**(Abstract**

Keywords: *Webmap application, Client-side, rendering, OpenLayers*

By 2020, many map visualizing/editing web applications have been developed. Google Maps, Bing Maps, just to mention a few from the most famous ones. If we do not calculate with the mobile applications, we open, use and edit these with web browsers, like Google Chrome or FireFox. Despite the fact that the browsers satisfy user demands equally, it’s well known, that the most popular browsers use different engines to compile the client-side web code, therefore the image rendering processes can represent different outputs. My research’ aim is to examine that by developing an own client-side web map application, and measuring the differences in visualization between the most frequently used browser engines with an image editing software. For programming the web map, I will use the )

**2. Introduction**

As, as of January, 2020, 4.54 billion people were active internet users (Simon Kemp, 2020, January 30) and owns devices to connect to it (computer, smartphone, for instance), the frequency of using web mapping applications increases visibly. The geoinformatics, as discipline, appeared in the 60’s, and is still a powerful conglomerate of informatic technologies with the main aim of decision preparation. The most popular geoinformatics-specialized companies’ (GDI-ESRI, for instance) desktop softwares are still powerful tools, which help us create, edit maps and store data within, but to share a complete map via internet, the best way is to develop web mapping applications. As every main browser product uses different methods to render image elements, we have to consider, that, in the output, measurable differences might appear.

During my Bachelor studies, I got to know the basics of geoinformatics, and since that I acquired the basic skills in web programming, and understood the basics of how internet works. My advisor’s, Gábor Farkas’s book (*Mastering OpenLayers 3- 2016 January*) showed me a complete way of how to build a client-side web map application, and can be related as a prelude of my current research.

As no one has done a complete research from this aspect, mine will be innovative, and will provide edification for web map developers in the future.

Hereby I would like to express my gratefulness for ….

**3. Aim of the research**

First, I will provide a detailed, professional way of building a basic client-side web map application with the guide of the OpenLayers API, using the client-side technologies (HTML for marking up, CSS for creating design rules, and JavaScript for interaction). To build the server-side, I will use XAMPP, as it’s a completely free, easy to install Apache distribution (*Retrieved from https://www.apachefriends.org/index.html*). After I customized the web map by adding basic controllers on it with JavaScript, I will also add an instance of every type of layer (vector – point, line, polygon -, raster). When it’s done, I will export the visualized map, and with an open-source image editor (GIMP) I will examine the differences.

According to my current knowledge, and experience in building client-side web maps and applications, I assume, the difference in the exported images – which were rendered with different engines – will be not relevant, but will exist, and if will so, my research will give an addition to both geoinformatics and web developers, with which, they can calculate in the future.

The side-aim of my research is to reflect on the relevance of using vectorgraphic map elements.

**4. Review** **of studies**

* 1. **History and basic operation of the Internet**

Internet is a connection of networks. It keeps devices connected to each other, lets us share data, regardless of the physical distance.

Every computer that is connected to the Internet is part of a network. (*TYSON, J., 2004 December. Retrieved from http://www.armchairpatriot.com/How%20Stuff%20Works/How%20Internet%20InfrastruInfra.pdf*). These computers, as connecting devices, have a unique ID, which we call IP (*Internet Protocol*) address. These are normally expressed in decimal formats, to make it easier to remember for human, but computers use it in binary form (*TYSON, J., 2004*). Before the internet became a world-wide infrastructure, computers could only connect to each other by “dialing” the IP address. As the number of users connected to the Internet increased, they attached a simple text file to the IP address, which was maintained by the Network Information Center. The University of Wisconsin created the Domain Name System (DNS), which connects text names with IP addresses automatically (*TYSON, J., 2004*). When you want to visit a website, or send an e-mail, you use the domain name for it. The URL (Uniform Resource Locator) holds it, and you use the Internet’s DNS servers to translate the domain name to the machine-readable binary IP address (*TYSON, J., 2004*).

Every machine, which is part of the world-wide web, is either a client, or a server. As it can be seen in the name, server machines provide service (data from database, for example) for the client machines, which connect to them. When client machine wants to connect to a server, it sends a request towards the server machine via HTTP, which stands for HyperText Transfer Protocol. HTTP is an application-layer protocol, for transmitting hypermedia documents, such as HTML. It was designed for communication between browsers (*clients*) and web servers (*MDN Contributors, 2019 December 2*). After the server accepted the request, sends the required data package back to the client. If the server does not approve the request from the client, the browser receives an error status code (for instance: 404 (*Server can not find the requested source*), 302 (*Source has been moved to other URL*).

* 1. **Client-side web technologies**

If the request, sent by client, is accepted on server-side, the client receives the requested data package (*Figure 1*).

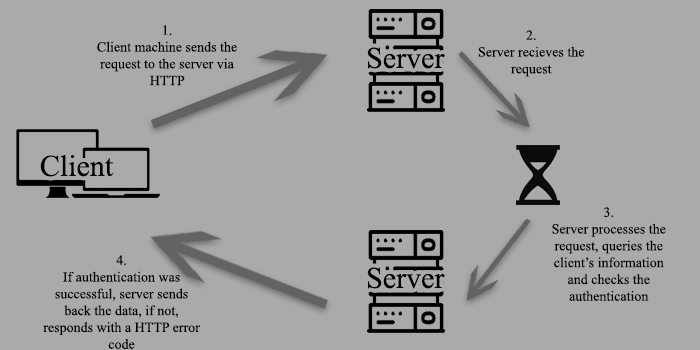


Figure 1: Client-Server communication (Bence Nagy)

Though, what client sees on his/her device, is a complete webpage; well-edited text fields, illustrated with images, containing links, for instance. But under the surface, it’s a bit more complicated. As the server has to store every data, what later becomes a spectacular web site, the safest and most practical way is to store them in code. When the server sends the requested data to the client, the actual web page (what the client sent a request for) is being transferred in code format. This pack of code is being processed and compiled by the web browser, which, after interpreting the code, shows the client the actual web site. In case of a complete web site, the code pack contains 3 type of files. These 3 types are HTML, CSS and JavaScript.

* + 1. **HTML**

HTML (HyperText Markup Language) is responsible for building blocks and elements of HTML pages. HTML’s history goes back to the early 90’s. It was first published as an Internet draft in 1993 (*Lubbers, P., Albers, B., Salim, F. - Pro HTML5 Programming, 2011*). The HTML5 (most recent version) specifications were written in 2004. The developers kept working on new features, which helped HTML5 apply to web applications. Now HTML5 is rapidly evolving to address real and practical improvements to the web platform (*Lubbers, P., Albers, B., Salim, F. - Pro HTML5 Programming, 2011*).

HTML builds the basic structure of a website by positioning its elements (images, paragraphs, headers, etc.). The browser knows from the header information, that the received file is an HTML file (every HTML file is named *filename.html*). As in most cases, server sends back a .php file (*filename.php*, PHP is a server-side programming language), in which the HTML file is included, we have to implement it between an opening and closing <html> </html> tag. Everything, what is written between these tags, will be interpreted by browser as HTML code.

Every element is written between the opening <tagname and the closing tagname/> tag (*Figure 2*), though there are some exceptions, which we call “self-closing elements”, such as <br />, which inserts a “*break*”.

<p id=”first\_paragraph”>This is a paragraph</p>

* <p> </p> is the tag name
* id=” ” is the identification
* between the tags, we can type the actual text (“*This is a paragraph*”, in this case), this is the *value* of the element

Figure2: an HTML element (Bence Nagy)

When the browser receives the HTML code, it interprets it sequentially, therefore, if no CSS (styling) rule is given for the elements, it puts them right below each other, in order of the declaration. From the tag name it knows, what kind of element does it deal with. If it’s a paragraph (seen in *Figure 1*), it simply prints out the value of it as a text. If it’s a link (written between <a> </a> tags, where “*a*” stands for “*anchor*”), the value of the element will be printed as an active link, on which the user can click, and if a “*href*” attribute is given, it will go on the given page:

<a href=”http://example.org”> This is the link text </a>

The other essential tools in HTML are the containers. The main aim of containers is to keep the elements together, which we want to handle together in the future. In the <head></head>block, we declare the file specifications, such as character set to use (<meta charset=”UTF-8”>), and also link the stylesheet (CSS), which gives the styling rules to our web page, and the script file (JS). This is a container, which can not be seen by the user. The default containers in HTML is <body> </body>. In the *body*, we put everything, we want the browser to visualize. Browsers use it as a special container, its position is tied to the browser window. The manual container declaration is done with the <div> </div> tag. We can put child elements in it (such as paragraphs, links, or even a child container (*div* in the *div*), and every rule, written on the parent element (the <div> in this case) will be automatically used on the child elements too.

The JavaScript and the CSS file can be included in the HTML 3 different ways:

1. External

This is the cleanest way to write a web code, and makes it easily readable and editable for other programmers.

The script file is inserted this way:

<script src="source/script.js"></script>,

where <script> is the tag name, src=”” is the source, for which the absolute path is given, as value (in this case, the script file’s path, but a URL link can be given as well).

The CSS file is inserted this way:

<link rel="stylesheet" href="style.css">,

where <link> defines a link to an external resource, rel=”” specifies the relationship between the HTML file and the linked CSS document, and for the href=”” we give the absolute path of the CSS file.

1. Internal

This way is often used, when a HTML page has a unique style and interaction. We write the CSS rules, and the script right into the HTML file. The advantage of this is that we only have to work with the HTML document, no need for other, external source files to be sent towards the client. The disadvantage is that it makes the code harder to be read.

The script file is written between the <script> </script> tags, and its content will be automatically interpreted by the browser as javascript code. For CSS it works the same way, the styling rules are written between the <style> </style> tags.

1. Inline

When the inline code insertion is being used, the styling rule and the script is written right in the HTML element. Inline code insertion was first used in HTML5.

With inline JavaScript code, a function can be added to a button element, for instance:

<button onclick=”function printLine(){alert(“Print this”)}; printLine()”></button>,

where we declare a <button> element, for which an onclick=”” attribute is given, whose value is the JavaScript code (in this case, a printLine() function is being declared, which will run, when the button is clicked).

In case of using inline styling rules, a style=”” attribute is given for the element, whose value is the CSS rule, i.e. this code:

<p style=”color: red;”>This is a paragraph </p>,

where “This is the paragraph” will be printed in red. Important to mention is, that the inline CSS code has the priority over internal/external CSS rules. If we link a CSS file, which uses a rule “color: red;” for a paragraph, and then we use an inline “style=”color: black;” rule in the HTML element, the paragraph’s color will be black.

* + 1. **CSS**

CSS (Cascading Style Sheet) is responsible for styling. If we try to build a web site with HTML only, the declared elements will be simply put right below each other. CSS is the language for describing the presentation of Web pages, including colours, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based mark-up language. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. (*W3C, 2016. Retrieved from https://www.w3.org/standards/webdesign/htmlcss#whatcss*).

A CSS rule contains a selector and the declaration, the declaration contains properties and values (*Figure 3*).

Figure 3: CSS rules (Bence Nagy)

#p1 {

background-color: red;

padding-top: 5px;

}

#header\_1, #header\_2, #title{

color: white;

margin: 10px;

}

* #p1, #header\_1, #header\_2 and #title are the selectors. The “#” sign marks the ID, which is given in the HTML
* “background-color:”,”padding-top:”,”color:” and “margin:” are the properties, these tell the browser, what to do with the selected element (i.e. color changes the text color, background-color changes the element’s background color)
* “red”, “5px”, “white”, “10px” are the values. We can give any value to a property, the only restriction is to give it the correct value type (color for background-color, px/cm/mm for font-size, etc.)

As shown in *Figure 3*, the same CSS rule can be written on multiple elements. The selector can have ID (with “#” mark) and class names (“.”) as well.

* + 1. **JavaScript**

JavaScript was developed by Brendan Eich, a developer at Netscape Communications Corporation, in 1995. The word “Java” in “JavaScript” is not coincidental. JavaScript has more in common with Self (a prototype-based language) and Scheme (a language, developed in the 1970s) than with Java (*Brown, E., Learning Javascript, 2016*). To have the name “JavaScript” was a marketing idea, as Java was one of, if not the most famous and frequently used programming language at that time.

JavaScript started its life as a browser-based client-side language used for adding interactivity to web pages, but it has evolved over the past few years. Examples of JavaScript applications include changing element focus, altering page and image loading behaviour, and interpreting mouse actions (*Grosskurth, A., Godfrey, M. W., 2006*). Later, with Node.js enabling JavaScript to be run on server-side, it opened up more and more possibilities (*The Ultimate Javascript Manual Third Edition, 2019*). JavaScript is an object-based programming language, that uses prototype objects to model inheritance (*Jensen, S.H., Møller, A., Thiemann, P., Type Analysis for JavaScript, 2009*). As it is a weakly typed language, unlike in some other Object Oriented Languages (C#, Java, for instance), the type of variables does not have to be declared, we simply mark them with the keywords var and let. The JavaScript code does not need to be compiled, it is only interpreted by the browser engine, therefore there is no need for IDE (Integrated Development Environment) to be installed, the browser contains it. Browsers use just-in-time compilation, which means, JavaScript code is compiled to bytecode right before it runs. Every modern browser has its own interpreter integrated (JavaScript engine), which helps us write or edit the JavaScript code via console (it can be opened with F12 key command). For proper code editing, any text editor can be used (even Notepad), though there are well-equipped editors, such as Visual Studio Code, Notepad++. These editors help in formatting the code to be well-readable, and highlight syntax errors, which could be seen only in the web browser console, right after running the code.

* 1. **Browsers**
     1. **History of browsers**

The web browser is perhaps the most widely used software application in history. It has evolved significantly over the past fifteen years; today, web browsers

run on diverse types of hardware, from cell phones and tablet PCs to desktop

computers (*Grosskurth, A., Godfrey, M. W., 2006*). By 1991, Tim Berners-Lee had written the first web browser, which could be used via GUI (Graphical User Interface), and was a basic HTML editor.

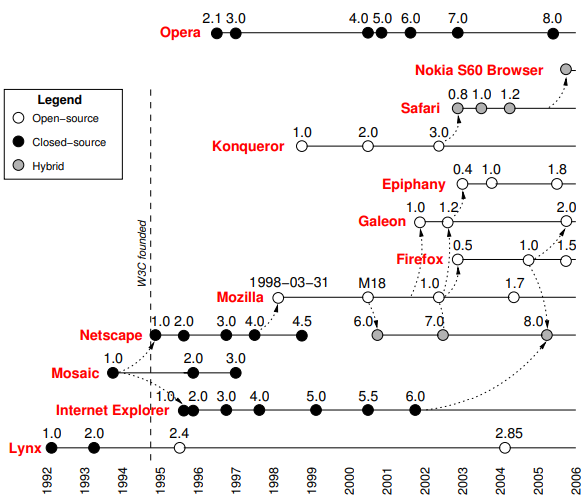


Figure 4: Web browser timeline (Grosskurth, A., Godfrey, M.W. – 2006)

Lynx (text-only hypertext browser) was released in 1993. In the same year, NCSA (National Center for Supercomputing Applications) released Mosaic, which was a graphical web browser. In 1994, Berners-Lee founded the World Wide Web Consortium (W3C) to guide the evolution of the web and promote interoperability among web technologies (*Grosskurth, A., Godfrey, M. W., 2006*). Mosaic’s lead-developer, Marc Andreesen formed his own company, Netscape in 1994, and, as Internet Explorer was released by Microsoft 1 year later, a marketing competition was formed between the two companies.

**Internet Explorer** is a series of graphical web browsers. The software is discontinued, but is still maintained. Its last version, IE10 was released in 2012. This is still the default browser for Windows 8 OS. Internet Explorer’s rendering engine is called Trident.

Netscape released its open-source browser, **Mozilla** in 1998. The first version of Firefox was released in 2003, which was a Mozilla variation. Firefox is a standalone browser with a streamlined user interface, eliminating Mozilla’s integrated mail, news, and chat client (*Grosskurth, A., Godfrey, M. W., 2006*). Firefox’s rendering engine is Gecko, which shows compatibility with IE’s Trident.

**Google Chrome** (released by Google LLC in 2008) is a cross platform web browser. First, it was released for Microsoft Windows operating system, later it was ported to Linux, macOS, iOS and to Android, too. Google used WebCore engine (widely known as WebKit), which was released by Apple in 2001. The WebKit project was, itself a fork of a rendering engine called KHTML. The project includes a core rendering engine for handling HTML and CSS (WebCore), a JavaScript engine (JavaScriptCore), and a high-level API for embedding it into browsers (WebKit) (*Bright, P., 2013*).

As the most widely used rendering engines are WebKit/Blink (Google Chrome, Chromium, Safari, Opera, Microsoft Edge), Gecko (Firefox) and Trident (Internet Explorer 10)(*Figure 5*), I will use Google Chrome, Firefox and Internet Explorer 10 to render web mapping images.

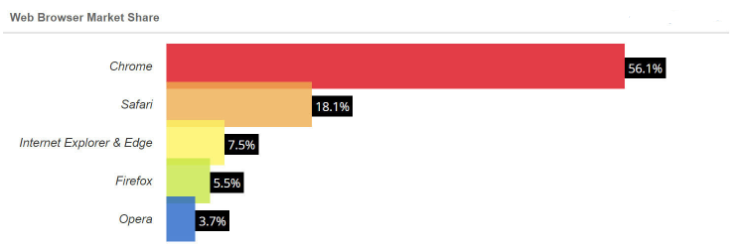


Figure 5: Web Browser Market Share (Martin, J., 2020)

* + 1. **Function of browser engines and hardware accelerating**

The browser engine works as a bridge between the user interface and the rendering engine. According to the inputs from various user interfaces, it queries and manipulates the rendering engine (*Raghuwanshi, M., 2017*). Rendering engine is responsible for visualizing the web page in the browser, the client sent the request for. The rendering engine interprets the HTML, XML document and the CSS rules, then parses the chunks of the HTML document and converts the elements to Document Object Model (DOM) nodes, which constitute the DOM tree (*Figure 6*).

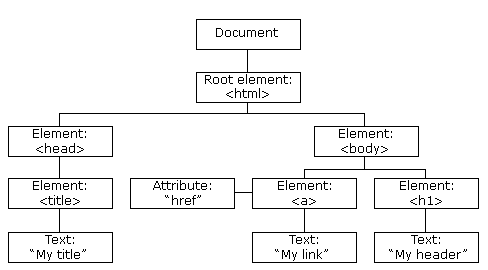


Figure 6: Visualization of the DOM tree (w3schools.com, 2020)

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order. Firefox calls the elements in the render tree “frames”. WebKit uses the term renderer or render object (*Raghuwanshi, M., 2017*). This is followed by the layout process. When the renderer tree is created, it has no dimensions and position. The position of the root renderer is 0, 0 (*Raghuwanshi, M., 2017*), which are the zero coordinates in the visible browser window, which is the actual coordinate system. Every child element is automatically being rendered to the 0, 0 point of its parent element.

The user demands for the speed and quality of visualization increase continuously for web mapping technologies. The first Web-based map viewers relied on early Web standards (*Farkas, G., 2019*), using DOM to visualize the elements. These applications later got evolved into dynamic web mapping libraries, which could create representation models based on raw vector data, images and image tiles (*Farkas, G., 2019*). Their rendering method did not change, and that gave a need for further development.

* + - 1. **Canvas**

The possibilities of native Web mapping libraries with next generation rendering engines came with the HTML5 standard’s <canvas> element and the API bound to it (*Farkas, G., 2019*). It has more visualizing tools than DOM-based rendering, and being run faster. It uses JavaScript, apart from the basic drawing methods, it is capable of creating animation as well. It was developed in 2004.

* + - 1. **WebGL**

WebGL is a subset of the Open Graphics Library (OpenGL) API, which allows developers to write programs executed on the GPU (*Farkas, G., 2019).* WebGL programs consist of control code written in JavaScript and shader code (GLSL) that is executed on a computer's Graphics Processing Unit (*MDN contributors, 2019*). It can be used to render 2D and 3D elements, too. The WebGL API was developed in 2011.

From the point of web mapping libraries, it does not matter, if a DOM, or Canvas-based renderer is used for visualization. If the web map has a larger dataset – which contains huge amount of data, it decreases the performance of the application. To accurately measure the benefits of a WebGL-based 2D map renderer over using the Canvas API, a capable library has to be chosen that has full Canvas support and can be extended with a WebGL rendering engine (*Farkas, G., 2019*). This is why I use the OpenLayers API, which is able to render images with Canvas, and WebGL renderer as well.

As web browser, I will use 3 browsers:

* Google Chrome: it uses Blink renderer, which is being used by Opera and Safari as well (they use WebKit, which is the successor of Blink)
* FireFox: it uses Gecko rendering engine
* Internet Explorer: as it is still a widely used browsing software, and Windows 8 OS still supports it, I feel the need to examine the rendering process on it

I will use WebGL and Canvas renderer for each browser, and will examine the outputs after.