UNIVERSITY OF TARTU

Faculty of Mathematics and Computer Science

Institute of Computer Science

Informatics

Artur Käpp

Comparison of JavaScript Graph Frameworks

Bachelor's Thesis

Supervisor: Margus Niitsoo, Ph.D.

Tartu 2014

JavaScripti graafide raamistike võrdlus

Keeruliste JavaScripti visualisatsioonide tegemine brauserites võib olla vägagi resurssikulukas. Antud töö võrdleb visualisatsioonide kõige algelisemat kuju – graafikud. Võrreldes nelja erinevat JavaScripti graafikute loomise raamistiku, saame vastused küsimustele, milline alus sobib kõige paremini Internetis graafikute loomiseks ning kuidas antud raamistikud üksteisest erinevad.

Võtmesõnad:

JavaScript, graafikud, visualiseerimine

Comparison of JavaScript Graph Frameworks

Creating JavaScript visualizations with large amount of data can cause big performance issues. Current thesis compares the most simplest form of data visualizations – graphs. By comparing four different JavaScript graphing frameworks, we analyze what is the best platform for rendering graphs on the Web and how the selected frameworks compare to one another.

Keywords:

JavaScript, graphs, rendering

Table of contents

Introduction 4
1 Overview 6
1.1 Data visualization 7
1.2 Rendering 7
1.2.1 Scalable Vector Graphics 8
1.2.2 HTML5 Canvas 9
1.2.3 Differences between Canvas and SVG 9
1.3 JavaScript 11
1.3.1 Short history of JavaScript 11
1.4 Methods for optimizing visualizations 12
1.4.1 Simplifying line graphs 12
1.4.2 Simplifying scatterplots 13
2 Comparing frameworks 14
2.1 Frameworks 14
2.1.1 D3 14
2.1.2 Highcharts 15
2.1.3 Flotr2 16
2.1.4 CanvasJS 16
2.2 Comparison methodology 17
2.2.1 Generating the data 17
2.2.2 Running the tests 19
2.2.3 Implementing frameworks 20
3 Results 22
3.1 Framework comparisons 22
3.1.1 Memory usage differences 22
3.1.2 Load time differences 24
3.2 Optimizations used in frameworks 26
3.3 SVG or Canvas? 26
Conclusion 27
References 28

Introduction

People have high expectations for web performance. End-users expect fast, engaging and interactive web experiences. Because of this demand, websites often use client-side programming.

Processing high volumes of data and making rich, interactive and mobile-friendly visualizations using just client side programming might sound impossible at first. However, by choosing the best data structures, algorithms and methods to optimize the visualizations, it might be possible to achieve this after all.

The simplest form of data visualizations are graphs. Visualizing information can give a quick overview of the data. Many websites use this to their advantage. To make these graphs interactive and process dynamic data, JavaScript is most often used. Just a plain image does not offer any interactivity or is not capable of changing itself with new data. The data amount given to these libraries might become very large in size. Therefore it is up to the used JavaScript framework to make the best out of a given situation and still perform well.

The aim of this thesis is to compare different JavaScript graphing libraries that are based on the HTML5 canvas element or Scalable Vector Graphics(“SVG”). Both are web technologies that allow users to create rich graphics inside the browser. This thesis gives some insight on how to choose the best JavaScript graphing library for the given situation. Whether the data is best rendered on a canvas element or SVG. What kind of graphing libraries are available and what might work the best?

The practical part of this thesis is to choose JavaScript graphing libraries with similar capabilities and to test how these frameworks react when the amount of data being shown increases. For this, their implementations, load times, memory usage and overall end-user experience are compared.

The thesis consists of 3 main chapters. The first chapter gives an overview of the technologies, optimization methods and the problem domain. The second

chapter deals with implementing the frameworks and setting up the testing environment. The final chapter analyzes the results and draws conclusions.

1 Overview

As of april 2014, the most popular client-side programming language, used by 87.7% of all the websites is JavaScript. [1] Next on the list is Flash, with being used by just 14.5% of all the websites. [2] On top of all that, the most common JavaScript technology used is jQuery which is used by 58.5% of all the webpages according to w3techs. [3] Recent studies also show that although JavaScript usage is in a slight decline, it is still far from extinction. [2] The author was not able to find any recent studies on how many users have JavaScript disabled. Although as of 2010, a research [4] by Yahoo, the percentage of these users was near 1%. The author believes this was due to fact that during that time JavaScript was considered by some to mostly be a security vulnerability and mostly used to display popups and alerts. Currently the percentage is probably even lower.

With JavaScript disabled many sites today become unusable [5]. For example trying to book a flight or a hotel on Hipmunk, the response is a plain text format of a JavaScript object. News feed in the most popular social networking site [6] Facebook is empty and it’s not even possible to send messages. The most popular online map service Google Maps does not display anything without JavaScript. Because of the high usage of JavaScript, it is becoming a norm to keep it enabled. For example the second most popular browser [7], Firefox, has removed the front-end option to disable JavaScript since Firefox 23.0 [8] - released in August 2013 [9].

Secondly, there is a rapidly increasing number of people who use a smartphone or a tablet to interact with web applications [10]. These end-users still expect a working site with reasonable interactivity and response times. Because of the rise in this trend, every website should consider making their site perform well on all kinds of devices. However, it is often a real challenge to make a rich and interactive webpage work smoothly both on a high powered desktop computer and also on a much less powerful mobile device.

Thirdly, the responsibility for processing data is shifting from the server to the browser [11]. Many Asynchronous Javascript (“Ajax”) based applications send

huge quantities of data between the client and the server - for example most of the online map services. In addition many websites tend to update Document Object Model (“DOM”) on the end-users side, which is a particularly time-consuming process. Poorly written DOM interactions can make the web application throw script alerts and in the worst case even make the browser go unresponsive.

Although Javascript calculations are fast and DOM interactions are slow, having a high volume of data and rendering together generally causes big performance issues [11]. It must also be taken into account that all devices have different hardware configurations and computing power. Different browsers have their own strengths and weaknesses. How to create a website that is rich, capable of processing lots of data within a reasonable time frame, capable of rendering the data to the end-user and all this without freezing the browser?

1.1 Data visualization

Data visualization is any kind of information represented in a graphical manner. Visual representations are often user to support and strengthen numerical data. Their main goal is to pass on information effectively and communicate a message more clearly.

By visualizing the numbers it is possible to effortlessly grasp the data. It becomes much easier to see patterns and make connections that really matter.

There is a variety of conventional ways to visualize data - tables, line charts, histograms, scatterplots and pie charts to name just a few. However to convey a message more effectively to the end-user, sometimes a simple pie chart may not suffice. Eventually the possibilities to visualize the data are endless and everything is up for the mind to create.

1.2 Rendering

There are mainly two ways to render content on a browser: Scalable Vector Graphics (“SVG”) and HTML5 canvas element.

1.2.1 Scalable Vector Graphics

According to the World Wide Web Consortium's (W3C) working draft of SVG 2 [12], SVG is a language for describing two-dimensional graphics in Extensible Markup Language (XML). SVG allows three types of graphic objects: vector graphic shapes, images and text. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively or via scripting.

All vector graphics shapes are made with geometrical primitives, such as points, lines and curves. More complex shapes can be achieved with polygons. What all of these shapes have in common is that all of them can be represented with mathematical formulas. For every point in these shapes it is possible to find corresponding x and y values.

According to the W3C's draft of SVG2 [12], SVG images are scalable. To be scalable means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size. Same quality can be achieved on different resolutions, so that the image can be used on every possible resolution with any size of screen, without losing quality. These images can be magnified to see fine detail, or to aid those with low vision. For comparison, all browser rendered text is typically TrueType [13], which means that these are also vector graphics. All the characters of a TrueType fonts are created from lines and bezier curves. It is possible to increase and decrease the font size, without losing much in quality.

SVG can also integrate raster images and can combine them with vector information such as clipping paths to produce a complete illustration [12]. Since basic image files are raster-only formats (such as PNG and JPG) which have to store information for every single pixel, the quality of the image goes down drastically when zooming in, as the pixel squares just get progressively bigger. There is a great loss in quality when magnifying the rasterized images up to the very pixels.

In the early days of the Web, it was clear that a vector graphics format for the Web would be useful [14]. SVG has been in open development since the end of 1999 [15] and is now implemented natively in most of the popular browsers [14].

Since SVG works within the XML environment, any kind of text editor can be used to modify or create SVG objects. Usually a more convenient vector graphic software is used to have a better overview of the image being created and to effortlessly modify it.

1.2.2 HTML5 Canvas

The Web Hypertext Application Technology Working Group ("WHATWG") defines HTML 5 canvas as a "resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art or other visual images on the fly" [16]. HTML5 canvas gives an easy and powerful way to draw graphics using JavaScript [17].

Canvas element is simply a DOM element, although the elements drawn on the surface of one are not accesible to the DOM, since Canvas works in immediate mode. Canvas doesn't have its own objects, only instructions on what to draw on any single frame [18].

Immediate mode means that the Canvas itself does not keep track of the objects drawn on it. Although the shape is visible, once it is drawn, it is not possible to manipulate the shape individually. This makes it similar in behavior to a painting canvas, since once something is painted, all that is left are color pigments. As for the HTML5 Canvas all that are left are pixels. To change the shapes or the image it is possible to draw over the objects or to start fresh.

1.2.3 Differences between Canvas and SVG

Graphics APIs can be divided into retained-mode APIs and immediate-mode APIs [19]. HTML5 Canvas element is an example of an immediate-mode API and SVG is an example of a retained-mode API.

SVG works in retained mode. A retained-mode API is declarative. The scene is constructed from graphics primitives, such as different polygons and lines.

Pretty much everything can be represented with mathematical formulas or coordinates.

The graphics library stores a model of the scene in memory. To draw a frame the graphics library transforms the scene into a set of drawing commands [19]. To modify a given scene, for example to add or remove a shape, the application issues a command to update the scene. Then the graphics library is responsible for redrawing the scene. This keeps the programmer away from any low-level operations but also gives less control over the final rendering [18].

As opposed to the SVG, HTML5 Canvas works in immediate mode. An immediate-mode API is procedural [19]. Every time a new frame is drawn, the application directly issues the drawing commands. The graphics library doesn’t keep track of the scene between frames. Instead the application only keeps track of the scene.

According to Brad Neuberg, former software engineer at Google and also a developer advocate at Google for HTML5 [20], SVG is higher level. It is possible to effortlessly import and export SVG images between different softwares. With SVG it is easier to create animations and user interfaces. Because SVG is a retained-mode API, it is possible to just add event handlers for objects in the DOM, which makes it very easy to add interactivity to the image [21].

Canvas is lower level compared to SVG. Depending on the needs of the project, that can be a strength or a weakness. Canvas is good when there is no need for the DOM tree or mouse interactions. When there is a high level of animations without interactivity, canvas is a very good choice - the DOM tree would just get in the way and make animations slower. Canvas is JavaScript-centric and needs more tracking of the object states than SVG. That, again, can be both a strength or a weakness depending on the application [21].

Both SVG [22] and HTML5 Canvas [23] are supported in all open-source modern browsers (Chrome, Firefox, Safari, Opera). Both are also supported in latest versions of Internet Explorer, since version 9.0. Additionally, both technologies are supported with the most popular mobile browsers.

1.3 JavaScript

JavaScript is a lightweight, interpreted programming language, which was designed for creating network-centric applications. JavaScript is Open Source and cross-platform. Currently JavaScript is the most popular programming language in the world [24].

There are three major types of JavaScript [25]:

- **Core JavaScript** - the base JavaScript language
- **Client-Side JavaScript** - an extended version of JavaScript that enables the enhancement and manipulation of web pages and client browsers
- **Server-Side JavaScript** - an extended version of JavaScript that enables back-end access to databases, file systems, and servers.

Client-side JavaScript and server-side JavaScript are dependent on the core JavaScript and can not work without it.

1.3.1 Short history of JavaScript [26]

JavaScript was created in 10 days in May 1995 by Brendan Eich who was then working at Netscape. The original name was Mocha, but in September of 1995 the name was changed to LiveScript - by Netscape founder Marc Andreessen. In December the same year the name JavaScript was adopted - the change to JavaScript may have been a marketing move due to the fact that Java was being very popular around that time.

During 1996 - 1997 JavaScript was taken to ECMA to give it a standard specification, so that other browser vendors could implement it to work with other browsers. The standardized version was named ECMAScript, with JavaScript being the most well known implementation of it. However, it is not the only one, as ActionScript 3, which is the basis of the Flash platform, also complies with the standard.

The standards process continued in cycles, with the releases ECMAScript 2 in 1998 and ECMAScript 3 in 1999, the last one being the basis for modern JavaScript. In 2000 even Microsoft stepped in (with work led by Waldemar

Horwat) and implemented some of the proposals in their JScript.net language. Later it became clear that Microsoft had no intention of cooperating or using proper JS in their Internet Explorer (even though they had no competing alternative for it - Microsoft had a partial implementation on the .NET server side). So in 2003 the JS2/original-ES4 was disbanded.

In 2005 the open source and developer communities set to work to revolutionize what could be done with JavaScript. Jesse James Garrett led the community effort and released a paper in which he coined the term Ajax. He described a set of technologies, of which JavaScript was the backbone, used to create web applications where data could be loaded in the background, avoiding the need for full page reloads and resulting in more dynamic applications. This resulted in a renaissance period of JavaScript usage spearheaded by open source libraries and the communities that formed around them, jQuery being one of them.

1.4 Methods for optimizing visualizations

JavaScript optimization can become very important when doing heavy visualizations. The author believes that in line charts and scatterplots the libraries will have at least one of the following optimization methods implemented.

1.4.1 Simplifying line graphs

Polyline simplification is the process of reducing the resolution of a polyline. Since all the displays have a fixed, although different, resolution, displaying multiple points very close to one another would simply not be visible to the eye therefore it would be a waste of resources. Polyline simplification removes the unnecessary points while still maintaining a good approximation of the original curve. These algorithms seem like most suitable for a line graph with a very large dataset.

Distance-based simplification [27] takes all the data points and marks the first point of the line as a key - element to be kept after the simplification. All consecutive points which lie in a specific distance tolerance from the key are removed. The first encountered vertex, that lies further away than tolerance is

marked as the new key and the same process is repeated. The first and last points are always marked as keys, despite their distance from other points.

Douglas-Peucker algorithm [27] uses a point-to-edge tolerance. It starts with a crude simplification that is a single edge joining the first and the last vertex of the original polyline. It then calculates all the intermediate vertices distance to the edge. The furthest away vertex is marked as a new key. The crude line from the beginning now has three points. The process is repeated until all intermediate points have a distance within the defined tolerance to the edge. In case all the intermediate points fall into the specified tolerance the final result has been achieved. Once again, the first and final vertices are always keys and will not be removed.

1.4.2 Simplifying scatterplots

Clustering organizes things that are close or similar into groups. The author believes that this is an appropriate way to reduce clutter in scatter charts. For example hierarchical clustering algorithm can be used.

Hierarchical clustering algorithm [28] in scatterplot could start by finding the two points which have a least distance between them and forms them into one pseudo-element with an average position between them. This process is continued until only one point is left. The process forms a tree called dendrogram which shows the distances between the points as displayed on Figure 1.1. By cutting the dendrogram at any height horizontally would give a number of clusters to display. By this way it is possible to reduce the number of data points to display.

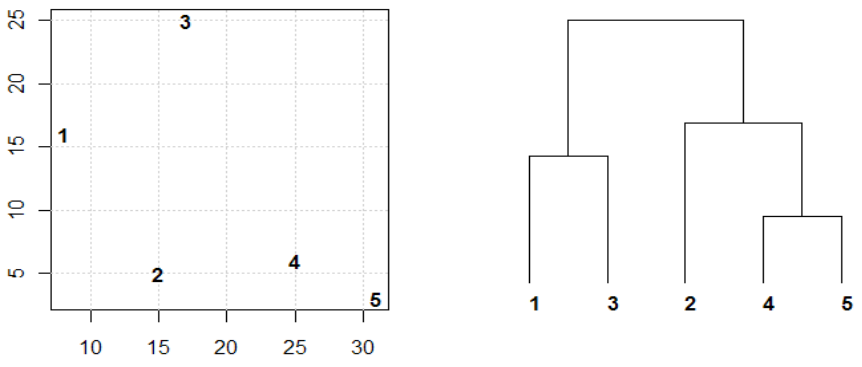


Figure 1.1 Example of a dendrogram

2 Comparing frameworks

Following frameworks were implemented and measured as part of this thesis: Data Driven Documents (D3), Highcharts, CanvasJS and Flotr2. Two of the most common types of charts [29] were compared: line chart and scatterplot.

2.1 Frameworks

When it comes to creating charts with JavaScript, there is a wide variety of libraries available. Current thesis focuses on four major libraries chosen by the author. The selection was made from a list of frameworks [30] bearing in mind that the chosen frameworks should have similar capabilities – the possibility to create a scatterplot and a line chart so that it would be possible to later compare the selections. Another key point in selecting frameworks was that it would be free to use to some degree.

2.1.1 D3

According to their official github wiki [31], D3.js is a JavaScript library for manipulating documents based on data. D3 helps to bring data to life using HTML, SVG and CSS. D3's emphasis on web standards gives the full capabilities of modern browsers without tying the programmer to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.

D3 stands for Data Driven Documents. The recommended use for D3 is when webpage interacts with data [32]. D3 allows to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, it is possible to use D3 to generate an HTML table from an array of numbers or to use the same data to create an interactive SVG bar chart with smooth transitions and interaction [33].

Perhaps the most important part of D3′s success is the position and approach it takes. It is not a graphics library, nor is it a data processing library. It does not have pre-built charts that limit creativity. Instead, it has tools that make the

connection between data and graphics easy. It sits right between the two, the perfect place for a library meant for data visualization [34].

D3 was chosen because the author feels that this library is with the widest variety of possibilities. D3 is not simply for creating basic charts. It is a framework on top of which it is possible to create even the most advanced visualizations with complex data sets.

2.1.2 Highcharts [35]

Highcharts is a charting library written in pure JavaScript which offers an easy way to add interactive charts to a website or a web application. Highcharts currently supports line, spline, area, areaspline, column, bar, pie, scatter, angular gauges, area range, areasplinerange, column range, bubble, box plot, error bars, funnel, waterfall and polar chart types.

Highcharts is solely based on native browser technologies and does not require client side plugins like Flash or Java. Furthermore it is not needed to install anything extra on the server. No PHP or ASP.NET. Highcharts needs only two JavaScript files to run: The highcharts.js core and either the jQuery, MooTools or Prototype framework. One of these frameworks is most likely already in use in the web page.

Setting the Highcharts configuration options requires no special programming skills. The options are given in a JavaScript Object Notation (JSON) structure, which is basically a set of keys and values connected by colons, separated by commas and grouped by curly brackets.

It works in all modern mobile and desktop browsers including the iPhone/iPad and Internet Explorer from version 6. On iOS and Android, multitouch support provides a seamless user experience. Standard browsers use SVG for the graphics rendering. In legacy Internet Explorer graphics are drawn using Vector Markup Language (VML).

Highcharts has a huge range of chart options available. It is easily customizable with great documentation [36]. Highcharts was the second framework in terms

of user likes in the Socialcompare framework comparison [30] with D3 being the first.

Highcharts was chosen by the author as the second SVG library for comparison with D3.

2.1.3 Flotr2

Flotr2 is a canvas library for drawing HTML5 charts and graphs. It is the improved version of Flotr which removes the Prototype dependency and includes many other improvements [37].

Flotr2 has a possibility to extend its capabilities with new graph types and plugins [37].

As described by Carl Sutherland, the creator of Flotr2, the main advantage of Flotr2 is speed because it uses canvas instead of SVG. Therefore it has an advantage particularly on mobile systems. Flotr2 also has touch event support. Although it is not a very flexible API nor great at creating new or abstract data visualizations, it is easy for creating simple graphs [38].

The biggest problem currently with this framework is that it has a legacy code base, supporting old browsers. Although supporting old browsers in a way is a positive thing, it does have a downside. Since it is inherited from the project Flotr, the creator thinks that it really needs a ground-up rewrite, which he claims would love to do if he ever gets time [38].

The author of this thesis believes that Flotr2 is a simple way of creating visualizations. It takes away the need to over-engineer for a simple graph. This framework was chosen for its simplicity [39].

2.1.4 CanvasJS

According to CanvasJS homepage [40], CanvasJS is an easy to use HTML5 & JavaScript Charting library built on Canvas element. Graphs can render across devices including iPhone, iPad, Android, Windows Phone, Desktops, etc. This allows you to create rich dashboards that work on all devices without compromising on maintainability or functionality of your web application. Charts

include several good-looking themes and claims to be over 10x faster than conventional Flash and SVG Charts – supposedly resulting in lightweight, beautiful and responsive dashboards.

The author chose this library as the second HTML5 canvas library to contrast against Flotr2. This library caught the author's attention by its promise on their official website [40]: "CanvasJS can render 100 000 Data-Points in just around 100 milliseconds".

2.2 Comparison methodology

By testing these frameworks, the author wanted to find differences between the SVG based and HTML5 Canvas based graphing libraries. By choosing the data to measure, the author tried to select the data which would matter the most to the end-user - perceived loading time and memory usage (memory usage would affect the end-users browser performance). In case the selected libraries have some huge memory leaks, the slow browser performance would become noticeable.

Another point of interest was optimization methods used in the selected libraries. The author believes that data optimization can become very important with high volumes of data, especially in SVG based libraries. Since all SVG objects add up to the DOM size, the browsers performance can suffer greatly if too many elements are added.

2.2.1 Generating the data

The code implementations were mostly done based on the example implementations on each of the framework websites. Data was generated randomly in the following amounts for line chart: 10, 100, 1 000, 10 000, 100 000, 500 000, 1 000 000 and 5 000 000. Line chart data was generated by setting a random number corresponding to each day. To reduce the massive random value jumps between two points, every next point was +/- 12 from the previous as shown in Figure 2.1. Initial point had a value of 150. For every data amount 10 different datasets were generated using PHP.

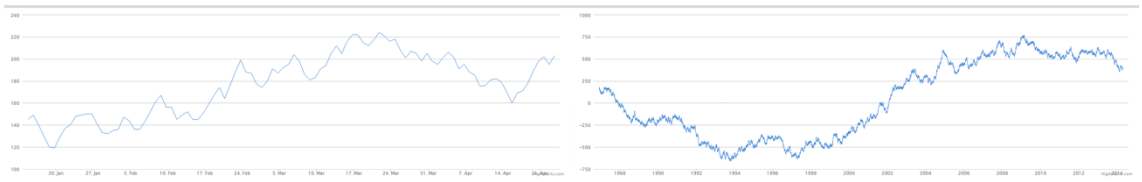


Figure 2.1 Example of a line chart used in tests

Data for scatterplots was generated randomly in the following amounts: 10, 100, 1 000, 10 000, 100 000, 500 000. The data was created by generating a random x value from 1 to 10 000 and calculating the y value from function x * 0.75. Y values were then randomly modified as follows:

- 5% chance of the value was modified to be positioned anywhere on the y axis, between minimum (yMin) and maximum (yMax) possible value
- 5% chance of the value being between [0.9 * yMin and 0.9 * yMax]
- 5% chance of the value being between [0.8 * yMin, 0.8 * yMax]
- 5% chance of the value being between [0.7 * yMin, 0.7 * yMax]
- 15% chance of the value being between [0.6 * yMin, 0.6 * yMax]
- 25% chance of the value being between [0.5 * yMin, 0.5 * yMax]
- 30% chance of the value being between [0.3 * yMin, 0.3 * yMin]
- 10% chance of the value being between [0. 1 * yMin, 0.1 * yMax]

The percentages were generated so that the scatterplot would seem a bit more realistic. Therefore, the scatterplot data was generated with a higher chance of the data points being near the regression line and less chance of it being further away as can be seen from Figure 2.2. For every data amount, 10 different datasets were again generated.

Figure 2.2 Example of a scatterplot with a regression line

To reduce overhead and get more accurate results between different frameworks, all generated data was stored in files, with JavaScript Object Notation format. Every library used the exact same data.

2.2.2 Running the tests

All the measurements were done in Google Chrome browser's incognito mode to reduce data caching and interference of plugins. Google Chrome was chosen because W3Schools browser statistics [41] show that as of April 2014, this is the most popular browser.

Secondly, after the browser was launched, an additional 30 seconds was waited so that the browser can finish with all the additional background actions including updates check. No other websites were open during the testing phase and the computer was kept at reasonable amount of background threads, meaning there was no heavy CPU usage.

All the tests were run on a Asus laptop with i7-3517U processor running at 1.90GHz up to 2.4GHz. The laptop had 10GB of RAM and was running on a 64-bit Windows 7 operating system. The harddrive was a 120GB Solid-State Drive (SSD).

The following data was measured:

- Time from the response end until the page was fully loaded - users perceived page load time (Page load)
- Time from the beginning of the visualization function until the end in JavaScript (Measured time)
- Total time displayed for the initial function of the framework, as displayed in Chrome profiler tools (Function time)
- Browser's memory usage before
- Browser's memory usage after

Page load time was measured using Web Navigation Timing API [42]. The time was calculated from the response end [43] until the end of load event. Response end time attribute is the time when the user agent receives the last byte of the document or before the transport connection is closed, whichever

comes first. Response end was selected instead of the navigation start [44], which displays the time of the unloading of the previous document, because the author feels that this is better suited for the tests. The scripts can launch only if all the scripts have been fully loaded into the user agent. So the time spent before that by the browser is irrelevant.

The second variable, measured time, was the time after the data had been converted in the suitable format for the library until the end of the drawing function. All in all this time should always be with a similar magnitude as the function time displayed in the Chrome's profiler tools.

The third time was taken from the Chrome's developer tools [45] by collecting the JavaScript CPU profile. The first function called from the framework's own script file was taken. This should be the initiation function which calls all the sub-functions.

Another value that was measured with the Chrome developer tools was webpage memory usage. After the function was finished and the graph was properly displayed, a heap snapshot was taken, which displays the memory allocated by the current page being analyzed.

The following versions of the frameworks were used:

- **jQuery** 1.11.0
- **D3** 3.4.3
- **Highcharts** 3.0.10
- **CanvasJS** 1.3.0
- **Flotr2** from Github repository as of January 21, 2014.

2.2.3 Implementing frameworks

All the tests were implemented by taking a base example from the framework's official webpage, which was modified accordingly to remove or disable all unnecessary features and to display the graph in a polite and minimal fashion. The base look of all the charts in the end was with x and y axes and with a blue line or dots accordingly. An example of this can be seen in Figure 2.3.

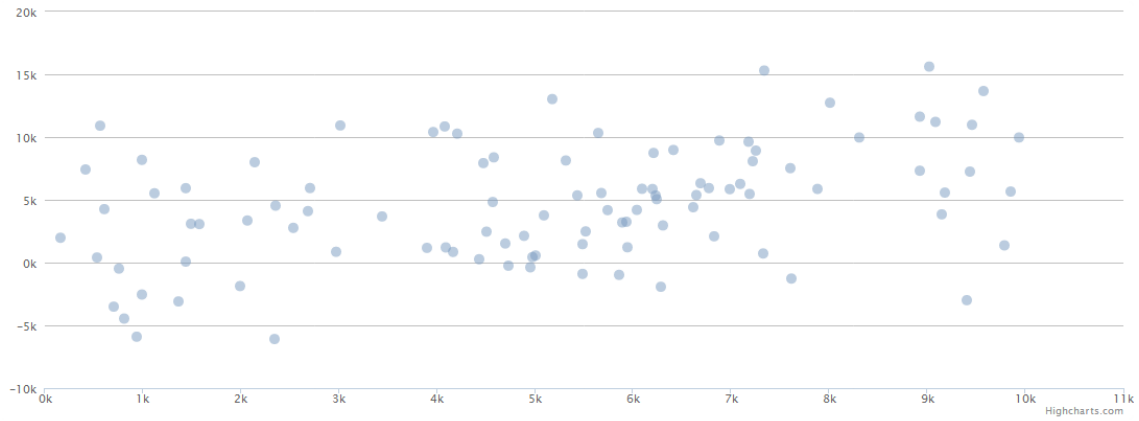


Figure 2.3 Example of a scatterplot

Author tried to make the graphs with minimalistic design so that the appearance would be fitting for the current trends on the Web. There is no need for the extra distractions on the chart and especially none for the current testing process. Although some charts had some limitations or not so logical ways of changing some of the visual attributes, the final goal was to make the charts quite similar, with good design. The few peculiarities of the frameworks should not have huge impact on the test results. The final appearance of the graphs can be found in Appendix 3.

3 Results

All the data for the line charts and the scatterplots after measurements can be found in the Appendix 4. Summary of the results can be found in the Appendices 1 and 2.

Final appearances of the implementations can be found in Appendix 3.

3.1 Framework comparisons

First of all, every single framework had possibilities to change the basic appearance of the chart: axes, width of lines/dots, grid, colors etc. No major shortfalls were noticed in this category by the Author.

At each number of data points, the differences between all pairs of frameworks were measured with a two-tailed distribution t-test and they were all significant of at least 1% confidence level, except for D3 and Flotr2 at 1 000 data points in the scatterplot. There were also a few occasions where t-test results for CanvasJS were not significant of at 1% level when compared to other frameworks.

3.1.1 Memory usage differences

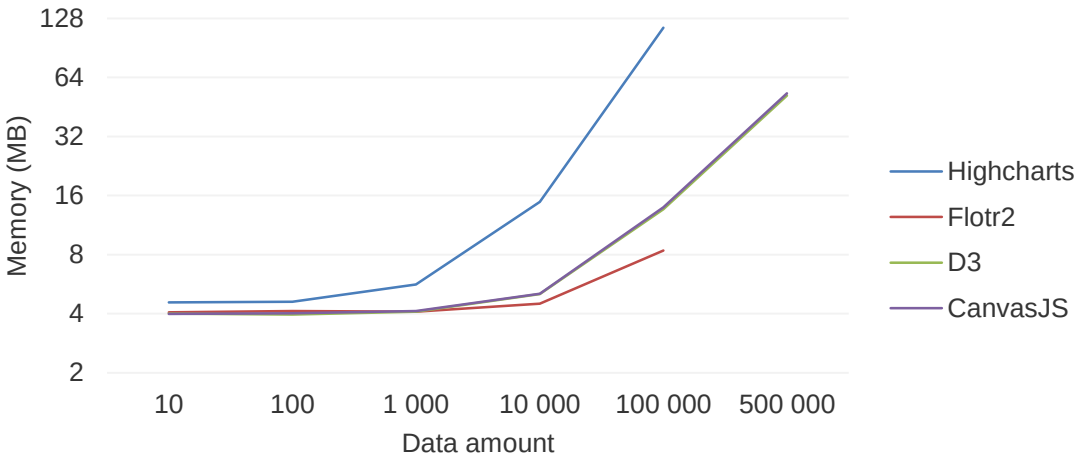


Figure 3.1 Scatterplot memory usage

There was a huge memory use difference between D3 and Highcharts when the number of data points became very large as can be seen from Figure 3.1. With 100 000 datapoints in the scatterplot, Highcharts memory usage was on average 114.7 MB, whereas D3's was 13.6 MB. An investigation into the cause led to an interesting difference. All the data points generated by D3's are SVG circle objects. Their position is marked with just x and y coordinates. On the other hand Highcharts data points are all SVG path objects. This means that just the data needed to store the information of the objects in DOM is already about twice as much as circle objects. The circle objects in D3 charts with current implementation take about 127 characters. Highcharts SVG paths take 241 characters.

Since this seemed a bit odd, a scatterplot demo [46] on their official website was checked again and the same behavior was observed there as well. This could very well be the main cause of the huge memory difference.

Another interesting observation is that CanvasJS and D3 have almost identical memory usages when rendering a scatterplot as shown in Figure 3.1.

The low amount of memory usage in line charts as can be seen from Figure 3.2 should not have a negative effect on the browser's performance. After 100 000 data points the memory usage stabilized for every library, however the author can not explain that. D3 needed the least amount of memory whereas Highcharts needed the most.

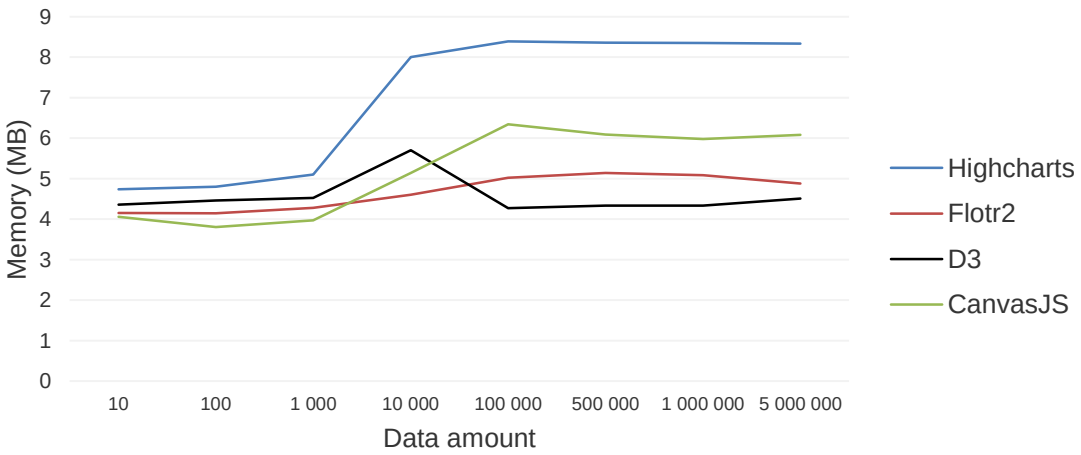


Figure 3.2 Line chart memory usage

3.1.2 Load time differences

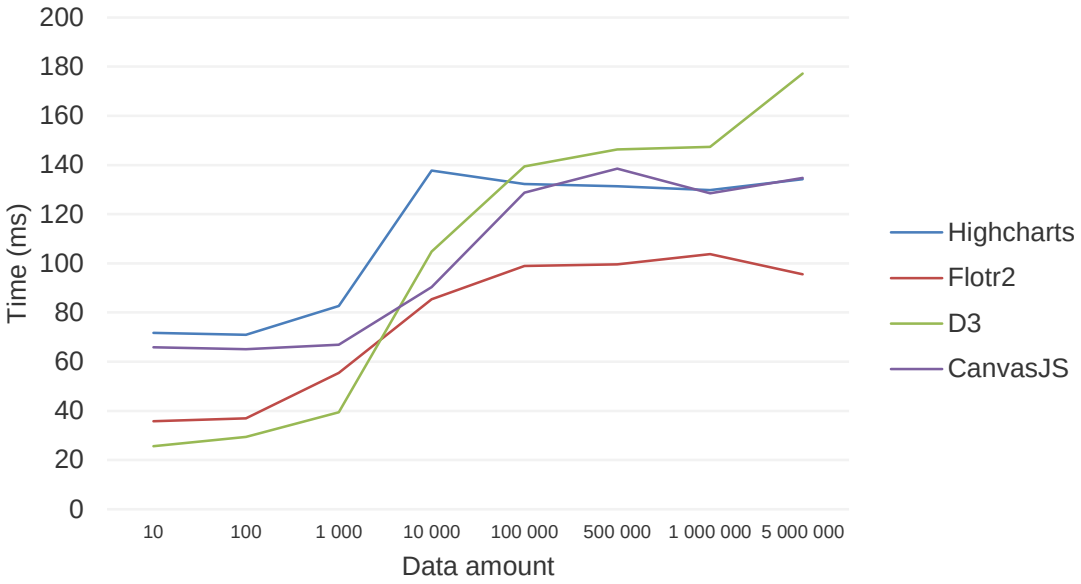


Figure 3.3 Line chart function times

The render times of CanvasJS and Flotr2 scatterplot were quite low up to 10 000 data points as can be seen from Figure 3.3. From there on CanvasJS performed better. With 100 000 data points the average rendering time for CanvasJS was 1 696 ms and the time for Flotr2 was 30 901 ms. As it later turned out the hardware acceleration had huge negative impact on Canvas based libraries.

The author would not call the computer that was used to run the tests average or below average. The settings for the Chrome browser were the default settings when the browser was installed. The default settings for the current versions of Chrome have hardware acceleration enabled by default. With hardware acceleration disabled the average time to render a scatterplot with 100 000 points decreased roughly 5 times for Canvas based frameworks.

As goes for the line chart with five million data points and hardware acceleration, the Flotr2 library performed 8 times worse, which is interesting, considering the behavior with scatterplot. CanvasJS, on the other hand, had a bit more positive effect when the hardware acceleration was disabled. CanvasJS gained an average of 37ms improvement in render times according to Chrome profiler.

In line chart implementations the SVG based libraries had an advantage with up to 1 000 data points. After that the Canvas libraries started to perform better, although the differences were not all that big. With 5 million data points the SVG libraries had an average rendering time of 156 ms, whereas the Canvas frameworks rendered the chart with just 115 ms.

The author believed there would be major performance problems when rendering line charts with SVG libraries. This was due to a misconception by the author that the DOM tree would get huge with the line chart when the data amount increases. What the author did not realise at the time was that drawing the line chart with SVG libraries is eventually just one SVG path with multiple coordinates to go through. Therefore, in the DOM there was only one path object representing the line, despite its size.

As can be seen from Figure 3.3 the rendering times for all the line graphs rose quite significantly between 1 000 and 10 000 data points. Although the rise is on average just 40 ms, it is still an interesting coincidence. The author has no idea what might have been the cause for this.

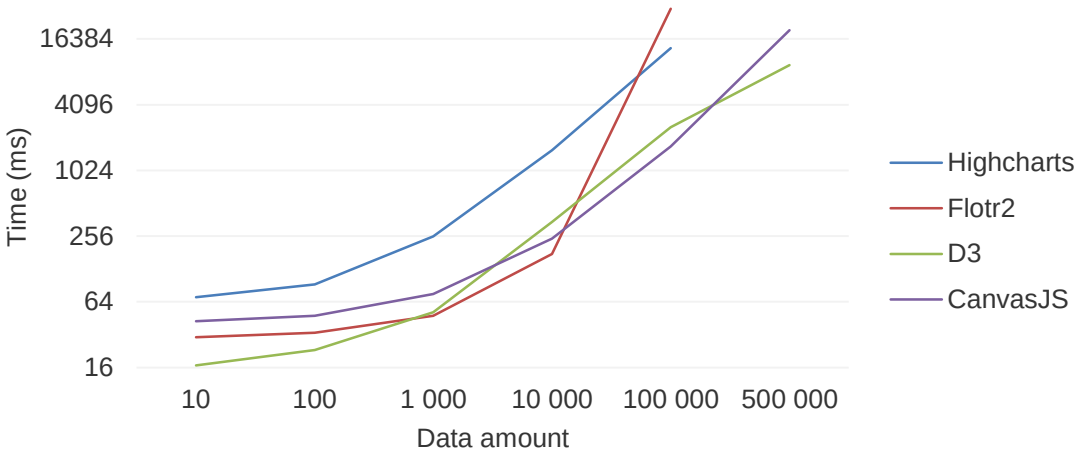


Figure 3.4 Scatterplot function time

The scatterplots results were surprising as the fastest library to render a scatterplot with 500 000 data points was D3 as can be seen from Figure 3.4. Because of the large number of objects in the DOM tree the author did not believe that an SVG library would have the best results. It is difficult to make

any meaningful comparisons because one SVG library and one Canvas library were not capable of rendering a scatterplot with 500 000 data points.

3.2 Optimizations used in frameworks

To the author's surprise no data optimizations were found in any of the libraries. All the data points were put on the chart as is. Without using even the most basic algorithms to simplify the polyline or to cluster the data points as described above.

Highchart suggests some optimizations to do on the server side before sending the data for display. Since the current thesis was testing the capabilities of just the JavaScript part of the frameworks, this was not an option.

Creator of Flotr2, Carl Sutherland, also noted [38] that no fancy optimizations are used in Flotr2 and that the library employs usual JavaScript microoptimization techniques and caching of the canvas element between redraws. The author believes similar optimizations are used in CanvasJS. But this does not improve the initial loading time of the charts.

3.3 SVG or Canvas?

Once again when choosing between SVG and Canvas, it really comes down to the goal of the visualization. If the visualization doesn't have many data points or has some user interactions, the SVG is the easiest option because event handling in SVG is much easier. Although everything that can be done with SVG can be done with Canvas as well.

Canvas performs generally better but once something is drawn on the canvas and the state changes, the whole canvas needs to be drawn again. Therefore, to change the current state of the Canvas, a lot more object tracking needs to be done and a lot more tracking means a lot more code. To generalize, the Canvas also seems to perform better when the number of objects goes past 1 000.

Conclusion

As tested by the author, rendering of very big amounts of data can lead to minutes of waiting. It is important to choose a suitable rendering method based on the needs of the project as poor choices can lead to poor performance.

For visualizations with interactions and where number of objects is less than 1 000 it is simpler to use SVG. Everything that can be accomplished with SVG can be done with Canvas as well, although for simpler visualizations the performance gain from Canvas might not be worth the coding effort.

For the most simplest graphs Flotr2 is the best choice, although Highcharts is also simple to implement with more customization options. For very creative visualizations, D3 is a suitable base framework.

The performance differences were not that significant for the line chart. Once the data points number increased to 100 000 in scatterplot, CanvasJS and D3 had the fastest rendering times. D3 performed the best when visualizing a large number of data points.

References

1 W3 Techs. Usage of JavaScript for websites. [Internet]. 2014 [cited 2014 May 06]. Available from: http://w3techs.com/technologies/details/cp-javascript/all/all.

2 W3 Techs. Historical trends in the usage of client-side programming languages for websites. [Internet]. 2014 [cited 2014 May 06]. Available from: http://w3techs.com/technologies/history_overview/client_side_language/all.

3 W3 Techs. Usage of JavaScript libraries for websites. [Internet]. 2014 [cited 2014 May 06]. Available from: http://w3techs.com/technologies/overview/javascript_library/all.

4 Yahoo! How many users have JavaScript disabled? [Internet]. 2014 [cited 2014 May 06]. Available from: https://developer.yahoo.com/blogs/ydn/many-users-javascript-disabled-14121.html.

5 Limi A. Checkboxes that kill your product. [Internet]. 2014 [cited 2014 May 06]. Available from: http://limi.net/checkboxes-that-kill/.

6 eBiz MBA. Top 15 Most Popular Social Networking Sites | May 2014. [Internet]. 2014 [cited 2014 May 06]. Available from: http://www.ebizmba.com/articles/social-networking-websites.

7 w3schools. Browser Statistics and Trends. [Internet]. 2014 [cited 2014 May 06]. Available from: http://www.w3schools.com/browsers/browsers_stats.asp.

8 Buckler C. Should Users be Permitted to Disable JavaScript? [Internet]. 2014 [cited 2014 May 06]. Available from: http://www.sitepoint.com/disable-javascript-option/.

9 Firefox. Firefox 23 Release Notes. [Internet]. 2014 [cited 2014 May 06]. Available from: https://www.mozilla.org/en-US/firefox/23.0/releasenotes/.

10 Bosomworth D. Statistics on mobile usage and adoption to inform your mobile marketing strategy. [Internet]. 2014 [cited 2014 May 09]. Available from: http://www.google.com/url?q=http%3A%2F%2Fwww.smartinsights.com%2Fmobile-marketing%2Fmobile-marketing-analytics%2Fmobile-marketing-statistics%2F&sa=D&sntz=1&usg=AFQjCNGfcK9hzsiAdi4K19yE7qFZYk73SA.

11 Agafonkin V. High Performance Data Visualizations. [Internet]. 2013 [cited 2014 May 09]. Available from: http://vimeo.com/68252990.

12 W3C. Working Draft of SVG 2. [Internet]. 2014 [cited 2014 May 09]. Available from: http://www.w3.org/TR/SVG2/intro.html.

13 How Stuff Works. What are TrueType fonts? [Internet]. 2014 [cited 2014 May 09]. Available from: http://computer.howstuffworks.com/question460.htm.

14 World Wide Web Consorteum. Secret Origin of SVG. [Internet]. 2010 [cited 2014 May 09]. Available from: http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG.

15 W3C. Scalable Vector Graphics (SVG) 1.0 Specification. [Internet]. 1999 [cited 2014 May 09]. Available from: http://www.w3.org/TR/1999/WD-SVG-19991203/.

16 WHATWG. The canvas element. [Internet]. 2014 [cited 2014 May 09]. Available from: http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element.

17 Sucan M. HTML5 Canvas - the Basics. [Internet]. 2009 [cited 2014 May 09]. Available from: http://dev.opera.com/articles/html5-canvas-basics/.

18 Steve Fulton JF. Introduction to HTML5 Canvas. [Internet]. 2013 [cited 2014 May 09]. Available from: http://chimera.labs.oreilly.com/books/1234000001654/ch01.html#the_document_object_model_open_parenthes.

19 Windows developers. Retained Mode Versus Immediate Mode. [Internet]. 2010 [cited 2014 May 09]. Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/ff684178(v=vs.85).aspx.

20 Neuberg B. Brad Neuberg biography. [Internet]. [cited 2014 May 10]. Available from: http://codinginparadise.org/about/.

21 Neuberg B. Introduction to HTML5. [Internet]. 2009 [cited 2014 May 10]. Available from: https://www.youtube.com/watch?v=siOHh0uzcuY.

22 Deveria A. Can I use SVG? [Internet]. 2014 [cited 2014 May 10]. Available from: http://caniuse.com/svg.

23 Deveria A. Can I use HTML5 Canvas element? [Internet]. 2014 [cited 2014 May 10]. Available from: http://caniuse.com/canvas.

24 Tutorialspoint. What is JavaScript? [Internet]. [cited 2014 May 10]. Available from: http://www.tutorialspoint.com/javascript/javascript_overview.htm.

25 Tmunotein IS. Clent-side and Server-side JavaScript. [Internet]. 2004 [cited 2014 May 10]. Available from: http://www.devarticles.com/c/a/JavaScript/Client-side-and-Server-side-JavaScript/.

26 World Wide Web Consorteum. A Short History of JavaScript. [Internet]. 2012 [cited 2014 May 10]. Available from: https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript.

27 Koning Ed. Polyline Simplification. [Internet]. 2011 [cited 2014 May 10]. Available from: http://www.codeproject.com/Articles/114797/Polyline-Simplification.

28 Mathworks. Hierarchical Clustering. [Internet]. 2014 [cited 2014 May 10]. Available from: http://www.mathworks.se/help/stats/hierarchical-clustering.html.

29 Visually. Common Chart Types. [Internet]. [cited 2014 May 12]. Available from: http://visual.ly/learn/common-chart-types.

30 JavaScript Graphs and Charts libraries. [Internet]. 2014 [cited 2014 May 10]. Available from: http://socialcompare.com/en/comparison/javascript-graphs-and-charts-libraries.

31 D3. D3 wiki. [Internet]. 2014 [cited 2014 May 10]. Available from: https://github.com/mbostock/d3/wiki.

32 Why Build Data Visualizations with D3.js. [Internet]. [cited 2014 May 10]. Available from: https://www.dashingd3js.com/why-build-with-d3js.

33 D3. Overview. [Internet]. 2014 [cited 2014 May 10]. Available from: http://d3js.org/.

34 Why D3.js is So Great for Data Visualization. [Internet]. 2013 [cited 2014 May 10]. Available from: http://blog.visual.ly/why-d3-js-is-so-great-for-data-visualization/.

35 Highcharts. What is Highcharts. [Internet]. [cited 2014 May 10]. Available from: http://www.highcharts.com/products/highcharts.

36 Highcharts. Highcharts Options References. [Internet]. [cited 2014 May 10]. Available from: http://api.highcharts.com/highcharts.

37 Flotr2. Flotr2. [Internet]. [cited 2014 May 10]. Available from: http://www.humblesoftware.com/flotr2/documentation.

38 Sutherland C. Optimization methods used in Flotr2. [Internet]. 2014 [cited 2014 May 10]. Available from: https://groups.google.com/forum/#!topic/flotr2/zdL5FVItiLs.

39 Teller S. Flotr2 - my favorite JavaScript graph library. [Internet]. 2012 [cited 2014 May 10]. Available from: http://swizec.com/blog/flotr2-my-favorite-javascript-graph-library/swizec/4558.

40 CanvasJS. CanvasJS. [Internet]. [cited 2014 May 10]. Available from: http://canvasjs.com/.

41 W3schools. Browser Statistics. [Internet]. 2014 [cited 2014 May 10]. Available from: http://www.w3schools.com/browsers/browsers_stats.asp.

42 Mozilla Developer Network. Navigation Timing. [Internet]. 2013 [cited 2014 May 10]. Available from: https://developer.mozilla.org/en-US/docs/Navigation_timing.

43 World Wide Web Consortium. Navigation Timing - responseEnd attribute. [Internet]. 2012 [cited 2014 May 10]. Available from: http://www.w3.org/TR/navigation-timing/#dom-performancetiming-responseend.

44 World Wide Web Consortium. Navigation Timing - navigationStart attribute. [Internet]. 2012 [cited 2014 May 10]. Available from: http://www.w3.org/TR/navigation-timing/#dom-performancetiming-navigationstart.

45 Google Developers. Profiling JavaScript Performance. [Internet]. 2014 [cited 2014 May 10]. Available from: https://developers.google.com/chrome-developer-tools/docs/cpu-profiling.

46 Highcharts. Demo - Scatter plot. [Internet]. [cited 2014 May 10]. Available from: http://www.highcharts.com/demo/scatter/.

Appendix 1: Line chart benchmark results

Highcharts n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.9	
10	221,80	8,55	44,70	2,05	71,70	3,35	4,74	0,05
100	216,00	7,47	44,30	1,85	71,00	3,16	4,80	0,00
1 000	218,50	4,78	53,80	1,99	82,60	3,64	5,10	0,00
10 000	249,90	12,94	94,10	2,74	137,70	3,66	8,00	0,00
100 000	187,30	7,99	162,20	14,15	132,30	3,66	8,39	0,03
500 000	186,60	12,76	153,90	6,01	131,30	5,50	8,36	0,05
1 000 000	191,90	14,31	155,70	7,90	129,80	4,56	8,35	0,05
5 000 000	180,20	12,34	157,70	11,68	134,20	8,45	8,33	0,12

D3 n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	6						1.9	
10	222,40	13,56	12,20	0,98	25,60	3,56	4,36	0,12
100	220,40	4,57	15,10	1,64	29,40	3,67	4,46	0,05
1 000	219,80	7,65	23,30	2,79	39,50	4,39	4,52	0,04
10 000	227,50	9,07	92,40	13,35	104,80	14,10	5,70	0,00
100 000	186,50	8,67	149,00	13,89	139,40	3,58	4,27	0,05
500 000	200,90	7,23	157,70	7,17	146,30	7,23	4,33	0,21
1 000 000	210,00	13,79	158,10	8,36	147,40	6,70	4,33	0,09
5 000 000	310,60	64,21	184,60	14,02	177,22	13,57	4,51	0,92

Flotr2 n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	6						1.9	
10	251,80	53,12	32,00	5,31	35,80	5,06	4,15	0,08
100	243,90	39,05	32,10	4,37	37,00	6,59	4,14	0,08
1 000	253,00	57,59	44,00	8,17	55,40	5,22	4,28	0,13
10 000	232,40	38,47	70,90	10,62	85,40	12,34	4,60	0,15
100 000	228,60	16,96	100,30	5,97	99,00	13,80	5,02	0,10
500 000	235,50	20,21	100,40	11,12	99,60	9,93	5,14	0,46
1 000 000	228,80	21,25	101,90	11,16	103,80	14,17	5,08	0,41
5 000 000	223,60	32,35	97,50	8,61	95,50	12,60	4,88	0,14

CanvasJS n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.9	
10	334,60	67,29	50,10	8,04	65,90	12,72	4,06	0,07
100	305,20	71,88	49,40	6,90	65,10	9,67	3,80	0,00
1 000	265,10	84,52	54,40	7,72	66,90	9,46	3,97	0,12
10 000	222,00	11,29	75,20	12,97	90,30	17,42	5,14	0,12
100 000	291,40	89,62	111,30	8,21	128,70	19,24	6,34	0,52
500 000	251,70	61,86	109,80	7,45	138,50	12,71	6,09	0,11
1 000 000	232,10	32,17	120,20	10,04	128,50	15,29	5,98	0,16
5 000 000	275,70	62,28	109,20	5,69	134,70	18,00	6,08	0,42

Appendix 2: Scatterplot benchmark results

Highcharts n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.6	
10	228,70	44,45	58,70	10,48	70,50	7,05	4,57	0,11
100	204,40	55,81	91,30	42,01	92,70	13,65	4,60	0,00
1 000	219,80	22,38	197,20	11,75	253,90	31,07	5,63	0,06
10 000	222,80	25,52	1508,00	59,65	1561,90	65,06	14,86	0,42
100 000	1122,70	328,67	11574,60	172,12	13467,30	246,30	114,70	0,46
500 000	Browser crashed							

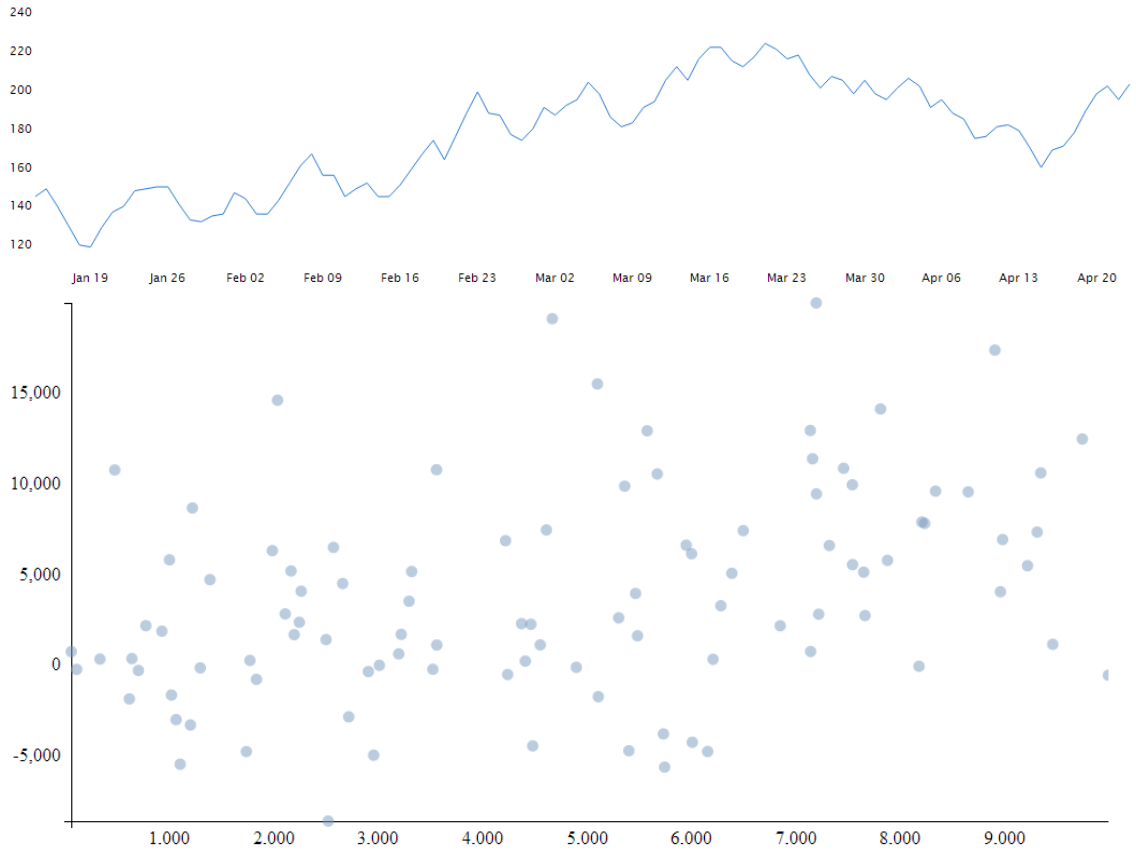
D3 n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.6	
10	254,80	41,41	13,10	1,14	16,80	1,60	4,00	0,00
100	230,20	21,34	25,20	10,51	23,20	4,42	3,96	0,12
1 000	257,60	27,61	42,10	7,37	51,40	8,42	4,10	0,00
10 000	252,80	24,24	296,90	13,63	344,50	18,92	5,04	0,12
100 000	317,30	18,04	2507,50	125,66	2548,10	115,03	13,64	0,08
500 000	525,80	40,46	11054,90	650,91	9416,00	2714,69	51,70	0,00

Flotr2 n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.6	
10	254,90	35,10	25,80	4,04	30,30	3,44	4,06	0,12
100	270,70	67,27	27,10	4,25	33,40	4,98	4,12	0,06
1 000	315,40	72,02	39,30	6,25	47,80	6,78	4,10	0,00
10 000	263,80	85,11	169,80	11,80	175,60	10,51	4,50	0,00
100 000	244,40	98,93	30740,80	1474,63	30901,30	1528,81	8,38	0,06
500 000	Took more than 2 minutes							

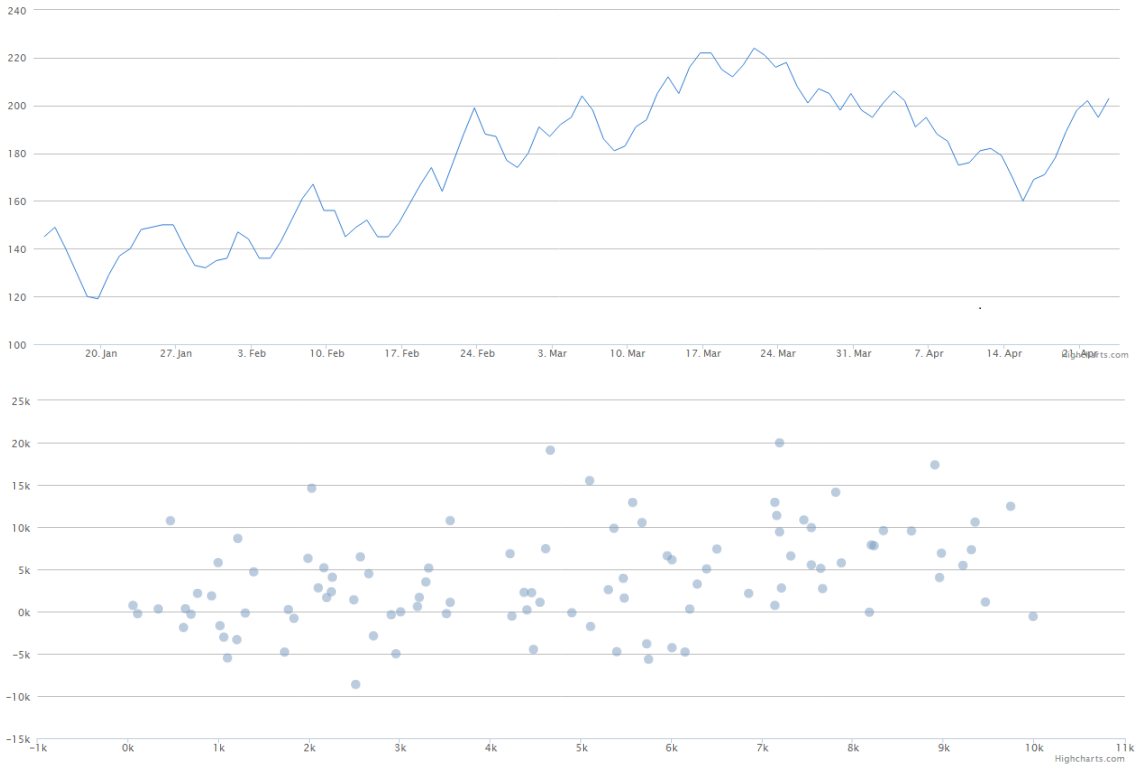
CanvasJS n = 10	Page load (ms)		Measured time (ms)		Function time (ms)		Memory usage (MB)	
	μ	σ	μ	σ	μ	σ	μ	σ
Blank page	7						1.6	
10	268,80	45,86	36,40	6,61	42,40	6,68	3,99	0,03
100	312,20	52,34	40,80	9,94	47,80	13,59	4,01	0,05
1 000	274,30	73,97	66,90	17,79	75,70	18,06	4,12	0,06
10 000	321,10	25,84	198,20	17,66	243,10	17,35	5,06	0,05
100 000	309,10	66,94	1507,80	38,27	1696,00	52,19	13,95	0,07
500 000	275,60	71,78	18435,10	361,48	19636,00	315,11	53,10	0,09

Appendix 3: Examples of visualizations

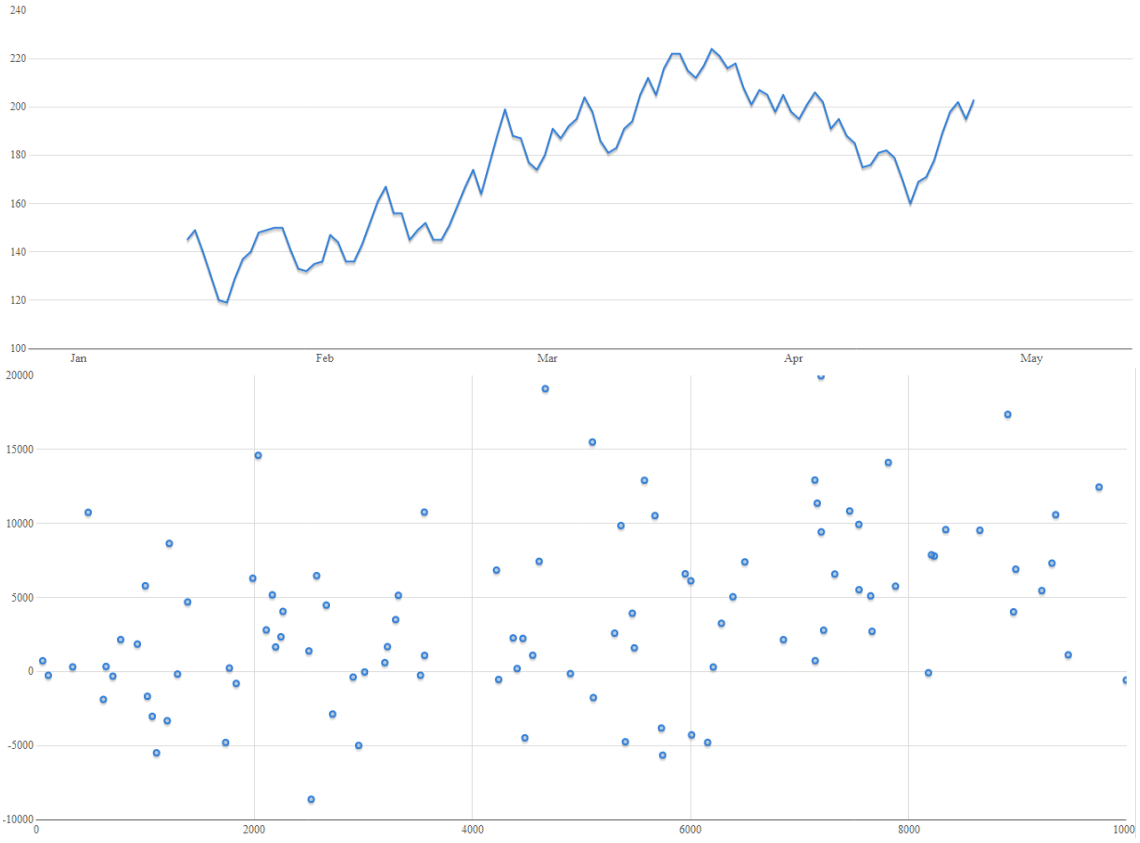
D3



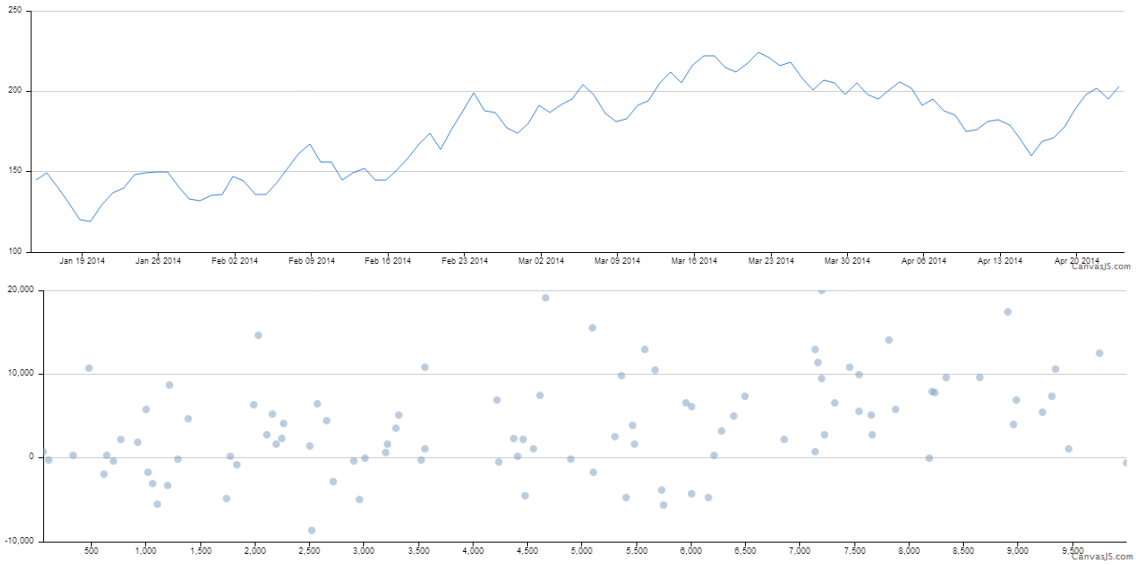
Highcharts



Flotr2



CanvasJS



Appendix 4: Attached Digitally

- Code
- Readme file
- Measured data
- Screenshots of all the final graphs

License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Artur Käpp** (date of birth: March 9, 1992),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

Comparison of JavaScript Graph Frameworks,

supervised by Margus Niitsoo,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **14.05.2014**