Tiled Vectors: A Method for Vector Transmission over the Web

Vyron Antoniou[1], Jeremy Morley[2], and Mordechai (Muki) Haklay[1]

[1] Department of Civil, Environmental & Geomatic Engineering
University College London
Gower St., WC1E 6BT, London, UK
[2] Centre for Geospatial Science, University of Nottingham
University Park, NG7 2RD, Nottingham, UK
{v.antoniou,m.haklay}@ucl.ac.uk, jeremy.morley@nottingham.ac.uk

Abstract. Transmitting vector data over the Web is a challenging issue. Existing methods for vector delivery over the Web focus on progressive transmission techniques. Long standing problems in the formalization of dynamic generalization and the time needed for complex algorithms does not allow their implementation in real life applications. We propose a new method for data transmission over the Web based on tiles. We show that in client-server architecture the coordination of all involved parts can create an efficient way to transmit vectors. We describe the methodology of how we can implement the successful for raster data, tile-based method to tackle the particularities of vector data transmission.

Keywords: Vector data, Geographic Information, Web mapping, AJAX.

1 Introduction

From the early days of the World Wide Web (or Web), it was clear that this new medium could facilitate the dissemination of spatial information. Indeed, the new medium has changed dramatically the way that maps and geographical information are presented and used, and consequently the way that cartographers 'design, produce and deliver' maps [1]. Helped by the evolution of Web 2.0, mapping applications and spatial information is now ubiquitous on the Web. Interestingly though, the vast majority of the maps available on the Web today are raster based. This is because transmission methods for raster data over the Web are well established and easily implemented.

Nevertheless, there are specific cases where raster images are inadequate. Researchers [2], [3] have pointed out the limitations of raster-only mapping applications and that in many cases the Web mapping applications require the user to be able to interact directly with the cartographic entities presented on the map. Interactivity and direct object manipulation is also essential in exploratory spatial data analysis [4], [5], [6], [7].

Despite the need for mapping applications to be able to host vector data, vector maps suffered from setbacks that prevented wide implementation. These problems stem from the voluminous nature of vector data and range from limitations inherent in each of the formats introduced to intrinsic disadvantages of vector encoding (such as on-the-fly generalization and efficient transmission over the Web). In order to tackle the latter issue, many efforts introduced are trying to imitate, with limited success in real-life applications, the successful progressive raster data transmission methods to the vector transmission problem.

This paper is presenting a new method for XML and text-encoded vector data transmission over the Web. Instead of creating a technique for progressive transmission, we suggest that sending asynchronously tiled parts of vector data from the server to the client can help to build interactive Web mapping applications. The tile-based method for raster data delivery has been successfully implemented by major mapping providers like Google, Yahoo! and Microsoft. In fact, the explosion of mapping applications on the Web and the consecutive phenomena of map mash-ups, neogeography [8] and VGI [9] have been based on the efficiency and easiness of raster data delivery using the tile-based technique. In this paper we present a tile-based method for data transmission, tailored to the particularities of vector data. The document is structured as follows: in Section 2 we present related work on the subject. The proposed methodology is explained in Section 3 followed by performance evaluation in Section 4. We discuss our findings and give indications for our future work in Section 5.

2   Related Work

2.1   Raster Data Transmission

Many progressive transmission techniques for raster data have been introduced. [2], [10] and [3] offer reviews of those techniques and describe in brief their main characteristics. In general, highly sophisticated compression algorithms and interleaving transmission methods made raster formats suitable for data delivery over the Web. The efficiency of these methods enabled the Geographic Information (GI) community to build the majority of Web mapping applications using raster data and thus easily deliver spatial information to the users.

The evolution of the Web and the pursuit of enhanced responsiveness and increased usability for Web applications lead to the development of new programming techniques like AJAX (Asynchronous Javascript and XML) and a new method of raster data transmission that uses tiled raster images and different levels of detail (LoD). According to this method the highest LoD of the mapping area is divided into a number of quadrants (tiles). Each of these quadrants is further sub-divided into new tiles that form the next LoD. This process continues until the lowest LoD is reached. Although this can lead to a huge number of tiles for a detailed or large dataset, the storage, indexing and handling of raster files is straightforward, especially as data storage becomes cheaper.

When a map of a given LoD is requested a number of tiles are sent to the user. The tiles are loaded into the Web browser window as a matrix, and from the user's perspective it seems to be continuous image. For any consequent request such as pan,

zoom or managing layers a new request is made by the application that runs on the client's Web browser and the server transmits to the client only the tiles that are needed in addition to the ones currently in the client's cache. This method makes the application considerably faster and more responsive since the use of AJAX techniques reduces the application's response time by communicating with the server without the user actually noticing it.

2.2   Vector Data Transmission

The volume of the vector data and the difficulty of transmitting it over the Web has been a long standing problem for Web mapping and Web GIS. The success of the progressive transmission methods for raster data turned the focus of research towards the development of similar techniques, tailored to vector encoding.

Efficient methods of progressive transmission have been introduced for a particular case of vector data: triangular meshes. Triangular meshes are usually used to describe digital terrain models or the surface of 3d objects. On the contrary, progressive transmission of cartographic vector data over the Web remains problematic despite numerous efforts (see [2], [3] and [11] for a review). According to progressive transmission methods a coarser map version is sent initially to the user, and depending on the user's requirements, consecutive data packets are send to improve the map. The coarser map versions can either be generated dynamically at the time of the request (on-the-fly) or can be pre-calculated through the process of generalization. On-the-fly generalization is an unsolved problem for cartography. A number of researchers have focused on dynamic generalization ([12], [13], [14]) in order to enhance progressive vector transmission but since there are no formalized cartographic generalization principles and applicable generalizing operators [15], automated dynamic generalization still remains a challenge. Moreover, the existing generalization algorithms are time-demanding and thus not applicable for real-life Web application. Another disadvantage of dynamic generalization is that it produces inconsistent results in terms of retaining topology and geometry attributes and thus often need an *a posteriori* evaluation of their consistency. A topologic consistent approach has been proposed for polygons and lines by [11] with the exception of isolated polygons that have area smaller than a given threshold and lines that belong to the smallest category (for example first-order streams in a river network).

On the other hand, off-line generalization and the creation of different LoDs is the norm for mapping agencies. In this case, generalization is time-insensitive and is usually performed interactively by expert cartographers with the help of specialized software [12]. [2] presented a method for pre-computing and storing multiple map representations suitable for progressive transmission to the user but without further implementation. Although the maintenance of different LoD is cumbersome, off-line generalization yields topologically and geometrically accurate products, and similar data management techniques to these used to maintain multi-LoD raster databases can be applied to such vector databases.

The advantage of progressive data transmission relies on the fact that users can perform preliminary operation even on a coarser version of the map or they can assess the suitability of the map requested and possibly change their request without waiting

for the whole dataset to be downloaded. The process is shown in Figure 1, where A, B, C, D1 and D2 are various time periods of the process. Period A is the time during mouse move, B is the time that the user waits for the coarser version of the map to be loaded, C is the time the server needs to extract the data. D1 is the time the user observes the map while it becomes more detailed and D2 is the time that the user observes the fully detailed map. Progressive transmission enables users to start observing the map before the whole map is downloaded.
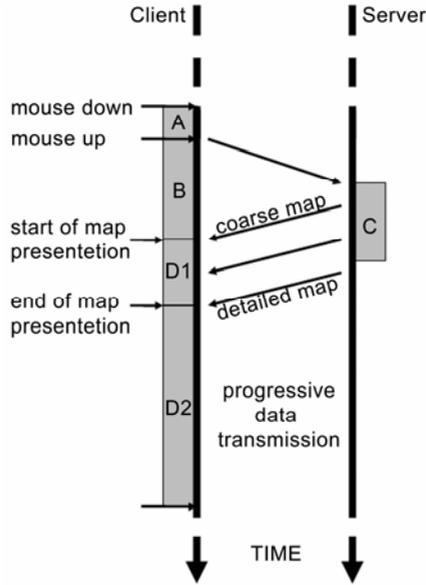

Fig. 1. Steps and time periods in the progressive transmission of vector data

Recognizing the need for more efficient transmission methods, some early efforts on vector tiling have been introduced ([16], [17]). These methods are based on the off-line preparation of vector tiles which introduces an extra step in the preparation and maintenance of the map (i.e. to create and maintain the tiles for each LoD). The main disadvantage though is that the proposed methods mainly focus on the simple visualization of the map and do not provide a merging mechanism of the tiles at the client side. Thus, the user is presented with a map composed by segmented entities at the edges of the tiles which leads in semantic inconsistencies between the real entities and the map entities presented to the user. In contrast, in [18] we suggested that a mechanism that provides a solution for on-the-fly tiling of the data on the server and the merging of the tiles on the client will be more appropriate.

3 Methodology

Instead of trying to implement a progressive transmission technique tailored to vector data needs, the method presented here follows the tile-based approach. In brief, according

to the proposed methodology, asynchronous data requests are submitted to the server only if the data has not already been sent to the user, otherwise data are read from the browser's cache. When a user's request reaches the server the data are cut into tiles and then send to the user's browser. At the user's machine the tiles are merged and the final map is presented to the user (see Table 1 for the steps followed after a typical map request). This approach provides a method to transmit vector data to the client using AJAX, but it does not solve the problem of on-the-fly vector generalization. Thus, this approach is applicable when we have pre-prepared multi-resolution spatial databases. It is understandable that the effective implementation of a methodology based on the client-server architecture needs the coordination of all engaged parts (spatial database, server, user's browser and the map document itself). In what follows we describe the architecture of the methodology with an emphasis on the structure of the map document, the interaction of map document with the browser and the server and the map preparation (i.e. tile merging).

3.1   The Map Document

The role of the map document's structure is central for the methodology. The map document has three different layers (Figure 2). The first layer consists of a 5x5 grid of tiles (background area). This layer is not visible to the user but is used to hold the tiled data sent to the map document either by the server or the browser's cache memory. Also, this layer provides data to the next layer of the map document. The second layer (viewable area) consists of a 3x3 grid of tiles. These tiles are cloned from the background area (first layer). Its role is to hold the data that are going to be merged and then assigned to the thematic layers at the next layer of the document. The third layer (map area) consists of one tile. This layer is the actual map requested by the user and is compiled from the thematic layers according to cartographic rules.
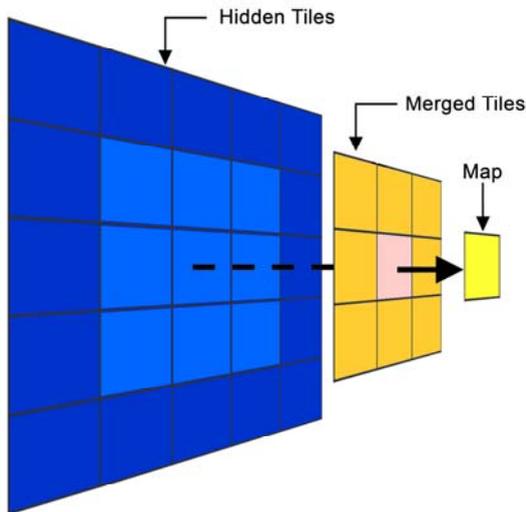

Fig. 2. The structure of the map document

The cloning of data from one part of the document to another is triggered by the user's actions. When the user requests to view (by panning the map) more data, the tiles that are already stored in the hidden area of the document are cloned to the merging area, where the merge of geometries takes place (see next paragraph), and finally are assigned to the correct thematic layers which are presented to the user as the final map. At the same time, the map document prepares itself for the next user actions by requesting new data either from the browser's cache memory or from the server.

3.2 Merging

An important step of the whole methodology is the merging process that takes place at the client. The geometries stored in each of the 9 tiles of the viewable area are merged before they get assigned to the correct thematic layer. This step is needed because the map entities presented to the user should be in logical accordance with the entities stored in the database. For example, if a single polygon is split into two polygons (each one stored in a different tile) during the extraction from the database, when presented to the user these polygons should be merged back into one entity. This will allow users to interact correctly with the elements of the map. Since thematic layers can hold either point, line or polygon geometry there needs to be a merging mechanism for every type of geometry.

3.2.1 Points
The merging of points is trivial, since the only thing needed is to clone all points from the tiles to the final thematic layers. Javascript can easily parse the Document Object Model (DOM) of XML documents and clone data from one part of the document to another.

3.2.2 Lines
The merging of lines is based on the use of the unique feature IDs. IDs are used as keys to search inside the 9 tiles of the viewable area. Line segment that have the same ID are grouped and then joined into single entities. Such a join may result either in a
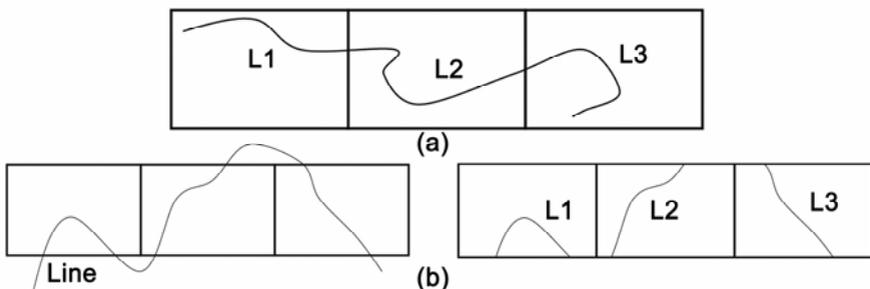

Fig. 3. The merge of a line that resides in different tiles can lead to: (a) polyline element or (b) multi-polyline element

polyline feature when the segments have common points or in a multi-polyline feature when there are no common points among segments that have the same ID, depending on how the original line, stored in the database, was split into tiles (Figure 3).

3.2.3 Polygons

Unique IDs are also used for merging polygons. Once again, polygons that share the same ID are grouped and then merged either into polygon or multi-polygon entities. The case of polygon merging though, presents a greater degree of difficulty in order to dis-ambiguate all possible cases. The main obstacle is the presence of unwanted border lines generated during the tiling process (see for example Ch. 3 of [19] for details about spatial operations and how new line segments are generated through the tiling-intersection process). For example, Figure 4a and 4b show two different cases of a bor-der line appearance. In Figure 4a a polygon border line generated during the tiling process is needed to achieve the correct coloring of the polygon presented to the user. In contrast, in Figure 4b that same border line is causing an unwanted visual effect. The correct way of rendering the merged parts of the polygon in 4b, is shown in Figure 4c.
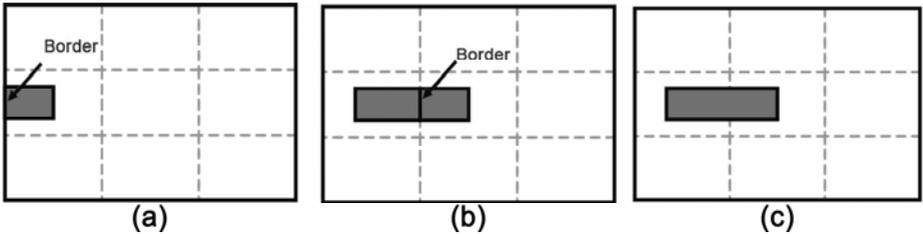

Fig. 4. Dealing with border lines during the merge of polygons

Still, the border lines generated during the tiling process are necessary during the merging phase since they help elucidate the rendering of polygons. For example, Figure 5 shows that when border lines are absent there is no indication of how a poly-gon should be colored.
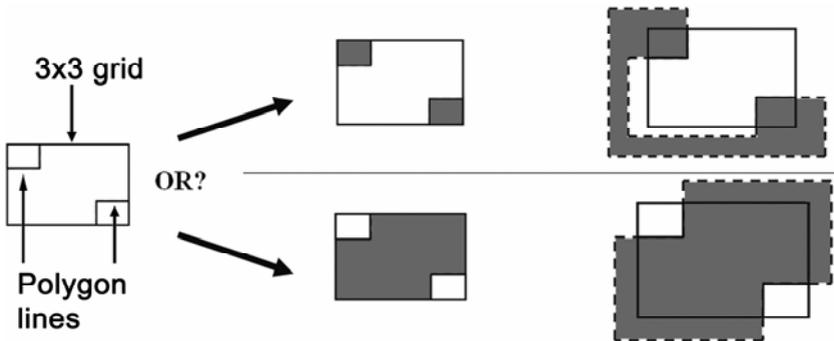

Fig. 5. A vague case of polygon coloring

To tackle that problem we need to have both the border lines of the polygons and an indication regarding when each border line should be used in the merging process. Since border lines are generated by the intersection of the polygon and the tile it is obvious that these border lines will coincide geometrically with the outline of the tile. We need to record at which side (1, 2, 3 or 4) of the tile (see Figure 6) there was a border line created. This element allows us to create a set of rules regarding the inclusion or not of the border line in the merging process. For example, border lines that coincide with the number 1 side of a tile should be included in the process only if the tile is placed in the (0, j) places of the 3x3 grid. In any other case the border line should be excluded by the process.
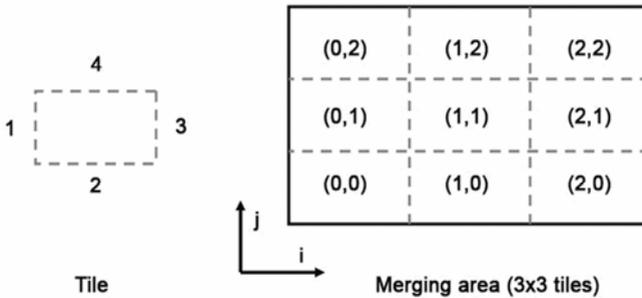

Fig. 6. The method to assign an indicator to border lines

By doing so, in the case shown in Figure 4a the border line would have been used to form the outline of the polygon but in case shown in Figure 4b the border would have been excluded in the merging process. This approach has been successfully tested in all possible polygon cases like ring polygons, island polygons or multi-part polygons. Finally, Table 1 describes the steps that take place inside the map document.

Table 1. The steps of map preparation

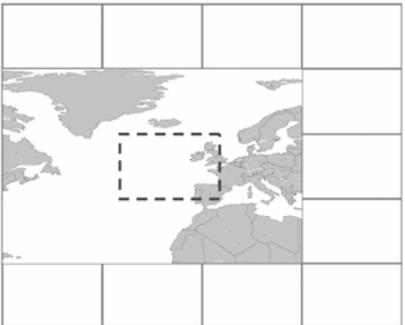| 1. The method starts with a request by the user. A number of tiles is extracted from the database, covering a broader map area and send to the user's browser. | 2. At the client's browser, for each layer, the geometry of the tiles is merged creating a continuous and correctly compiled map. |

Table 1. (*continued*)

3. The user is presented with the requested map.

4. So, in fact, while the user sees a restricted map area (the one that originally requested), a much larger area is available to accommodate the next panning gesture without the need for further client-server communication.

5. Additionally, another set of tiles is also extracted from the server and send to the user. These tiles are not merged but are kept in the background section of the map which is not visible to the user. These tiles work as cache memory inside the document to accommodate the next panning gestures without making the user wait for new data.
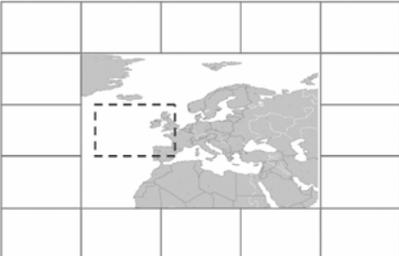
6. When the user makes a panning gesture the map document is able to respond without delay. Nevertheless, the map document starts the process to prepare itself in order to accommodate the next user's moves by re-arranging the tiles in the map document and by getting the missing data, either from browser's cache memory (if the needed data are already there) or from the server (see how in the next steps).
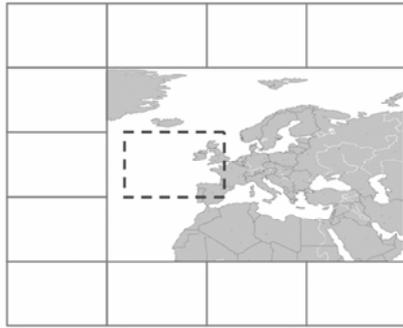
7. First the column (or row) of tiles in the background section that is on the opposite from the pan direction is dropped from the document and stored in browser's cache.
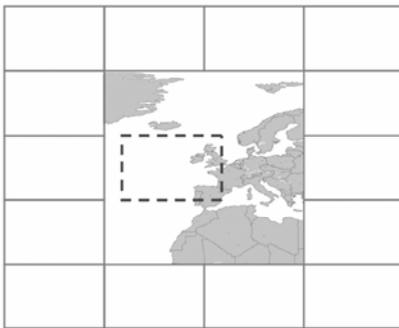
8. The tiles that exist in the viewable area and are not needed any more are dropped, but remain in the browser's cache too.

Table 1. (*continued*)

9. The tiles that reside in the background area and are needed to accommodate the new map are cloned to the viewable mapping area.

10. The geometry of the tiles is once again merged for each thematic layer.
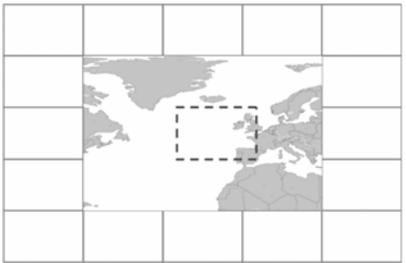
11. At the same time, a new set of tiles, needed to prepare the map document, is requested from the browser. If not found there, the request is propagated to the server. These tiles will be stored in the background area of the map document and the process is ready to start all over again**.**

3.3 Map Document's Interaction with Browser and Server

Client side caching is a common programming technique. This technique allows browsers to hold locally for later use data that have been sent from the server without any further interaction. Thus, reduced network latency and enhanced user experience is achieved. In our case, browser's cache memory is used to hold tiles of data sent from the server to the user but not any more stored inside the map document. Thus, whenever new tiles are required from the map document, the search starts in the browser's cache. If the tiles needed are stored in the cache memory, then the data is inserted into the map document. If the data has not been already sent to the user, then the request for new tiles is propagated from the client to the server.

This client-server interaction takes place asynchronously using AJAX requests. AJAX is a well known methodology that has played a key role in the evolution of Web 2.0 by helping programmers to build desktop-like applications over the Web. In Web mapping AJAX made possible to overcome long standing obstacles related with the transmission of raster data (due to their volume) or poor user interaction with the

mapping applications. In our case we use AJAX to manage the client-server commu-
nication behind the scenes. Given the fact that vector tiles have fixed dimensions
(defined by the map window of the application), the construction of spatial queries to
retrieve the missing tiles is a straightforward process.

Figure 7 shows the time periods involved in the whole process. Period A is the
mouse movement; using AJAX we can trigger a new map request while the user is
still panning in a direction. Period B is the time that the user observes the map. The
tiled method coupled with AJAX requests provides data constantly to the user and
thus there is no need for the user to wait for the map to be downloaded. Period C is
the time the server needs to extract the data. The key point of the method is that it
fully exploits the time period of user inactivity (i.e. does not interact with the map re-
questing new data). While the user observes or queries the map entities available,
behind the scenes there is work in progress in order to prepare the map document to
accommodate the next user's moves. If time period B is very small (i.e. the user is
quickly panning towards the same direction) the process of requesting new data from
the server can be suspended with the help of a time counter. This process is imple-
mented only for those requests that cannot be accommodated by data stored in
browser's cache.

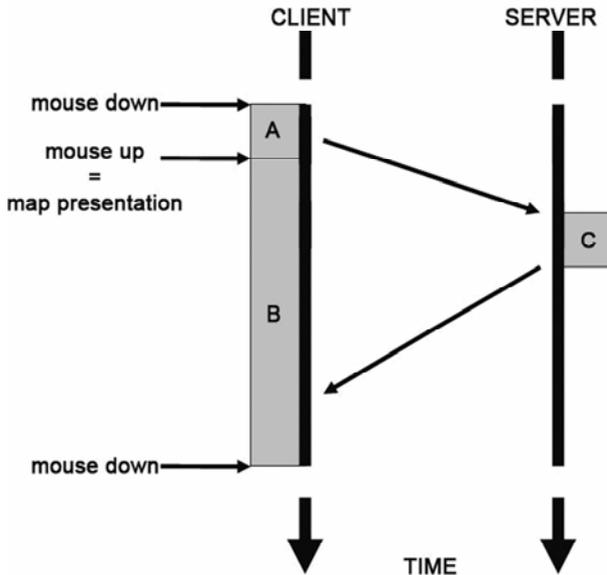**Fig. 7.** Steps and time periods in tile-based transmission of vector data

Figure 8 shows a screenshot of a prototype built in order to test and improve our
algorithms. We used SVG as the format to build our map because it is an XML based
format that supports scripting. Any practices and methods applied during the imple-
mentation can be implemented in other XML-based or text-based vector formats.
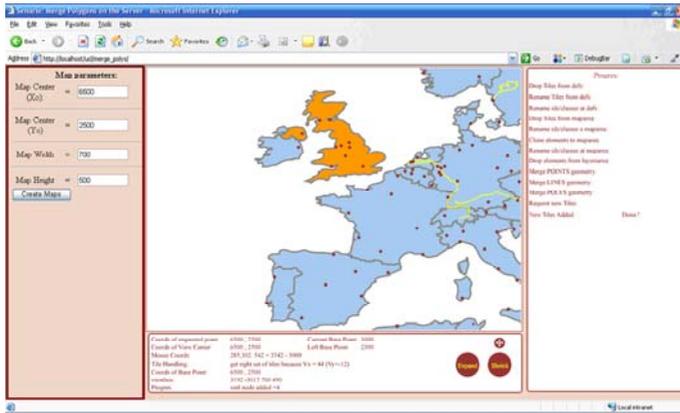
Fig. 8. A screenshot of the prototype

4   Performance

Usually the performance of progressive transmission methods for vector data are tested by examining the time needed to transmit a dataset of a certain size (for example see [20], [11], [3]). This type of performance evaluation is not applicable in our case since the method aims to the exact opposite target: to avoid the transmission of bulk sets of data and instead to split data in small packets and exploit the real-life user behavior to efficiently transmit vector data. Therefore, for the performance evaluation of our method a more user-oriented and thus more suitable method was used. We examined the time periods for panning (time period A in Figure 8) and map observation (time period B in Figure 8) that will allow our method to present to the user a ready to use map immediately (i.e. not fail to fulfill the condition $t_{mouse\ up} = t_{map\ presentation}$).

The experiments were performed over the Web using a server with 2.13 GHz CPU double-core processor and 1 GB of memory stationed in Athens (Greece), connected to the Internet with nominal value for download 24Mb/s and for uploading 1Mb/s (the actual average values recorded during the experiments were 3.1Mb/s and 280Kb/s respectively) and a client with 1.66 GHz CPU double-core processor and 1 GB memory stationed in London (UK), connected to the Internet with 8Mb/s download and 1Mb/s upload speed nominal values (the actual average values recorded during the experiments were 3.8Mb/s and 682.9Kb/s respectively). The average ping time between client and server during the experiments was 112 ms.

In our evaluation we used two different sets of data: a world level dataset that contains countries, rivers and cities and one at a street level one that contains parcels, parcels' registration points and street centerlines. Table 2 shows the analysis both of the entire datasets stored in a spatial database and of the data accessed by the user during the experiments.

We examined the performance of our method in the worst case scenario: the user was panning constantly in the same direction and thus no help from the browser's cache was provided. Instead, every user's request was propagated to the server in order to retrieve data not yet sent to the client. We recorded the time for the panning

Table 2. The analysis of the datasets used and accessed during the experiments

World Level Data

Geometry Type	Entire Dataset			Data Accessed by the User		
	Num. of Entities	Num. of Parts	Num. of Points	Num. of Entities	Num. of Parts	Num. of Points
Polygons	147	249	28,162	140	228	24,983
Polylines	98	177	3,965	95	147	3,852
Points	606	606	606	550	550	550
Total	**851**	**1,032**	**32,733**	**785**	**925**	**29,385**

Street Level Data

Geometry Type	Entire Dataset			Data Accessed by the User		
	Num. of Entities	Num. of Parts	Num. of Points	Num. of Entities	Num. of Parts	Num. of Points
Polygons	5,879	5,879	37,213	2,360	2,360	15,122
Polylines	15,186	15,186	32,128	5,941	5,941	12,535
Points	6,245	6,245	6,245	2,447	2,447	2,447
Total	**27,310**	**27,310**	**75,586**	**10,748**	**10,748**	**30,104**

gestures and the minimum time that the user had to observe the requested map until the map document prepares itself for the next panning gesture without introducing further delays.

In every step of our experiments the user was presented with a correctly composed and fully functional map. This means that all entities where styled and assigned to the correct thematic layers which were used to compose the map. Additionally, a link was assigned to every entity presented to the user so to enable further AJAX queries (for example, request the attributes of an entity from the server). Table 3 shows the time needed for the presentation of the map after the first request and the minimum, maximum and average map observation times in order the user to be constantly presented with such a map.

Table 3. Performance results of the proposed method

	First map presentation (sec)	Average Pan Time (sec)	Min - Max – Average Observation Time (sec)
World Level	7.8	0.7	0.4 - 3.5 - 1.3
Street Level	18.1	0.7	0.6 - 5.5 - 2.8

As expected, the slowest part of the method is the time needed for the presentation of the map after the first request. Subsequent map presentations though, need considerably less time which varies due to differences in the volume of data transmitted and

the number of entities that need to be merged each time. As explained earlier, in a real life scenario (i.e. the user does not pan only in one direction) the observation time is expected to be further reduced by using data stored in browser's cache memory.

5   Discussion and Future Work

The proposed method provides a smooth user interaction with the map, overcoming the problem of long waits for the download of vector data since there is no need for the user to wait for such downloads. The efficiency of the method depends on the correct coordination and refinement of all the method's steps. So, in addition to what has been discussed so far, a new database structure can be developed that will hold directly pre-calculated tiles instead of extracting them at the time of request, based on global gridding similar to existing tiled raster services. This will considerably improve server performance and server-side caching.

The key point of the proposed method is that the map document should always have the necessary data to accommodate the next user's move; the tiled vector data must reach the browser before the user requests them in order to eliminate waiting times. In other words the performance of the method is in close relationship with the user's behavior as well as the efficiency of the method itself. The bigger the time periods the user remains inactive in terms of data request, the more time is available for the preparation of the map document for the next request. In contrast with what takes place in the case of raster tiles, this preparation time varies in the vector case. Raster tiles have a standard dimension which also results in having almost a fixed size. In contrast, there is no guarantee of the size of a tile that holds vector data and this can introduce delays in the process. A solution to that could be tiles of variable dimension (i.e. different spatial coverage) that will hold data that has size below a chosen threshold. In this case specialized algorithms for tile indexing and manipulation are needed. Alternatively, progressive transmission techniques could be implemented for the data that each tile holds.

The proposed method has a number of advantages compared to the transmission methods available today. Following this strategy, a continuous vector map will be available to the user by sending only small pieces of data in order to achieve reduced network latency. The procedure does not need to interact with the server, and thus introduce network latency, for small panning gestures since the client will have enough data to accommodate such user actions. Additionally, when needed, the use of client caching and AJAX will avoid the unnecessary client-server transactions but will still allow the missing tiles to be embodied in the map behind the scene without the user noticing it.

By testing our method over the Web we proved that it can be implemented in real life applications. The approach builds upon the architecture that most mapping agencies use: multi-scale databases. While the on-the-fly generalization problem remains unsolved, map providers serve maps from different LoDs on the Web. Exploiting that fact and in contrast with the existing limitations of progressive transmission techniques, this method disturbs neither the geometry nor the topology of the features presented on the map. Moreover, it has no restrictions in handling multi layered requests of any geometry and in applying all cartographic principles during map composition.

Also, the method can be implemented to any XML or text-encoded format like KML or GeoJSON. It is interesting to note that, after the first map presentation, this method offers an efficient way so users can access unlimited volume of vector data with waiting/observation times that not differ from the typical waiting/observation times that occur when they browse any other Web application. Finally, this tiling approach makes more difficult to compromise intellectual property rights (IPRs) of vector data compared with methods that send the whole dataset to the client, progressively or not. The compromise of IPRs has been a major factor that made developers and mapping agencies reluctant to publish vector data on the Web. In the proposed method, while in the first step more data than requested are sent, this disadvantage is balanced by reduced client-server interaction and network latency during the subsequent steps.

The combination of the evolution in the Web and the efficient mechanisms for raster delivery on the one hand and the need for vector data for enhanced interactivity and object manipulation on the client side on the other form a new environment for Web mapping. We have suggested that in such an environment the role of hybrid Web maps able to host both raster and vector data will be considerably increased [21]. Such maps will use only the strong points of raster and vector data and is likely to be the most efficient way to deliver spatial data for complex Web mapping applications. In that context, our future work will focus on researching methods that will enable the harmonious cooperation of the established raster delivery methods with our proposed method for vector data transmission.

References

1. Cartwright, W.E.: Mapping in a digital age. In: Wilson, P.J., Fotheringham, A.S. (eds.) The Handbook of Geographic Information Science, pp. 199–221. Blackwell Publishing Ltd., Malden (2008)
2. Bertolotto, M., Egenhofer, M.J.: Progressive Transmission of Vector Map Data over the World Wide Web. GeoInformatica 5(4), 345–373 (2001)
3. Bertolotto, M.: Progressive Techniques for Efficient Vector Map Data Transmission: An Overview. In: Belussi, A., Catania, B., Clementini, E., Ferrari, E. (eds.) Spatial Data on the Web: Modeling and Management, pp. 65–84. Springer, New York (2007)
4. Dykes, J.A.: Exploring spatial data representation with dynamic graphics. Computers & Geosciences 23(4), 345–370 (1997)
5. Cook, D., Symanzik, J., Majure, J.J., Cressie, N.: Dynamic graphics in a GIS: More examples using linked software. Computers & Geosciences 23, 371–385 (1997)
6. Andrienko, G.L., Andrienko, N.V.: Interactive maps for visual data exploration. International Journal of Geographical Information Science 13(4), 355–374 (1999)
7. Zhao, H., Shneiderman, B.: Colour-coded pixel-based highly interactive Web mapping for georeferenced data exploration. International Journal of Geographical Information Science 19(4), 413–428 (2005)
8. Turner, A.: Introduction to neogeography. O'Reilly Media, Sebastopol (2006)
9. Goodchild, M.: Citizens as sensors: the world of volunteered geography. GeoJournal 69(4), 211–221 (2007)
10. Yang, C., Wong, W.D., Yang, R., Kafatos, M., Li, Q.: Performance-improving techniques in web-based GIS. International Journal of Geographical Information Science 19(3), 319–342 (2005)

11. Yang, B., Purves, R., Weibel, R.: Efficient transmission of vector data over the Internet. International Journal of Geographical Information Science 21(2), 215–237 (2007)
12. Cecconi, A., Galanda, M.: Adaptive zooming in Web cartography. In: SVGOpen 2002 (2002),
    `http://www.svgopen.org/2002/papers/`
    `cecconi_galanda__adaptive_zooming/index.html`
13. Lehto, L., Sarjakoski, L.T.: Real-time generalization of XML-encoded spatial data for the Web and mobile devices. International Journal of Geographical Information Science 19(8), 957–973 (2005)
14. Jones, C.B., Ware, J.M.: Map generalization in the Web age. International Journal of Geographical Information Science 19(8), 859–870 (2005)
15. Weibel, R., Dutton, G.: Generalizing spatial data and dealing with multiple representations. In: Longley, A.P., Goodchild, F.M., Maguire, J.D., Rhind, W.D. (eds.) Geographical information systems. John Wiley & Sons, Chichester (1999)
16. Campin, B.: Use of vector and raster tiles for middle-size Scalable Vector Graphics' mapping applications. In: SVGOpen 2005 (2005),
    `http://www.svgopen.org/2005/papers/`
    `VectorAndRasterTilesForMappingApplications/`
17. Langfeld, D., Kunze, R., Vornberger, O.: SVG Web Mapping. Four-dimensional visualization of time- and geobased data. In: SVGOpen 2008 (2008),
    `http://www.svgopen.org/2008/papers/92-SVG_Web_Mapping/`
18. Antoniou, V., Morley, J.: Web Mapping and WebGIS: do we actually need to use SVG? In: SVGOpen 2008 (2008),
    `http://www.svgopen.org/2008/papers/`
    `82- Web_Mapping_and_WebGIS_do_we_actually_need_to_use_SVG/`
19. Rigaux, P., Scholl, M., Voisard, A.: Spatial Databases: With Application to GIS. Morgan Kaufmann Publishers, San Francisco (2002)
20. Yang, B.: A multi-resolution model of vector map data for rapid transmission over the Internet. Computers & Geosciences 31(5), 569–578 (2005)
21. Antoniou, V., Morley, J., Haklay, M.: Is your Web map fit for purpose? Drawing a line under raster mapping. In: AGI Geocommunity 2008 (2008),
    `http://www.agi.org.uk/site/upload/document/Events/AGI2008/`
    `Papers/VyronAntoniou.pdf`