Network drawing application for use in modern web browsers

Marko Letić, Branislav Atlagić, Dragan Ivetić

***Abstract* — The problem of client-side drawing of electrical network elements on a city map in a web browser is analyzed in this paper. Element drawing is programmed in JavaScript language with the support of Cascading Style Sheets and Hyper Text Markup Language 5 elements. Algorithms for object clustering and best fit view are implemented in this solution. Application performance is analyzed in different web browsers and on different platforms.**

***Keywords* — Client-side drawing, Hyper Text Markup Language 5, JavaScript**

I. Introduction

One of the most important information that an interactive web geographic application should provide about some object is its location. Also, it should show its connections with the other ones, and provide various kind of data associated with it. Whether it's an application that gives information about the location of ATMs, all the restaurants near user's current position or more complex systems such as electrical grid systems, the user must have a clear overview of where the relevant objects are located. The application has to make the view responsive, informative and accurate, independently on the type of the target device [1].

Main subject of this research is the drawing of parts of an electrical grid that are geographically positioned on a city map. For the base city view, on which the grid is drawn, background layers implemented in *OpenStreetMap* [2] are used. These layers are provided by *MapBox* [3] service provider.

Drawing of electrical elements and their integration with the background layers is implemented using *JavaScript* language with the support of *Leaflet.js* library [4]. *Leaflet.js* library presents an open-source *JavaScript* library that implements drawing of objects on HTML5 canvas and their integration with map layers. Special attention is dedicated to the performance optimization of drawing and displaying elements on the canvas while maximizing the usage of resources provided by the client device. Picture 1 shows the application process flow diagram.

Picture 1. *Application process flow diagram*

II. Input Data Preparation

As an input data XML (*Extensible Markup Language*) document with the description of the city grid is used. Each of the elements in this document, besides the attributes that are specific to that element type, contains a unique geographic location of that element with which the mapping of that element on the city map will be implemented. Coordinates provided are in UTM (*Universal Transverse Mercator*) format. As the *Leaflet* library uses latitude and longitude input format a conversion from UTM had to be done.

III. HTML5 Canvas

Leaflet library supports two types of map and element drawing: using HTML5 canvas element or using SVG (*Scalable Vector Graphics*) format. An explanation of their differences is required to explain why the canvas was used instead of SVG.

SVG image format is an XML format, so the SVG object are directly inserted into the DOM (*Document Object Model*) three. It is possible to manipulate SVG objects using CSS and *JavaScript*. SVG represents a data model that persists in memory. As HTML creates an object model from elements, attributes and styles so does the SVG. When the SVG element is found inside the HTML document it behaves as the part of the DOM three.

HTML5 creates a bitmap inside the canvas of the browser which is easily manipulated. On the canvas it's possible to draw vector graphics but it is rendered using a raster format. One of the main differences compared to the SVG format is that the only way to manipulate the canvas is via *JavaScript*, but it adds only one line in the DOM three. Since canvas represents an HTML5 object, it's supported only in the newer web browsers. Also it is important to add that the SVG is based more on the

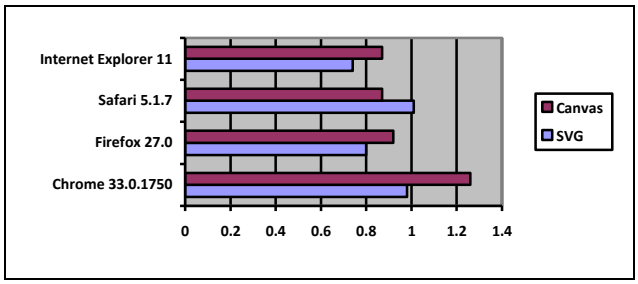
Marko Letić, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: marko.letic@uns.ac.rs).

Branislav Atlagić, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: branislav.atlagic@gmail.com).

Dragan Ivetić, Faculty of Technical Sciences, University of Novi Sad, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: ivetic@uns.ac.rs).

description of the shapes itself and the canvas is more pixel oriented.

For the need of performance comparison a simple test scenario is created using *jsPerf* web application. In the test two completely equal HTML container elements are created. Inside the first a map is drawn using the canvas while inside the other one a map is created using SVG. On the both maps the same number of elements is added. Response time of the application while performing a panning action is measured. Picture 2 shows the test results obtained by the use of different web browsers.



Picture 2. *Test results*

It is noticeable that the canvas performs better then SVG in most web browsers. The only browser that shows different results is *Safari* but as it takes only 4% of the market. Surely, much more relevant information is the performance measured in *Chrome* browser which takes 56% [5]. When the number of drawn elements is increased, the difference in performance is also higher. Information relevant for the electrical grid that is displayed on the map needs to be updated in certain time intervals. This type of update would be an extremely expensive operation if, for every change in the grid, the entire DOM three should be re-created, as the SVG demands. Also it's important to state that web browsers such as *Mozilla Firefox* don't perform well with large amount of SVG elements.

HTML5 canvas is able to draw large number of elements and updates the relevant information about those elements in short amount of time and it's supported in almost all web browsers used today.

IV. Drawing Implementation

Before introducing the functionalities available in *Leaflet*, the structure of the page must be defined. All the necessary libraries, basic HTML structure, styles and *JavaScript* functions needed for the map drawing must be included.

After the main page structure is created, a layer that will represent the base on which all the elements will be drawn must be defined. As this application is based on *OpenStreetMap* format, a provider must be found that will on request return an image tile containing map base layer. Requested layer is made of image tiles with the resolution of 256x256 pixels that are merged inside the browser to create a map. This application used *MapBox* service provider, but the choice of the provider is entirely left up to the user.

Layer types that the application will use should be chosen based on the user's needs and preferences. Besides the basic layers that display the vector data for the buildings and streets on the map, it is also possible to choose the layer for night time visibility, some geographic information such as rivers, mountains etc.

There are two types of elements chosen to be displayed onto the map: substation elements that represent medium voltage transformer stations and electrical lines.

Before the actual drawing of the element itself, all the relevant information must be loaded from the XML file, most important of which is the location. Application reads the information from the XML file, creates arrays of all the elements contained inside and then passes these arrays to the functions that are responsible for the drawing.

Function responsible for the substation element drawing as the only parameter takes an array of all the elements of substation type and draws them onto the map. For each element a conversion from UTM to latitude/longitude coordinates takes place. A blue circle marker is chosen as the shape that will represent a single substation. After the marker has been created it is added onto the map.

For a small number of elements which have large enough distance from each other, this way of drawing is quite good. But with the increase of the number of elements that are displayed at once onto the map, the performances start to decline. The solution is to clusterize (group) the elements of the same type, in this case the substations. Criteria for the grouping is the location of the elements on the map [6]. If the elements are close enough and zoom level is high enough, they will be clustered. As the zoom level gets lower, the elements will de-cluster. After positioning the mouse cursor over one of the clusters, the application will mark the surface it takes on the map. Picture 3 shows substation elements added onto the map and their clusterization.

Picture 3. *Substation elements drawn on the map*

Electrical line elements contain more information than the substations. One line is consisted of more connected segments. Also the difference between overhead and cable lines must be displayed. This difference is shown in different line coloring.

Algorithm used for the drawing of electrical lines is different from the one used for substations. Every point of a single line segment must be added to an array that creates

a polyline object that represents line segments. Before the drawing, a check must be made to see if the type of the line matches applied coloring. A legend is added to the bottom right side of the application to connect the line type with the used color. After this object is created, it needs to be added to an array that contains all the polylines that will eventually be drawn on the map. These actions are done for all the segments in iterations.

V. Element Manipulation

As frequent change of values describing the system are expected, these changes must be visible on the map as soon as the information of the change is received.

This means that the application should only update the data on the map and not redraw the entire view with every new change. So with periodic request for change updates, only those element that were affected by the change will be updated. This type of information can be sent in any format readable for the web browser such as XML or JSON format. With the decrease of sent information, so does the size of the data sent to the client decreases while in the same time updates are being applied faster. Global performance of the application is increased as less data is needed for the update.

Server processes are simulated on the client side with a random updates of the values contained in the substation elements. Each of the elements has a unique identifier. With the form available for the element update, the user can change values of a substation with the appropriate identifier. Element changes the color to better illustrate the speed required for the update operation.

Leaflet library doesn't contain the possibility for the map to run in full screen mode. That's why a plugin that enables this is added to the application. Simple click on the control opens the map in full screen.

Surface that is covered by the electrical grid is usually equal if not larger than the surface of the city itself. On the lower zoom levels user is not aware of the grid size and this is why a control is created to set the zoom level on the optimum level.

With the increase of element types displayed on the map, a user must be able to choose which elements should be visible and which should not. The map layer itself carries information such as street and building names, different colorings of those objects etc. When the information about the electrical grid is added on top of the map, the view becomes overpopulated and unreadable.

This is why the user is allowed to show or hide the available element types that are also grouped into layers. But also an option of alternative map base layer should be available. A functionality that enables a simple layer switch is implemented. Picture 4 shows the map view with implemented plugins and controls.

Picture 4. *Map with available layers and added controls*

VI. Result Comparison

For the testing purposes a plugin that measures frame rate of the rendered view is included in the application. Measured values are recorded for two scenarios. In Scenario 1 there is no value updates, while in Scenario 2, 5 random elements are updated on each 5 seconds. Values shown in the table represent average frame rate per second (fps) in the browser in which the application was tested. Measurement results are shown in the Table 1.

Web browser	*Scenario 1*	*Scenario 2*
Google Chrome 33	*54 fps*	*43 fps*
Mozilla Firefox 27	*32 fps*	*21 fps*
Safari 5.1.7	*36 fps*	*22 fps*

Table 1. *Measurement results*

VII. Conclusion

Although better performances can be acheved using hardware accelerated drawing, this type of client side drawing can take the load off the server and use the all the potential of modern web browsers to acheve more than satisfying rendering speeds at low cost. Also with this kind of implementation the user can choose any device that has a web browser and not worry about the compatibility of the software with the used device.

Besides the shown use in displaying the electrical grid of the city, this application can be used to display telecomunication, water or gas infrastructure status in real time in the web browser.

References

[1] W. Zenghong, "An emerging trend of GIS interaction development: Multi-touch GIS", Information Science and Control Engineering 2012 (ICISCE 2012)

[2] OpenStreetMap's community, "OpenStreetMap". Available: www.openstreetmap.org/about

[3] MapBox company, "MapBox maps". Available: www.mapbox.com

[4] Vladimir Agafonkin, "Leaflet - a JavaScript library for mobile-friendly maps". Available: www.leafletjs.com

[5] W3Schools, "Browser Statistics", Available: www.w3schools.com/browsers/browsers_stats.asp

[6] H. Yan and R. Weibel, An algorithm for point based cluster generalization based on the Voronoi diagram, Computers and Geosciences 34, 2008