unload(tile, true);
146                 t.element.parentNode.removeChild(t.element);
147             }
148             pMap.map.dispatch({type:'move'});
149         });
150 
151         this.map.add(po.compass().pan("none"));
152         this.add_controller();
153         this.add_legend();
154         this.add_prob_filter();
155         this.add_map_classifier();
156         this.set_colorscheme(this.colorscheme);
157 
158     },
159 
160     add_controller: function(){
161         var map_container = $('#' + this.container);
162         map_container.append('<div id="' + this.controller +
163         '" class="controller"></div>');
164         $('#' + this.controller).css("left", this.width);
165     },
166 
167     add_legend: function(){
168         var map_controller = $('#' + this.controller);
169         this.legend = this.container + '_legend';
170         map_controller.append('<div class="legend_button"><p>Legend</p><div id="
                ' +
171         this.legend + '" class="legend_content"></div></div>');
172     },
173 
174     set_colorscheme: function(cs){
175         this.colorscheme = cs;
176         this.map.container().setAttribute("class", this.colorscheme);
177         if (this.classes != null)
178             this.setLegend();
179     },
180 
181     add_prob_filter: function(){
182         var map_controller = $('#' + this.controller);
183         this.prob_filter = this.container + '_prob_filter';
184         var prob_filter = '<div class="prob_filter"><p>Probability Filter</p><
                select id="'
185         + this.prob_filter + '">';
186         prob_filter += '<option value="1">None</option>';
187         prob_filter += '<option value="0.1">0.1</option>';
188         prob_filter += '<option value="0.05">0.05</option>';
189         prob_filter += '<option value="0.01">0.01</option>';
190         prob_filter += '</select>';
191         map_controller.append(prob_filter);
192         var thismap = this;
193         $('#' + this.prob_filter).change(function(){
194             if (thismap.probabilities != null) {
195                 thismap.filterRateMap(null);
196                 return;
197             }
198             var qs = thismap.qs;
199             qs['s_method'] = 'Choynowski';
200             var t1 = (new Date()).getTime();
201             console.log(thismap.container + ":Starting to get probabilities:" +
                    t1);
202             eventLog.push([thismap.container, "StartToGetProb", t1]);
203             $.ajax(
204                 {url: proxy + '/web_esda/service_proxy/smoothing/',
205                  data: qs,
```

```
206                 dataType: 'jsonp',
207                 success: function(res, stat, xhr){
208                     var t2 = (new Date()).getTime();
209                     console.log(thismap.container + ":Ending to get
                            probabilities:" + t2);
210                     eventLog.push([thismap.container, "EndToGetProb", t2]);
211                     rateTimes[thismap.container].push(t2 - t1);
212                     thismap.filterRateMap(res, stat, xhr);
213                 }
214             }
215         );
216     });
217 },
218
219 filterRateMap: function(res, stat, xhr){
220     if (res != null) this.probabilities = res.data;
221     if (this.probabilities == null || this.rates == null) return;
222     this.prob_threshold = $('#' + this.prob_filter).val();
223     var t0 = (new Date()).getTime();
224     console.log(this.container + ":Starting to filter rates:" + t0);
225     eventLog.push([this.container, "StartToFilter", t0]);
226     if (this.classes == null) this.classifyRates();
227     var classids = this.get_classids();
228     var t1 = (new Date()).getTime();
229     console.log(this.container + ":Ending to filter rates:" + t1);
230     eventLog.push([this.container, "EndToFilter", t1]);
231     rateTimes[this.container].push(t1 - t0);
232     var pMap = this;
233     var t2 = (new Date()).getTime();
234     console.log(this.container + ":Starting to cache classes:" + t2);
235     eventLog.push([this.container, "StartToCacheClasses", t2]);
236     $.ajax({
237         url: proxy + '/tileservice/raster/service_proxy/cl',
238         data: {dat: classids.join(','), ts:tileset},
239         type: 'POST',
240         dataType: 'jsonp',
241         success: function(res, stat, xhr){
242             pMap.classification_id = res['clsf_id'];
243             var t3 = (new Date()).getTime();
244             console.log(pMap.container + ":Ending to cache classes:" + t3);
245             eventLog.push([pMap.container, "EndToCacheClasses", t3]);
246             rateTimes[pMap.container].push(t3 - t2);
247             var old_tiles = {};
248             var tiles = pMap.layer.cache.locks();
249             for(var tile in tiles){
250                 old_tiles[tile] = tiles[tile].element.href.baseVal;
251             }
252             var checkTileUpdated = function(oldInfo){
253                 var new_tiles = pMap.layer.cache.locks();
254                 var res = true;
255                 for (var tile in oldInfo){
256                     if (new_tiles[tile].element.href.baseVal == oldInfo[tile
                            ]){
257                         res = false;
258                         break;
259                     }
260                 }
261                 return res;
262             };
263             console.log(pMap.container + ":Starting to render:" + t3);
264             eventLog.push([pMap.container, "StartToRender", t3]);
265             pMap.layer.update_tile();
266             var update_complete = checkTileUpdated(old_tiles);
267             while (update_complete == false){
```

```
                update_complete = checkTileUpdated(old_tiles);
            }
            var t4 = (new Date()).getTime();
            console.log(pMap.container + ":Ending to render:" + t4);
            eventLog.push([pMap.container, "EndToRender", t4]);
            rateTimes[pMap.container].push(t4 - t3);
            window.postMessage("evt://filteredByProb/" + pMap.container + "/"
                +
            String(t4), "*");
        }
    });

},

add_map_classifier: function(){
    var map_controller = $('#' + this.controller);
    this.classification_tool = this.container + '_classification';
    map_controller.append('<div class="classification_tool" id="' +
    this.classification_tool + '"><p>Map Classification</p></div>');
    var classification_tool = $('#' + this.classification_tool);
    classification_tool.append($('#classification_selector').clone().
    attr('id', this.container + '_clsf').show());
    classification_tool.append('<br>');
    classification_tool.append($('#noclass_selector').clone().
    attr('id', this.container + '_nocls').show());
    classification_tool.append('<br>');
    classification_tool.append($('#colorscheme_selector').clone().
    attr('id', this.container + '_color').show());
    var thismap = this;
    $('#' + this.container + '_clsf').change(function(){
        var clsf = {method: $(this).val(), nocls: thismap.nocls,
        colorscheme: thismap.colorscheme};
        thismap.update_classification(clsf);
    });
    $('#' + this.container + '_nocls').change(function(){
        var clsf = {method: thismap.classification, nocls: $(this).val(),
        colorscheme: thismap.colorscheme};
        thismap.update_classification(clsf);
    });
    $('#' + this.container + '_color').change(function(){
        var clsf = {method: thismap.classification, nocls: thismap.nocls,
        colorscheme: $(this).val()};
        thismap.update_classification(clsf);
    });
},

update_classification: function(clsf){
    var cs_changed = false,
        cl_changed = false;
    if (this.colorscheme != clsf.colorscheme){
        this.set_colorscheme(clsf.colorscheme);
        cs_changed = true;
    }
    if (this.classification != clsf.method || this.nocls != clsf.nocls){
        this.classification = clsf.method;
        this.nocls = clsf.nocls;
        this.classifyRates();
        cl_changed = true;
    }
    this.renderRateMap(cl_changed, cs_changed);
},

classifyRates: function(){
    if (this.classification == 'Quantile')
```

```
331             this.classes = pv.Scale.mapquantile().intervals(this.nocls);
332         else if (this.classification == 'Percentile')
333             this.classes = pv.Scale.percentile().intervals();
334         else if (this.classification == 'Boxmap')
335             this.classes = pv.Scale.boxmap().intervals();
336         else if (this.classification == 'Equal_Interval')
337             this.classes = pv.Scale.equal_interval().intervals(this.nocls);
338 
339         this.classes.domain(pv.values(this.rates)).range();
340         this.setLegend();
341     },
342 
343     setLegend: function(){
344         var legend_content = $("#" + this.legend);
345         legend_content.empty();
346         var css = document.styleSheets[0].cssRules[1].styleSheet.cssRules;
347         var colors = {};
348         for (var i = 0; i < css.length; i++){
349             if (css[i].selectorText.toLowerCase().indexOf("." +
350             this.colorscheme.toLowerCase()) != -1){
351                 colors[css[i].selectorText.toLowerCase()] =
352                 css[i].style.cssText.substring(6);
353             }
354         }
355 
356         var breakpoints = this.classes.get_q(),
357             n = this.classes.get_n();
358         for (var i = 0; i < breakpoints.length - 1; i++){
359             var cls = "q" + i + "-" + n;
360             var color = colors["." + this.colorscheme.toLowerCase() + "␣." + cls
                ];
361             var classItem = '<span␣style="background:␣' + color
362             + ';">   </span> ';
363             classItem += breakpoints[i].toPrecision(3).toString() + '␣~␣' +
364             breakpoints[i+1].toPrecision(3).toString();
365             classItem += '<br>';
366             legend_content.append(classItem);
367         }
368     },
369 
370     get_classids: function(){
371         var classids = [];
372         for (var r in this.rates){
373             if (this.probabilities != null && this.probabilities[r] > this.
                prob_threshold)
374                 classids.push(0);
375             else {
376                 classids.push(this.classes(this.rates[r]));
377             }
378         }
379         return classids;
380     },
381 
382     renderRateMap: function(res, stat, xhr){
383         if (res != null && typeof res != "boolean") this.rates = res.data;
384         var t0 = (new Date()).getTime();
385         console.log(this.container + ":Starting␣to␣classfy␣rates:" + t0);
386         eventLog.push([this.container, "StartToClassify", t0]);
387         if (this.classes == null) this.classifyRates();
388         var classids = this.get_classids();
389         var t1 = (new Date()).getTime();
390         var classificationTime = t1 - t0;
391         console.log(this.container + ":Completed␣to␣classfy␣rates:" + t1);
392         eventLog.push([this.container, "EndToClassify", t1]);
```

```
393         console.log(this.container + ":Time␣for␣rate␣classification:" +
394         classificationTime);
395         var t2 = (new Date()).getTime();
396         eventLog.push([this.container, "StartToRender", t2]);
397         rateTimes[this.container].push(classificationTime);
398         var pMap = this;
399         var old_cl_id = pMap.classification_id,
400             old_cs_id = pMap.colorscheme_id,
401             isFirst = (arguments.length == 3);
402         var old_tiles = {};
403         var tiles = pMap.layer.cache.locks();
404         for(var tile in tiles){
405             old_tiles[tile] = tiles[tile].element.href.baseVal;
406         }
407         var checkTileUpdated = function(oldInfo){
408             var new_tiles = pMap.layer.cache.locks();
409             var res = true;
410             for (var tile in oldInfo){
411                 if (new_tiles[tile].element.href.baseVal == oldInfo[tile]){
412                     res = false;
413                     break;
414                 }
415             }
416             return res;
417         };
418         var rerenderMap = function(newMap, checkTarget){
419             var evtType = newMap? 'layerRendered': 'classificationUpdated';
420             var update = false;
421             if (!newMap && (pMap.classification == 'Quantile' ||
422             pMap.classification == 'Equal␣Interval')) update = true;
423             else if (checkTarget == 'cs' && pMap.colorscheme_id != old_cs_id)
424             update = true;
425             else if (checkTarget == 'cl' && pMap.classification_id != old_cl_id)
426             update = true;
427             if (update){
428                 pMap.layer.update_tile();
429                 var update_complete = checkTileUpdated(old_tiles);
430                 while (update_complete == false){
431                     update_complete = checkTileUpdated(old_tiles);
432                 }
433                 var t3 = (new Date()).getTime();
434                 window.postMessage("evt://" + evtType + "/" + pMap.container + "
                        /" +
435                 String(t3), "*");
436                 var renderTime = t3 - t2;
437                 console.log(pMap.container + ':Time␣elapsed␣since␣classification
                        ␣is␣done:'
438                 + renderTime);
439                 eventLog.push([pMap.container, "EndToRender", t3]);
440                 rateTimes[pMap.container].push(renderTime);
441             }
442         };
443         if (isFirst || arguments[0]){
444             $.ajax({
445                 url: proxy + '/tileservice/raster/service_proxy/cl',
446                 data: {dat: classids.join(','), ts:tileset},
447                 type: 'POST',
448                 dataType: 'jsonp',
449                 success: function(res, stat, xhr){
450                     pMap.classification_id = res['clsf_id'];
451                     rerenderMap(isFirst, 'cs');
452                 }
453             });
454         }
```

```
455         if (isFirst || pMap.classification == 'Percentile' || pMap.
                classification ==
456         'Boxmap'){
457             var nocls = this.classes.get_n(),
458                 css = document.styleSheets[0].cssRules[1].styleSheet.cssRules,
459                 colors = [];
460             for (var i = 0; i < css.length; i++){
461                 for (var j = 0; j < nocls; j++){
462                     var style_cls = "." + this.colorscheme.toLowerCase() + ' .q'
                            + j + '-'
463                     + nocls;
464                     if (css[i].selectorText.toLowerCase().indexOf(style_cls) !=
                            -1){
465                         var c = css[i].style.getPropertyValue("fill");
466                         if (c[0] != '#') c = rgb2hex(c);
467                         colors.push(c);
468                     }
469                 }
470             }
471             $.ajax({
472                 url: proxy + '/tileservice/raster/service_proxy/cs',
473                 data: {colors: colors.join(',')},
474                 dataType: 'jsonp',
475                 success: function(res, stat, xhr){
476                     pMap.colorscheme_id = res['cs_id'];
477                     rerenderMap(isFirst, 'cl');
478                 }
479             });
480         }
481     },
482
483     getRates: function(data, evt, pop, sm_method, wgt){
484         var qs = this.qs = {
485             'service': 'smoothing',
486             'data': data,
487             'e': evt,
488             'b': pop,
489             's_method': sm_method
490         };
491         if (sm_method == 'Spatial Empirical Bayes' || sm_method == 'Spatial Rate
                ' ||
492         sm_method == 'Locally Weighted Average')
493             qs['w'] = this.qs['w'] = wgt;
494         var pMap = this;
495         var t1 = (new Date()).getTime();
496         rateStartTimes[qs.s_method].push(t1);
497         console.log(this.container + ':Starting the acquisition of rates:' +  t1
                );
498         eventLog.push([this.container, "StartToGetRate", t1]);
499         $.ajax(
500             {url: proxy + '/web_esda/service_proxy/smoothing/',
501              data: qs,
502              dataType: 'jsonp',
503              success: function(res, stat, xhr){
504                  var t2 = (new Date()).getTime();
505                  console.log(pMap.container + ':Completing the acquisition of
                         rates:'
506                  + t2);
507                  eventLog.push([pMap.container, "EndToGetRate", t2]);
508                  var ratetime = t2 - t1;
509                  console.log(pMap.container + ':Getting rates:' +  ratetime);
510                  rateTimes[pMap.container].push(ratetime);
511                  pMap.renderRateMap(res, stat, xhr);
512              }
```

```
            }
        );
    },

    startGettingRates: function(){
        this.getRates(this.data, this.evt, this.pop, this.sm_method, this.wgt);
    },

    init: function(data, evt, pop, sm_method, wgt){
        this.initTime = (new Date()).getTime();
        console.log(this.container + ": Starting to draw a map:" + this.initTime)
            ;
        eventLog.push([this.container, "StartMapDraw", this.initTime]);
        this.draw_map();
        this.data = data;
        this.evt = evt;
        this.pop = pop;
        this.sm_method = sm_method;
        this.wgt = wgt;
    }

}
```

A.2.3 Event handling

```
function MessageEventHandler(){
    this.listeners = {};
    var that = this;
    window.addEventListener("message",function(evt){
            if (evt.data.slice(0,6)=="evt://")
                that.onMessage(evt);
        },false);
}
MessageEventHandler.prototype.addEventListener = function(evtName,callback){
    if (evtName in this.listeners){
        this.listeners[evtName].push(callback);
    } else {
        this.listeners[evtName] = [callback];
    }
}
MessageEventHandler.prototype.removeEventListener = function(evtName){
    if (evtName in this.listeners)
        delete this.listeners[evtName];
}
MessageEventHandler.prototype.onMessage = function(evt){
    var evtData = evt.data.slice(6);
    var evtName = evtData.split('/')[0];
    if (evtName in this.listeners){
        for (var i=0; i<this.listeners[evtName].length; i++){
            this.listeners[evtName][i](evtData);
        }
    }
}

var po = org.polymaps;
sessionStorage.clear();

po.select = function(){
    var select = {},
        layers = [];

    function dispatchSelectEvt(select_evt, feat, tile){
        var evt = {type:'select', features: feat, tile: tile};
        if (!select_evt) evt.type = 'unselect';
```

```
        for (var i = 0; i < layers.length; i++)
            layers[i].dispatch(evt);
    };

    function handleSelection(e){
        var feat = sessionStorage.getItem(e.bbox);
        if (feat) return dispatchSelectEvt(e.select, eval(feat));
        $.ajax({
            url: proxy + '/web_esda/userdata/features/',
            data: {layers: e.layers, bbox:e.bbox},
            dataType: 'json',
            success: function(res, stat, xhr){
                sessionStorage.setItem(e.bbox, JSON.stringify(res.features));
                dispatchSelectEvt(e.select, res.features, e.tile);
            }
        });
    };

    select.add = function(lyr){
        layers.push(lyr);
    };

    select.getSelectedFeature = function(bbox){
        return sessionStorage.getItem(bbox);
    };

    select.removeSelectedFeature = function(bbox){
        return sessionStorage.removeItem(bbox);
    };

    select.clearSelectedFeatures = function(){
        sessionStorage.clear();
    };

    select.getFeatureCount = function(){
        return sessionStorage.length;
    };

    select.dispatch = po.dispatch(select);
    select.on("click", handleSelection);

    return select;
}
```

A.3 Components for vector-based test application

A.3.1 Test tool

```
var rate_maps = {};
var myEventObserver;
var testResults = [];
var rateTimes = {"Raw_Map":[],"Spatial_Rate_Map":[]};
var rateStartTimes = {'Raw':[], 'Spatial Rate':[]};
var rateEndTimes = {'Raw':[], 'Spatial Rate':[]};
var rateTypes = [];
var noPans = 0;
var renderedMaps = [];
var tileset = '';
var eventLog = [];
var currentTestOp;
var eventHeaders = {
    0:["StartToDisplayBaseTiles","EndToDisplayBaseTiles","StartToGetRates",
    "EndToGetRates","StartToClassify","EndToClassify","StartToRender","
        EndToRender"],
```

```
16 };
17 var timeHeaders = {
18     0:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles"],
19     3:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","GetProbs",
20     "FilterRates","RerenderTiles"],
21     4:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","ZoomIn",
22     "ZoomIn","ZoomIn"],
23     6:["DisplayBaseTiles","GetRates","ClassifyRates","RenderTiles","Pan","Pan","
           Pan"],
24 };
25 var preZoom;
26
27 //
28 // test operations
29 //
30 function drawRateMap(map, data, map_id){
31    var sm_method = {"Raw_Map":"Raw", "Spatial_Rate_Map":"Spatial Rate"};
32    map.init(data.data, data.evt, data.pop, sm_method[map_id], data.wgt);
33 }
34
35 function changeClassification(map){
36    map.initTime = (new Date()).getTime();
37    var classifier = $('#' + map.container + '_clsf');
38    classifier.val("Percentile");
39    classifier.change();
40 }
41
42 function filterByProb(map){
43    map.initTime = (new Date()).getTime();
44    var probFilter = $('#' + map.container + '_prob_filter');
45    probFilter.val("0.05");
46    probFilter.change();
47 }
48
49 function zoomIn3x(map){
50    map.initTime = (new Date()).getTime();
51    eventLog.push(["Raw_Map", "Starting to zoom in", map.initTime]);
52    map.map.zoomBy(1);
53 }
54
55 function zoomOut3x(map){
56    map.initTime = (new Date()).getTime();
57    map.map.zoomBy(-1);
58 }
59
60 function pan3x(map){
61    map.initTime = (new Date()).getTime();
62    eventLog.push(["Raw_Map", "Starting to pan", map.initTime]);
63    map.map.panBy({x:256, y:256});
64    noPans = 1;
65 }
66
67 function centroid(coords){
68    var area = 0.0,
69        cx = 0.0,
70        cy = 0.0;
71    for (var i = 0; i < (coords.length - 1); i++){
72        var p1 = coords[i],
73            p2 = coords[i+1];
74        var areaInc = (p1[0]*p2[1] - p2[0]*p1[1]);
75        area += areaInc;
76        cx += (p1[0] + p2[0])*areaInc;
77        cy += (p1[1] + p2[1])*areaInc;
78    }
```

```
79      cx = cx/(3*area);
80      cy = cy/(3*area);
81      return {lon:cx, lat:cy};
82  }
83
84  function getTargetFeature(tiles){
85      var tileKeys = [];
86      for (var tile in tiles) tileKeys.push(tile);
87      var targetTile = null;
88      while (targetTile == null){
89          var aTile = tiles[tileKeys[Math.floor(tileKeys.length*Math.random())]];
90          if (aTile && aTile.length > 0) targetTile = aTile;
91      }
92      var targetFeat = null;
93      while (targetFeat == null){
94          var aFeat = targetTile[Math.floor(targetTile.length*Math.random())];
95          if (aFeat && aFeat.element) targetFeat = aFeat;
96      }
97      return targetFeat;
98  }
99
100 function highlightPolyInOneMap(map){
101     var targetFeat = getTargetFeature(map.layer.tiles());
102     var targetFeatLoc = centroid(targetFeat.data.geometry.coordinates[0]);
103     var targetFeatPoint = map.map.locationPoint(targetFeatLoc);
104     map.initTime = (new Date()).getTime();
105     $(targetFeat.element).trigger('click', [targetFeatPoint]);
106 }
107
108 function linkMaps(maps){
109     var targetFeat = getTargetFeature(maps["Raw_Map"].layer.tiles());
110     var targetFeatLoc = centroid(targetFeat.data.geometry.coordinates[0]);
111     var targetFeatPoint = maps["Raw_Map"].map.locationPoint(targetFeatLoc);
112     maps["Spatial_Rate_Map"].initTime = (new Date()).getTime();
113     $(targetFeat.element).trigger('click', [targetFeatPoint]);
114 };
115
116 //
117 // utility functions
118 //
119 function debug(logTxt){
120     if (window.console != undefined) {
121         console.log(logTxt);
122     } else if (opera) {
123         opera.postError(logTxt);
124     }
125 }
126
127 function avg(results) {
128     var sum = 0;
129     for (i = 0; i < results.length; i++) {
130         sum += results[i];
131     }
132     return sum/results.length;
133 }
134
135 function sleep(milliseconds){
136     var start = new Date().getTime();
137     while (new Date().getTime() < start + milliseconds);
138 }
139
140 function destroyMap(map){
141     var cont = $("#" + map.container);
142     if (cont) {
```

```
143         cont.prev().remove();
144         cont.next().remove();
145         cont.remove();
146     }
147     if (map.container in rate_maps){
148         map.layer.cache.size(0);
149         delete rate_maps[map.container];
150     }
151     delete map.container;
152 }
153
154 function resetEvtObserver(){
155     myEventObserver = null;
156 }
157
158 //
159 // evaluation functions
160 //
161 function startTest(){
162     debug('test started');
163     var testOp = getTestOp();
164     currentTestOp = testOp;
165     var data = getTestData();
166     var mapSize = getMapSize();
167     var noTests = getNoTests();
168     setupAndRunTest(testOp, mapSize, data, noTests);
169 }
170
171 function setupAndRunTest(testOp, mapSize, data, noTests){
172     testResults = [];
173     prepareTest(testOp, mapSize, data, noTests);
174     prepareTestRun(testOp, mapSize, data);
175     if (testOp < 2)
176         runTest(testOp, data);
177 }
178
179 function addResult0(edata){
180     testResults.push(parseFloat(edata[2]) - rate_maps["Raw_Map"].initTime);
181     console.log('time diff:' + rate_maps["Raw_Map"].initTime + "-" + edata[2] +
182     "=" + testResults[testResults.length -1]);
183 }
184
185 function addResult1(edata){
186     addResult0(edata);
187 }
188
189 function addResult2(edata){
190     testResults.push(parseFloat(edata[2]) - rate_maps["Spatial_Rate_Map"].
        initTime);
191     console.log('time diff:' + rate_maps["Spatial_Rate_Map"].initTime + "-" +
        edata[2]
192     + "=" + testResults[testResults.length -1]);
193 }
194
195 function nextTest0(noTests, testOp, mapSize, data, srcMap){
196     if (testResults.length < noTests){
197         window.setTimeout(function(){
198             prepareTestRun(testOp, mapSize, data);
199             runTest(testOp, data);
200             }, 1);
201     }
202     else if (testResults.length == noTests && currentTestOp == 1) {
203         debug("Avg time: " + (avg(testResults) + 1.0)  + " ms");
204         var eHeader = eventHeaders[currentTestOp - 1];
```

```
205         var tHeader = timeHeaders[currentTestOp - 1];
206         var eHeaderLen = eHeader.length*2;
207         var tHeaderLen = tHeader.length;
208         debug("Response times");
209         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
210         for (var i = 0; i < testResults.length; i++) {
211             debug("Test" + i);
212             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
213             for (var j = 0; j < l.length; j++){
214                 debug(l[j].join(","));
215             }
216         }
217         debug("Raw_Map");
218         debug(tHeader.join(","));
219         for (var i = 0; i < testResults.length; i++) {
220             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
221             debug(l.join(","));
222         }
223         debug("Spatial_Rate_Map");
224         debug(tHeader.join(","));
225         for (var i = 0; i < testResults.length; i++) {
226             var l = rateTimes["Spatial_Rate_Map"].slice(i*tHeaderLen, (i+1)*
                    tHeaderLen);
227             debug(l.join(","));
228         }
229     }
230     else if (testResults.length == noTests) {
231         debug("Avg time: " + (avg(testResults) + 1.0)  + " ms");
232         var eHeader = eventHeaders[currentTestOp];
233         var tHeader = timeHeaders[currentTestOp];
234         var eHeaderLen = eHeader.length;
235         var tHeaderLen = tHeader.length;
236         debug("Response times");
237         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
238         for (var i = 0; i < testResults.length; i++) {
239             debug("Test" + i);
240             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
241             for (var j = 0; j < l.length; j++){
242                 debug(l[j].join(","));
243             }
244         }
245         debug(tHeader.join(","));
246         for (var i = 0; i < testResults.length; i++) {
247             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
248             debug(l.join(","));
249         }
250     }
251 }
252
253 function nextTest1(noTests, testOp, mapSize, data, srcMap){
254     nextTest0(noTests, testOp, mapSize, data, srcMap);
255 }
256
257 function nextTest2(noTests, testOp, mapSize, data, srcMap){
258     if (testResults.length < noTests){
259         window.setTimeout(function(){
260             prepareTestRun(testOp, mapSize, data);
261             }, 1);
262     }
263     else if (testResults.length == noTests && currentTestOp == 3) {
264         debug("Avg time: " + (avg(testResults) + 1.0)  + " ms");
265         var tHeader = timeHeaders[currentTestOp];
266         var eHeaderLen = 15;
267         var tHeaderLen = tHeader.length;
```

```
268         debug("Response times");
269         for (var i = 0; i < testResults.length; i++) debug(testResults[i]);
270         for (var i = 0; i < testResults.length; i++) {
271             debug("Test" + i);
272             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
273             for (var j = 0; j < l.length; j++){
274                 debug(l[j].join(","));
275             }
276         }
277         debug(tHeader.join(","));
278         for (var i = 0; i < testResults.length; i++) {
279             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
280             debug(l.join(","));
281         }
282     }
283     else if (testResults.length == noTests) {
284         var addition = 1;
285         if (testOp == 4 || testOp == 5 || testOp == 6) addition = -1.0;
286         else if (testOp == 7 || testOp == 8) addition = 0;
287         debug("Avg time: " + (avg(testResults) + addition)  + " ms");
288         debug("Response times");
289         for (var i = 0; i < testResults.length; i++) debug(testResults[i] +
                addition);
290         var tHeader = timeHeaders[currentTestOp];
291         var eHeaderLen = 13;
292         var tHeaderLen = tHeader.length;
293         for (var i = 0; i < testResults.length; i++) {
294             debug("Test" + i);
295             var l = eventLog.slice(i*eHeaderLen, (i+1)*eHeaderLen);
296             for (var j = 0; j < l.length; j++){
297                 debug(l[j].join(","));
298             }
299         }
300         debug(tHeader.join(","));
301         for (var i = 0; i < testResults.length; i++) {
302             var l = rateTimes["Raw_Map"].slice(i*tHeaderLen, (i+1)*tHeaderLen);
303             debug(l.join(","));
304         }
305     }
306 }
307
308 function prepareTest(testOp, mapSize, data, noTests){
309     var addResult = {0: addResult0, 1: addResult1};
310     var runNextTest = {0: nextTest0, 1: nextTest1, 2: nextTest2};
311     var runTestOp2_3 = function(evtData){
312         console.log(evtData);
313         var edata = evtData.split('/');
314         addResult0(edata);
315         nextTest2(noTests, testOp, mapSize, data);
316     };
317     var runTestOp4 = function(evtData){
318         console.log(evtData);
319         var edata = evtData.split('/');
320         addResult2(edata);
321         nextTest2(noTests, testOp, mapSize, data);
322     };
323     if (myEventObserver == null) myEventObserver = new MessageEventHandler();
324     if (testOp == 0) {
325         myEventObserver.addEventListener("layerRendered", function(evtData){
326             console.log(evtData);
327             var edata = evtData.split('/');
328             addResult[testOp](edata);
329             runNextTest[testOp](noTests, testOp, mapSize, data, edata[1]);
330         });
```

```
331         myEventObserver.addEventListener("loadEnd", function(evtData){
332             console.log(evtData);
333             var edata = evtData.split('/');
334             var map_type = edata[1];
335             var loadEndTime = parseInt(edata[2]);
336             eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
337             base_tile_display = loadEndTime - rate_maps[edata[1]].initTime;
338             rateTimes[edata[1]].push(base_tile_display);
339             console.log(edata[1] + ':base tile display (t1-t0):' +
                    base_tile_display);
340             rate_maps["Raw_Map"].startGettingRates();
341         });
342     }
343     else if (testOp == 1) {
344         myEventObserver.addEventListener("layerRendered", function(evtData){
345             console.log(evtData);
346             var edata = evtData.split('/');
347             renderedMaps.push(edata[1]);
348             if (renderedMaps.length == 2){
349                 addResult[testOp](edata);
350                 runNextTest[testOp](noTests, testOp, mapSize, data, edata[1]);
351             }
352         });
353         myEventObserver.addEventListener("loadEnd", function(evtData){
354             console.log(evtData);
355             var edata = evtData.split('/');
356             var map_type = edata[1];
357             var loadEndTime = parseInt(edata[2]);
358             eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
359             base_tile_display = loadEndTime - rate_maps[edata[1]].initTime;
360             rateTimes[edata[1]].push(base_tile_display);
361             console.log(edata[1] + ':base tile display (t1-t0):' +
                    base_tile_display);
362             rate_maps[edata[1]].startGettingRates();
363         });
364     }
365     else if (testOp >= 2 && testOp <= 8){
366         if (testOp != 4 && testOp != 5)
367             myEventObserver.addEventListener("loadEnd", function(evtData){
368                 console.log(evtData);
369                 var edata = evtData.split('/');
370                 var loadEndTime = parseInt(edata[2]);
371                 eventLog.push([edata[1], "EndBaseTileDisplay", loadEndTime])
372                 base_tile_display = loadEndTime - rate_maps[edata[1]].initTime;
373                 rateTimes[edata[1]].push(base_tile_display);
374                 console.log('base tile display (t1-t0):' + base_tile_display);
375                 rate_maps[edata[1]].startGettingRates();
376             });
377         myEventObserver.addEventListener("readyForTest", function(evtData){
378             console.log(evtData);
379             var edata = evtData.split('/');
380             runTest(parseInt(edata[1]), data);
381         });
382         if (testOp == 2 || testOp == 3 || testOp == 7 || testOp == 8){
383             if (testOp != 8){
384                 myEventObserver.addEventListener("layerRendered", function(
                        evtData){
385                     console.log(evtData);
386                     window.postMessage("evt://readyForTest/" + String(testOp), "
                            *");
387                 });
388             }
389             else
```

```
390                 myEventObserver.addEventListener("layerRendered", function(
                        evtData){
391                     console.log(evtData);
392                     var edata = evtData.split('/');
393                     renderedMaps.push(edata[1]);
394                     if (renderedMaps.length == 2)
395                         window.postMessage("evt://readyForTest/" + String(testOp
                                ), "*");
396                 });
397             if (testOp == 2)
398                 myEventObserver.addEventListener("classificationUpdated",
                        runTestOp2_3);
399             else if (testOp == 3)
400                 myEventObserver.addEventListener("filteredByProb", runTestOp2_3)
                        ;
401             else if (testOp == 7)
402                 myEventObserver.addEventListener("featHighlighted", runTestOp2_3
                        );
403             else if (testOp == 8)
404                 myEventObserver.addEventListener("featHighlighted", function(
                        evtData){
405                     console.log(evtData);
406                     var edata = evtData.split('/');
407                     if (edata[1] == "Spatial_Rate_Map")
408                         runTestOp4(evtData);
409                 });
410         }
411         else if (testOp == 4 || testOp == 5){
412             var zoomInterval = testOp == 5 ? -1: 1;
413             var evtName = testOp == 5? 'zoomOut3x': 'zoomIn3x';
414             var goalZ = testOp == 5? 4 : 7;
415             myEventObserver.addEventListener("layerRendered", function(evtData){
416                 console.log(evtData);
417                 console.log("zoom␣level:" + rate_maps["Raw_Map"].map.zoom());
418                 preZoom = (new Date()).getTime();
419                 myEventObserver.addEventListener("loadEnd", function(evtData){
420                     console.log(evtData);
421                     var edata = evtData.split('/');
422                     if (edata.length != 4) return;
423                     var curZoom = parseInt(edata[2]);
424                     eventLog.push(["Raw_Map", "Ending␣to␣zoom␣in", curZoom]);
425                     if (rate_maps["Raw_Map"].map.zoom() == 5)
426                         rateTimes["Raw_Map"].push(curZoom - preZoom);
427                     else
428                         rateTimes["Raw_Map"].push(curZoom - preZoom - 1);
429                     preZoom = curZoom;
430                     console.log("zoom␣level:" + rate_maps["Raw_Map"].map.zoom())
                            ;
431                     var z = rate_maps["Raw_Map"].map.zoom();
432                     if (z != goalZ) window.setTimeout(function(){
433                         rate_maps["Raw_Map"].map.rendered = {};
434                         eventLog.push(["Raw_Map", "Starting␣to␣zoom␣in",
435                         (new Date()).getTime()]);
436                         rate_maps["Raw_Map"].map.zoomBy(zoomInterval);
437                         }, 1);
438                     else {
439                         var edata = evtData.split('/');
440                         window.postMessage("evt://" + evtName + "/Raw_Map/" +
441                         edata[2], "*");
442                     }
443                 });
444                 runTest(testOp, data);
445             });
446             myEventObserver.addEventListener(evtName, runTestOp2_3);
```

```
447         }
448         else if (testOp == 6){
449             myEventObserver.addEventListener("layerRendered", function(evtData){
450                 console.log(evtData);
451                 preZoom = (new Date()).getTime();
452                 myEventObserver.addEventListener("loadEnd", function(evtData){
453                     console.log(evtData);
454                     var edata = evtData.split('/');
455                     if (edata.length != 4) return;
456                     var curZoom = parseInt(edata[2]);
457                     eventLog.push(["Raw_Map", "Ending to pan", curZoom]);
458                     if (noPans == 0)
459                         rateTimes["Raw_Map"].push(curZoom - preZoom);
460                     else
461                         rateTimes["Raw_Map"].push(curZoom - preZoom - 1);
462                     preZoom = curZoom;
463                     if (noPans != 3) window.setTimeout(function(){
464                         rate_maps["Raw_Map"].map.rendered = {};
465                         eventLog.push(["Raw_Map", "Starting to pan",
466                         (new Date()).getTime()]);
467                         rate_maps["Raw_Map"].map.panBy({x:256, y:256});
468                         noPans += 1;
469                         }, 1);
470                     else {
471                         var edata = evtData.split('/');
472                         window.postMessage("evt://pan3x/Raw_Map/" + edata[2], "*
                                ");
473                     }
474                 });
475                 runTest(testOp, data);
476             });
477             myEventObserver.addEventListener("pan3x", runTestOp2_3);
478         }
479     }
480 }
481
482 function prepareTestRun(testOp, mapSize, data){
483     var origTestOp = testOp;
484     if (testOp >= 2 && testOp <= 7) testOp = 0;
485     else if (testOp == 8) testOp = 1;
486     var z = (origTestOp == 5 || origTestOp == 6)? 7: 4;
487     if (testOp == 0 || testOp == 1) {
488         var map_ids = ["Raw_Map"];
489         if (testOp == 1) map_ids.push("Spatial_Rate_Map");
490         for (var i = 0; i < map_ids.length; i++){
491             var map_id = map_ids[i];
492             if (map_id in rate_maps) destroyMap(rate_maps[map_id]);
493             var clsf = {
494                 method: 'Quantile',
495                 nocls: 5,
496                 colorscheme: 'Blues'
497             };
498             rate_maps[map_id] = new RateMap(map_id, mapSize, null, z, null, clsf
                    );
499         }
500         if (testOp == 1) renderedMaps = [];
501     }
502     if (origTestOp >= 2 && origTestOp <= 7){
503         if (origTestOp == 4 || origTestOp == 5 || origTestOp == 6)
504         myEventObserver.removeEventListener("loadEnd");
505         drawRateMap(rate_maps["Raw_Map"], data, "Raw_Map");
506         if (origTestOp == 4 || origTestOp == 6){
507             var t = (new Date()).getTime();
508             rateTimes["Raw_Map"].push(t - rate_maps["Raw_Map"].initTime);
```

```
509             rate_maps["Raw_Map"].startGettingRates();
510         }
511     }
512     else if (origTestOp == 8){
513        renderedMaps = [];
514        for (var map_id in rate_maps)
515            drawRateMap(rate_maps[map_id], data, map_id);
516     }
517 }
518
519 function runTest(testOp, data){
520     console.log('runTest' + testOp);
521     var func = getTestFunction(testOp);
522     if (testOp == 0 || testOp == 1) {
523         var map_ids = ["Raw_Map"];
524         if (testOp == 1) map_ids.push("Spatial_Rate_Map");
525         for (var i = 0; i < map_ids.length; i++){
526             var map_id = map_ids[i];
527             func(rate_maps[map_id], data, map_id);
528         }
529         return true;
530     }
531     else if (testOp >= 2 && testOp <= 7){
532         console.log('test operation ' + testOp + ' starts ..');
533         func(rate_maps["Raw_Map"], data, "Raw_Map");
534     }
535     else if (testOp == 8){
536         console.log('test operation ' + testOp + ' starts ..');
537         func(rate_maps);
538     }
539 }
540
541 function getTestOp(){
542     return document.forms["options"].test_operation.selectedIndex;
543 }
544
545 function getTestFunction(op){
546     if (op == 0 || op == 1) return drawRateMap;
547     else if (op == 2) return changeClassification;
548     else if (op == 3) return filterByProb;
549     else if (op == 4) return zoomIn3x;
550     else if (op == 5) return zoomOut3x;
551     else if (op == 6) return pan3x;
552     else if (op == 7) return highlightPolyInOneMap;
553     else if (op == 8) return linkMaps;
554 }
555
556 function getTestData(){
557     var mapdata = document.forms["options"].data.selectedIndex;
558     if (mapdata == 0){
559         tileset = 'counties';
560         return {data: 'unemployment_2009_main_', evt: 'UNEMP', pop: 'POP1',
561         wgt: 'unemployment_2009_main.gal'};
562     }
563     else if (mapdata == 1){
564         tileset = 'pcsa';
565         return {data: 'pcsa_mainland_', evt: 'SUM_Sum_AG', pop: 'SUM_Sum_PO',
566         wgt: 'pcsa_mainland_q1.gal'};
567     }
568     else if (mapdata == 2){
569         tileset = 'sd';
570         return {data: 'sd_from_tracts_', evt: 'SUM_AGE_65', pop: 'SUM_POP200',
571         wgt: 'sd_from_tracts_q1.gal'};
572     }
```

```
    else if (mapdata == 3){
        tileset = 'zip';
        return {data: 'usps_zip_', evt: 'Over_65', pop: 'Sum_POP200',
        wgt: 'zip_mainland_q1.gal'};
    }
    else if (mapdata == 4){
        tileset = 'tracts';
        return {data: 'tracts_', evt: 'AGE_65_UP', pop: 'POP2000',
        wgt: 'tracts_q1.gal'};
    }
    return null;
}

function getMapSize(){
    var res = {width: null, height: null};
    var mapsize_inx = document.forms["options"].mapsize.selectedIndex;
    if (mapsize_inx == 0) {
        res.width = 400;
        res.height = 200;
    } else if (mapsize_inx == 1){
        res.width = 800;
        res.height = 400;
    } else if (mapsize_inx == 2){
        res.width = 1200;
        res.height = 600;
    } else {
        res.width = 1600;
        res.height = 800;
    }
    return res;
}

function getNoTests(){
    return parseInt(document.forms["options"].no_tests.value);
}
```

A.3.2 Map component

```
var po = org.polymaps;

function RateMap(container, size, center, zoom, zoomRange, classification){
    this.container = container;
    this.controller = this.container + '_controller';
    this.map = null;
    this.width = 800;
    this.height = 400;
    this.mapClass = "map_800_400";
    if (size != null){
        this.width = size.width;
        this.height = size.height;
        this.mapClass = "map_" + String(this.width) + "_" + String(this.height);
    }
    if ($("#" + container).length == 0){
        $("body").append('<br><div id="' + container + '" class="' + this.
            mapClass
        + '"></div><br>');
    }
    this.center = {lat: 39, lon: -96};
    if (center != null) this.center = center;
    this.zoom = 4;
    if (zoom != null) this.zoom = zoom;
    this.zoomRange = [3,7];
    if (zoomRange != null) this.zoomRange = zoomRange;
    this.classification = 'Quantile';
```

```
26      if ('method' in classification)
27          this.classification = classification.method;
28      this.nocls = 5;
29      if ('nocls' in classification)
30          this.nocls = classification.nocls;
31      this.colorscheme = "Blues";
32      if ('colorscheme' in classification)
33          this.colorscheme = classification.colorscheme;
34      this.qs = null;
35      this.layer = null;
36      this.rates = null;
37      this.probabilities = null;
38      this.tiles = null;
39      this.selections = {};
40      this.filtered = {};
41      this.classes = null;
42      this.legend = null;
43      this.initTime = null;
44      this.rendered = {};
45      this.classids = {};
46  }
47
48  RateMap.prototype = {
49
50      draw_map: function(){
51          if (this.map != null) return;
52
53          this.map = po.map()
54              .size({x:this.width, y:this.height})
55              .container(document.getElementById(this.container).
56              appendChild(po.svg("svg")))
57              .center(this.center)
58              .zoom(this.zoom)
59              .zoomRange(this.zoomRange)
60              .add(po.interact());
61
62          var pMap = this;
63          this.layer = po.geoJson()
64                .url("http://129.219.93.206/tileservice/vector/" + tileset + "/{Z
                      }/{X}/{Y}.json")
65                .on("load", function(t){
66                          pMap.rendered[t.tile.key] = true;
67                          var all_rendered = true;
68                          for(var lock in pMap.layer.cache.locks()){
69                              if (!(lock in pMap.rendered)){
70                                  all_rendered = false;
71                                  break;
72                              }
73                          }
74                          if (all_rendered){
75                              window.postMessage("evt://loadEnd/" + pMap.
                                    container + "/"
76                              + String((new Date()).getTime()), "*");
77                          }
78                      })
79                .id("county");
80
81          this.map.add(this.layer);
82          this.map.add(po.compass().pan("none"));
83          this.add_controller();
84          this.add_legend();
85          this.add_prob_filter();
86          this.add_map_classifier();
87          this.set_colorscheme(this.colorscheme);
```

```
88      },
89
90      add_controller: function(){
91          var map_container = $("#" + this.container);
92          map_container.append('<div id="' + this.controller + '" class="
                controller"></div>');
93          $('#' + this.controller).css("left", this.width);
94      },
95
96      add_prob_filter: function(){
97          var map_controller = $("#" + this.controller);
98          this.prob_filter = this.container + '_prob_filter';
99          var prob_filter =
100         '<div class="prob_filter"><p>Probability Filter</p><select id="' +
101         this.prob_filter + '">';
102         prob_filter += '<option value="1">None</option>';
103         prob_filter += '<option value="0.1">0.1</option>';
104         prob_filter += '<option value="0.05">0.05</option>';
105         prob_filter += '<option value="0.01">0.01</option>';
106         prob_filter += '</select>';
107         map_controller.append(prob_filter);
108         var thismap = this;
109         $('#' + this.prob_filter).change(function(){
110             if (thismap.probabilities != null) {
111                 thismap.filterRateMap(null);
112                 return;
113             }
114             var qs = thismap.qs;
115             qs['s_method'] = 'Choynowski';
116             var t1 = (new Date()).getTime();
117             console.log(thismap.container + ":Starting to get probabilities:" +
                    t1);
118             eventLog.push([thismap.container, "StartToGetProb", t1]);
119             $.ajax(
120                 {url:'http://129.219.93.206/web_esda/service_proxy/smoothing/',
121                  data: qs,
122                  dataType: 'jsonp',
123                  success: function(res, stat, xhr){
124                      var t2 = (new Date()).getTime();
125                      console.log(thismap.container + ":Ending to get
                            probabilities:" + t2);
126                      eventLog.push([thismap.container, "EndToGetProb", t2]);
127                      rateTimes[thismap.container].push(t2 - t1);
128                      thismap.filterRateMap(res, stat, xhr);
129                  }
130                 }
131             );
132         });
133     },
134
135     add_map_classifier: function(){
136         var map_controller = $('#' + this.controller);
137         this.classification_tool = this.container + '_classification';
138         map_controller.append('<div class="classification_tool" id="' +
139         this.classification_tool + '"><p>Map Classification</p></div>');
140         var classification_tool = $('#' + this.classification_tool);
141         classification_tool.append($('#classification_selector').clone().
142         attr('id', this.container + '_clsf').show());
143         classification_tool.append('<br>');
144         classification_tool.append($('#noclass_selector').clone().
145         attr('id', this.container + '_nocls').show());
146         classification_tool.append('<br>');
147         classification_tool.append($('#colorscheme_selector').clone().
148         attr('id', this.container + '_color').show());
```

```
149         var thismap = this;
150         $('#' + this.container + '_clsf').change(function(){
151             var clsf = {method: $(this).val(), nocls: thismap.nocls,
152             colorscheme: thismap.colorscheme};
153             thismap.update_classification(clsf);
154         });
155         $('#' + this.container + '_nocls').change(function(){
156             var clsf = {method: thismap.classification, nocls: $(this).val(),
157             colorscheme: thismap.colorscheme};
158             thismap.update_classification(clsf);
159         });
160         $('#' + this.container + '_color').change(function(){
161             var clsf = {method: thismap.classification, nocls: thismap.nocls,
162             colorscheme: $(this).val()};
163             thismap.update_classification(clsf);
164         });
165     },
166
167     add_legend: function(){
168         var map_controller = $("#" + this.controller);
169         this.legend = this.container + '_legend';
170         map_controller.append('<div class="legend_button"><p>Legend</p><div id="
                ' +
171         this.legend + '" class="legend_content"></div></div>');
172     },
173
174     set_colorscheme: function(cs){
175         this.colorscheme = cs;
176         this.map.container().setAttribute("class", this.colorscheme);
177         if (this.classes != null)
178             this.setLegend();
179     },
180
181     update_classification: function(clsf){
182         if (this.colorscheme != clsf.colorscheme)
183             this.set_colorscheme(clsf.colorscheme);
184         if (this.classification != clsf.method || this.nocls != clsf.nocls){
185             this.classification = clsf.method;
186             this.nocls = clsf.nocls;
187             this.classifyRates();
188             var t1 = (new Date()).getTime();
189             this.renderRateMap('classificationUpdated');
190             console.log('renderRateMap.' + ((new Date()).getTime() - t1));
191         }
192     },
193
194     getClsName: function(val){
195         return "q" + this.classids[val] + "-" + this.classes.get_n();
196     },
197
198     get_classids: function(){
199         for (var r in this.rates){
200             var r_value = this.rates[r];
201             if (this.probabilities != null && this.probabilities[r] > this.
                    prob_threshold)
202                 this.classids[r_value] = 0;
203             else {
204                 this.classids[r_value] = this.classes(r_value);
205             }
206         }
207     },
208
209     setLegend: function(){
210         var legend_content = $("#" + this.legend);
```

```
211         legend_content.empty();
212         var css = document.styleSheets[0].cssRules[1].styleSheet.cssRules;
213         var colors = {};
214         for (var i = 0; i < css.length; i++){
215             if (css[i].selectorText.toLowerCase().indexOf("." +
216             this.colorscheme.toLowerCase()) != -1){
217                 colors[css[i].selectorText.toLowerCase()] =
218                 css[i].style.cssText.substring(6);
219             }
220         }
221         var breakpoints = this.classes.get_q(),
222             n = this.classes.get_n();
223         for (var i = 0; i < breakpoints.length - 1; i++){
224             var cls = "q" + i + "-" + n;
225             var color = colors["." + this.colorscheme.toLowerCase() + "␣." + cls
                ];
226             var classItem = '<span␣style="background:␣' + color +
227             ';">   </span> ';
228             classItem += breakpoints[i].toPrecision(3).toString() + '␣~␣' +
229             breakpoints[i+1].toPrecision(3).toString();
230             classItem += '<br>';
231             legend_content.append(classItem);
232         }
233     },
234
235     classifyRates: function(){
236         if (this.classification == 'Quantile')
237             this.classes = pv.Scale.mapquantile().intervals(this.nocls);
238         else if (this.classification == 'Percentile')
239             this.classes = pv.Scale.percentile().intervals();
240         else if (this.classification == 'Boxmap')
241             this.classes = pv.Scale.boxmap().intervals();
242         else if (this.classification == 'Equal_Interval')
243             this.classes = pv.Scale.equal_interval().intervals(this.nocls);
244         this.classes.domain(pv.values(this.rates)).range();
245         this.setLegend();
246     },
247
248     changeFeatureBoundaryColor: function(){
249         var pMap = this;
250         return function(evt, pnt){
251             var elm_id = evt.target.getAttribute("id");
252             var sep = elm_id.indexOf('/');
253             elm_id = elm_id.slice(sep - 1);
254             var elm_meta = elm_id.split("_");
255             var elm_tile = elm_meta[0].split("/");
256             var z = elm_tile[0];
257             var column = parseInt(elm_tile[1]);
258             var row = parseInt(elm_tile[2]);
259             if (pMap.tiles == null) pMap.tiles = pMap.layer.tiles();
260             var toHighlight = [evt.target];
261             var targetElement = $("#" + pMap.container);
262             for (var i = column - 1; i <= column + 1; i++){
263                 if (i < 0) continue;
264                 for (var j = row - 1; j <= row + 1; j++){
265                     if (j < 0) continue;
266                     var search_tile = [z, i, j].join("/");
267                     if (search_tile != elm_meta[0]){
268                         var s = targetElement.find('path[id=' + pMap.container +
                            '_' +
269                         search_tile + "_" + elm_meta[1] + "]");
270                         if (s.length == 1) toHighlight.push(s[0]);
271                     }
272                 }
```

```
273             }
274             for (var i = 0; i < toHighlight.length; i++)
275                 pMap.highlightFeature(toHighlight[i], toHighlight[i].
                        getAttribute("class"));
276
277             window.postMessage("evt://featHighlighted/" + pMap.container + "/" +
278             String((new Date()).getTime()), "*");
279
280             if ((evt.layerX != null && evt.layerY != null) || pnt){
281                 var evtLatLng = null;
282                 if (pnt) evtLatLng = pMap.map.pointLocation(pnt);
283                 else evtLatLng = pMap.map.pointLocation({x:evt.layerX, y:evt.
                        layerY});
284                 var elm_id = toHighlight[0].getAttribute("id"),
285                     cur_class = toHighlight[0].getAttribute("class"),
286                     sep = elm_id.indexOf('/');
287                 window.postMessage("evt://highlightFeature/"+ elm_id.slice(0,sep
                        -2) + "/"
288                 + String(evtLatLng.lat) + "/" + String(evtLatLng.lon) + "/" +
                        cur_class +
289                 "/" + elm_id.slice(sep-1).replace(/\//g, '_'), "*");
290             }
291         };
292     },
293
294     highlightFeature: function(feat, ref_class){
295         var cur_class = feat.getAttribute("class"),
296             ref_selected = ref_class.search('selected'),
297             cur_selected = cur_class.search('selected'),
298             feat_id = feat.getAttribute("id").split("_").pop();
299         if (ref_selected == -1 && cur_selected == -1) {
300             this.selections[feat_id] = true;
301             feat.setAttribute("class", cur_class + "␣" + "selected");
302         }
303         else if (ref_selected != -1 && cur_selected != -1){
304             delete this.selections[feat_id];
305             feat.setAttribute("class", cur_class.split("␣")[0]);
306         }
307     },
308
309     load: function() {
310         var pMap = this;
311         return function(e){
312             for (var i = 0; i < e.features.length; i++) {
313                 var feature = e.features[i];
314                 var feat_id = feature.data.id;
315                 if (feat_id in pMap.rates){
316                     var fid = pMap.container + '_' + e.tile.key + '_' + feat_id;
317                     if (feature.element.getAttribute("id") != fid)
318                         feature.element.setAttribute("id", pMap.container + '_'
                                +
319                         e.tile.key + '_' + feat_id);
320                     if (feat_id in pMap.filtered) feature.element.
                            removeAttribute("class");
321                     else {
322                         var cls = pMap.getClsName(pMap.rates[feat_id]);
323                         if (feat_id in pMap.selections) cls += "␣selected";
324                         feature.element.setAttribute("class", cls);
325                     }
326                     feature.element.appendChild(po.svg("title").appendChild(
327                         document.createTextNode(feat_id + ":␣" + pMap.rates[
                                feat_id]))
328                         .parentNode);
329                     $(feature.element).click(pMap.changeFeatureBoundaryColor());
```

```
330                 }
331             }
332             if (e.type == 'load'){
333                 pMap.rendered[e.tile.key] = true;
334                 var tiles = pMap.layer.curtiles();
335                 var z = pMap.map.zoom();
336                 var rendered = true;
337                 for (var tile in tiles){
338                     if (tile[0] != String(z)) continue;
339                     if (!(tile in pMap.rendered)) {
340                         rendered = false;
341                         break;
342                     }
343                 }
344                 if (rendered){
345                     window.postMessage("evt://loadEnd/" + pMap.container + "/" +
346                     String((new Date()).getTime()) + "/361", "*");
347                 }
348             }
349         };
350     },
351 
352     filterRateMap: function(res, stat, xhr){
353         var t0 = (new Date()).getTime();
354         console.log(this.container + ":Starting to filter rates:" + t0);
355         eventLog.push([this.container, "StartToFilter", t0]);
356         if (res != null) this.probabilities = res.data;
357         this.tiles = this.layer.tiles();
358         if (this.tiles == null || this.probabilities == null || this.rates ==
                null) return;
359         var threshold = $("#" + this.prob_filter).val();
360         this.get_classids();
361         var t1 = (new Date()).getTime();
362         console.log(this.container + ":Ending to filter rates:" + t1);
363         eventLog.push([this.container, "EndToFilter", t1]);
364         rateTimes[this.container].push(t1 - t0);
365         var t2 = (new Date()).getTime();
366         console.log(this.container + ":Starting to render:" + t2);
367         eventLog.push([this.container, "StartToRender", t2]);
368         for (var tile in this.tiles){
369             var feats = this.tiles[tile];
370             for (var feat in feats){
371                 var feat_id = feats[feat].data.id;
372                 var cls_id = this.classids[this.rates[feat_id]];
373                 if (cls_id == 0){
374                     this.filtered[feat_id] = true;
375                     feats[feat].element.removeAttribute("class");
376                 }
377                 else {
378                     var cls = this.getClsName(this.rates[feat_id]);
379                     delete this.filtered[feat_id];
380                     feats[feat].element.setAttribute("class", cls);
381                 }
382             }
383         }
384         var t3 = (new Date()).getTime();
385         console.log(this.container + ":Ending to render:" + t3);
386         eventLog.push([this.container, "EndToRender", t3]);
387         rateTimes[this.container].push(t3 - t2);
388         window.postMessage("evt://filteredByProb/"+ this.container + "/" +
389         String((new Date()).getTime()), "*");
390     },
391 
392     renderRateMap: function(res, stat, xhr){
```

```
393         var loadFunc = this.load();
394         if (res != null && typeof res != "string") {
395             this.rates = res.data;
396             this.layer.on('load', loadFunc);
397             this.layer.on('show', loadFunc);
398         }
399         this.tiles = this.layer.tiles();
400         if (this.tiles == null || this.rates == null) return;
401         var t0 = (new Date()).getTime();
402         console.log(this.container + ":Starting to classfy rates:" + t0);
403         eventLog.push([this.container, "StartToClassify", t0]);
404         if (this.classes == null) this.classifyRates();
405         this.get_classids();
406         var t1 = (new Date()).getTime();
407         var classificationTime = t1 - t0;
408         console.log(this.container + ":Completed to classfy rates:" + t1);
409         eventLog.push([this.container, "EndToClassify", t1]);
410         console.log(this.container + ":Time for rate classification:" +
                classificationTime);
411         rateTimes[this.container].push(classificationTime);
412         var t2 = (new Date()).getTime();
413         eventLog.push([this.container, "StartToRender", t2]);
414         for(var tile in this.tiles) {
415             if (this.tiles[tile] instanceof Array){
416                 console.log(tile + ":" + this.tiles[tile].length);
417                 var t11 = (new Date()).getTime();
418                 loadFunc({tile:{key:tile}, features:this.tiles[tile]});
419                 var t22 = (new Date()).getTime();
420                 if (arguments.length == 1) console.log((t22 - t11));
421             }
422         }
423         var t3 = (new Date()).getTime();
424         var renderTime = t3 - t2;
425         console.log(this.container + ':Time elapsed since classification is done
                :'
426         + renderTime);
427         eventLog.push([this.container, "EndToRender", t3]);
428         rateTimes[this.container].push(renderTime);
429         if (arguments.length == 1)
430             window.postMessage("evt://" + arguments[0] + "/"+ this.container + "
                    /" +
431             String((new Date()).getTime()), "*");
432         else
433             window.postMessage("evt://layerRendered/"+ this.container + "/" +
434             String((new Date()).getTime()), "*");
435     },
436
437     getRates: function(data, evt, pop, sm_method, wgt){
438         var qs = this.qs = {
439             'service': 'smoothing',
440             'data': data,
441             'e': evt,
442             'b': pop,
443             's_method': sm_method
444         };
445         if (sm_method == 'Spatial Empirical Bayes' || sm_method == 'Spatial Rate
                ' ||
446         sm_method == 'Locally Weighted Average')
447             qs['w'] = this.qs['w'] = wgt;
448         var pMap = this;
449         var t1 = (new Date()).getTime();
450         rateStartTimes[qs.s_method].push(t1);
451         console.log(this.container + ':Starting the acquisition of rates:' +  t1
                );
```

```
452         eventLog.push([this.container, "StartToGetRate", t1]);
453         $.ajax(
454             {url:'http://129.219.93.206/web_esda/service_proxy/smoothing/',
455              data: qs,
456              dataType: 'jsonp',
457              success: function(res, stat, xhr){
458                  var t2 = (new Date()).getTime();
459                  console.log(pMap.container + ':Completing the acquisition of
                         rates:' + t2);
460                  eventLog.push([pMap.container, "EndToGetRate", t2]);
461                  var ratetime = t2 - t1;
462                  console.log(pMap.container + ':Getting rates:' +  ratetime);
463                  rateTimes[pMap.container].push(ratetime);
464                  pMap.renderRateMap(res, stat, xhr);
465              }
466             }
467         );
468     },
469
470     startGettingRates: function(){
471         this.getRates(this.data, this.evt, this.pop, this.sm_method, this.wgt);
472     },
473
474     init: function(data, evt, pop, sm_method, wgt){
475         this.initTime = (new Date()).getTime();
476         console.log(this.container + ": Starting to draw a map:" + this.initTime)
                 ;
477         eventLog.push([this.container, "StartMapDraw", this.initTime]);
478         this.draw_map();
479         this.data = data;
480         this.evt = evt;
481         this.pop = pop;
482         this.sm_method = sm_method;
483         this.wgt = wgt;
484         var thismap = this;
485         myEventObserver.addEventListener("highlightFeature", function(evtData){
486             var edata = evtData.split('/');
487             if (edata[1] != thismap.container){
488                 var lat = parseFloat(edata[2]),
489                     lng = parseFloat(edata[3]),
490                     ref_class = edata[4],
491                     tileInfo = thismap.map.locationCoordinate({lat:lat, lon:lng})
                             ,
492                     elm_id = edata[5].split('_'),
493                     targetElement = $("#" + thismap.container);
494                 var tile_index = tileInfo.zoom + '/' + Math.floor(tileInfo.column
                     ) + '/' +
495                 Math.floor(tileInfo.row);
496                 elm_id = thismap.container + '_' + tile_index + '_' + elm_id[
                     elm_id.length - 1];
497                 var s = targetElement.find('path[id=' + elm_id + ']');
498                 if (s.length == 1) $(s[0]).click();
499             }
500         });
501     }
502 }
```

A.3.3 Event handling component

```
1  function MessageEventHandler(){
2      this.listeners = {};
3      var that = this;
4      window.addEventListener("message",function(evt){
5              if (evt.data.slice(0,6)=="evt://")
```

```
6                   that.onMessage(evt);
7           },false);
8   }
9   MessageEventHandler.prototype.addEventListener = function(evtName,callback){
10      if (evtName in this.listeners){
11          this.listeners[evtName].push(callback);
12      } else {
13          this.listeners[evtName] = [callback];
14      }
15  }
16  MessageEventHandler.prototype.removeEventListener = function(evtName){
17      if (evtName in this.listeners){
18          delete this.listeners[evtName];
19      }
20  }
21  MessageEventHandler.prototype.onMessage = function(evt){
22      var evtData = evt.data.slice(6);
23      var evtName = evtData.split('/')[0];
24      if (evtName in this.listeners){
25          for (var i=0; i<this.listeners[evtName].length; i++){
26              this.listeners[evtName][i](evtData);
27          }
28      }
29  }
```

APPENDIX B
DETERMINATION OF SIMPLIFICATION TOLERANCE

```
import shapefile
import numpy as np
from shapely.geometry import Point, LineString
from polygon_from_lines import lines_to_polygons

def get_dp_max_tolerance(line):
    """
    compute the maximum tolearnce value for the given line

    Parameters
    ----------
    line: a Line object

    Returns: a tuple of
             (the index of the tolearnce, the tolerance)
    """
    coords = list(line.coords)
    if len(coords) == 2:
        return
    line_vertices = coords[1:-1]
    base_line = LineString([coords[0],coords[-1]])
    tolerances = []
    for vertex in line_vertices:
        tolerances += [Point(vertex).distance(base_line)]
    tolerances = np.array([-1] + tolerances + [-1])
    return tolerances.argmax(), tolerances.max()

def get_dp_tolerances(line, loc=0):
    """
    compute the maximum tolearnce values for the left
    and right segments of the line

    Parameters
    ----------
    line: a Line object
    loc: an integer
         the index of a coordinate at which the line is split
         to two pieces

    Returns: a dictionary where
             the key is the index of coordinates from which
             the tolerance is obtained, and
             the value is the tolerance
    """
    coords = list(line.coords)
    tolerances = {}
    r = get_dp_max_tolerance(line)
    if r:
        tolerances[r[0] + loc] = r[1]
        left, right = coords[:r[0]+1], coords[r[0]:]
        r1 = get_dp_tolerances(LineString(left), loc + 0)
        r2 = get_dp_tolerances(LineString(right), loc + r[0])
        if r1: tolerances.update(r1)
        if r2: tolerances.update(r2)
    return tolerances

def get_vertice_count(lines_list):
    """
    get the numbers of vectices constituting lines

    Parameters
    ----------
    line: a list of Line objects

```

```
65      Returns: a list of integers
66               each integer is the number of vertices in a line
67      """
68      return sum([len(t) for line, t, s in lines_list])
69
70  def apply_dp_simplification(lineset, tolerance):
71      """
72      return simplified lines
73      by using Douglas-Peucker algorithm
74
75      Parameters
76      ----------
77      lineset: a list of Line objects
78      tolerance: a float, tolerance threshold
79
80      Returns: a list of simplified Line objects
81      """
82      lines = []
83      for line, tolerances, shared in lineset:
84          coords = list(line.coords)
85          if coords[0] == coords[-1] or (shared == 1 and len(coords) > 3):
86              sim_line = list(LineString(coords[:-1]).simplify(tolerance, True).
                    coords)
87              sim_line.append(coords[-1])
88          else:
89              sim_line = list(line.simplify(tolerance, True).coords)
90          if len(sim_line) == 2 and (sim_line in lines or list(reversed(sim_line))
                in lines):
91              if len(coords) > 3:
92                  sim_line = list(LineString(coords[:-1]).simplify(tolerance, True
                        ).coords)
93                  sim_line.append(coords[-1])
94              else:
95                  sim_line = coords
96          lines.append(sim_line)
97      return lines
98
99  def lines_to_shapefile(lines, recs, out_filename, ref_filename):
100     """
101     writes lines to a shape file
102
103     Parameters
104     ----------
105     lines: a list of Line objects
106     recs: a list of attributes of the Line objects
107     out_filename: the name of an output shape file
108     ref_filename: the name of a reference output file from which
109                   data columns are copied to the output shape file
110
111     Returns: a shape file
112     """
113     out_shp = shapefile.LineShp(out_filename, 'w')
114     out_shp.copy_fields(ref_filename)
115     out_data = zip(lines, recs)
116     for line, rec in out_data:
117         X = [coord[0] for coord in line]
118         Y = [coord[1] for coord in line]
119         out_shp.add(X, Y, rec)
120
121 if __name__ == '__main__':
122
123     testfile = 'test.shp'
124     shp = shapefile.LineShp(testfile)
125
```

```
126     line2tolerance = []
127     dmax, dmin = 0, 0
128     recs = []
129     for line, rec in shp:
130         l = LineString([tuple(coord) for coord in line['coordinates']])
131         recs.append(rec)
132         tolerances = [t[1] for t in sorted(get_dp_tolerances(l).items())]
133         if len(tolerances) > 0:
134             max_tolerance = max(tolerances)
135             min_tolerance = min(tolerances)
136         if max_tolerance > dmax:
137             dmax = max_tolerance
138         if min_tolerance < dmin:
139             dmin = min_tolerance
140         line2tolerance.append((l, [-1] + tolerances + [-1], rec['InnerRingS']))
141 
142     print 'the maximum tolerance is ', dmax
143     print 'the minimum tolerance is ', dmin
144     print 'the total number of vertices is ', get_vertice_count(line2tolerance)
145 
146     tolerance_range = dmax - dmin
147     for i in [0.2, 0.16, 0.12, 0.08, 0.04]:
148         t = dmin + i*tolerance_range
149         lineset = apply_dp_simplification(line2tolerance, t)
150         line_data_name = 'test_%s.shp' % str(t)
151         lines_to_shapefile(lineset, recs, line_data_name, testfile)
152         polygon_data_name = 'test_poly_%s.shp' % str(t)
153         lines_to_polygons(line_data_name, polygon_data_name)
```