FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Comparative Study of web-based Visualization Technologies on Mobile Devices

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Matthias Krug
Matrikelnummer 0828965

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Dipl.Ing. Bilal Alsallakh, Dr.techn.

Wien, 22.11.2015 _______________________ _______________________
(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

A Comparative Study of web-based Visualization Technologies on Mobile Devices

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Matthias Krug
Registration Number 0828965

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Dipl.Ing. Bilal Alsallakh, Dr.techn.

Vienna, 22.11.2015

(Signature of Author) (Signature of Advisor)

Abstract

Smartphones and tablets become widespread computing devices, both for personal users, and in business domains. In many cases, these applications involve displaying visual representations of data, as well as interactivity features to explore these visualization.

Due to the varying operating systems used by these handheld devices, it becomes cumbersome to provide native applications (apps) for a large spectrum of devices. To overcome this problem, one alternative is to develop these applications using web-based technologies that are supported and standardized for multiple platforms. Various web-based technologies have been developed for displaying graphical elements, such as Scaleable Vector Graphics and WebGL. This poses a challenge for web developers who aim to select suited technologies for their applications.

This thesis surveys state-of-the art tehnologies for creating interactive web-based visualizations for mobile applications. Furthermore, the thesis reports a quantiative performance comparison of three main technologies: SVG, Canvas, and WebGL.

For each technology a representative library was selected and three pre-defined visualization scenarios were implemented using these libraries to enable the comparison of these technologies in a unified way. The three test prototypes of each technology were executed on multiple platforms and using different: a desktop computer, a tablet, and a mobile device, using three different browser: Safari, Chrome, and Internet Explorer. The corresponding execution times needed to create the respective visualizations are measured and reported. Furthermore, the three technologies are compared with respect to their support of interactivitiy and the extensibility to create new chart types.

These results show that every framework has its merits and short-comings, depending on the application. By this, the thesis provides a guidance for developers who in choosing an appropriate technology for their applications. Furthermore, our open-source prototypical implementation of three types of visualizations using three different technologies help conducting further comparisons between these technologies.

Contents

CHAPTER 1

Introduction

Every day more and more data is beeing generated. In 2013 [Dragland, Åse, 2013] stated that 90% of all the data in the world has been generated over the last two years. To make sense of huge amount of data, visualizations can help a lot. Our visual system is extremely well built for visual analysis [Iliinsky, Noah, 2012]. That makes it easier to get an overview over a data set and recognise connections and patterns with the help of charts and graphs than with simply looking at raw data.

Nowadays more than 42% of the world population has access to the internet through a variety of devices. About 33% of the total web pages served were to mobile phones [Kemp, Simon, 2015]. Since there are many different types of devices with different capabilities available, standardised technologies like HTML and CSS and Javascript are a good way to target a diverse audience. Native mobile applications (apps) have to be developed for every single platform, whereas the aforementioned technologies run in a browser and usually only have to be slightly tweaked for specific platforms. In addition to HTML [W3C, 1999], CSS [W3C, 2015] and Javascript [MDN, 2015], dedicated technololgies have been developed to create graphical elements such as Scalable Vector Graphics (SVG) [W3C, 2011], Canvas [W3C, 2014b] and WebGL [Khronos Group, 2015]. Adobe Flash [Adobe, 2015] and Java [Oracle, 2015b] can also be used to create interactive web-based visualization, but the support of those two technologies on non-desktop devices is not very good [Jobs, Steve, 2010] [Oracle, 2015a]. The downside of the web-based approach is that all these standards vary across operating systems and browsers in terms of usage, functional support and performance. But to assist in developing this kind of web-based applications there are a number of toolkits/libraries available. These libraries try to hide cross-browser compatibility issues and to give web developers an easy-to-use set of functions.

This thesis gives an overview of state-of-the-art web-based visualization frameworks using SVG, Canvas or WebGL with a special focus on mobile devices. For each technology one representative library is selected from a number of libraries and then compared to the others in regards to performance, interaction/animation and extensibility. It aims to give the reader a solid guidance for choosing an appropriate technology and software library for a given task.

In the following section some related work in the fields of information visualization using web standards and information vosualization on mobile devices is presented. In section 3 it is then explained how the libraries were found and how they are going to be compared. Section 4 lists the found libraries with some key facts grouped by the underlying technology they utilize. The subsequent section (5) describes the results of each scenario for every choosen library. Concluding this thesis, in section 6, with a summary of the comparison.

CHAPTER 2

Related Work

This section gives an overview over the current state of the art regarding information visualization using web standards in general and then more specifically on mobile devices.

2.1 Information visualization using web standards

Seven years ago [Lammarsch et al., 2008] compared server-based rendering and plugins like Adobe Flash, Java Applets and Microsoft Silverlight in respect to interaction. They disregard technologies such as SVG and HTML5 [W3C, 2014a] Canvas, because at this time they were either cumbersome to implement or still at an early stage of their development.

In contrast [Johnson and Jankun-Kelly, 2008] explored the capabilities of SVG and HTML5's Canvas for 2D information vizualisation. They implemented parallel coordinates and squarified treemaps und used a Java implementation as a control. They concluded that this two technologies have the potential to be used where Java Applets and Flash could not be used. But warned about their limitations regarding performance for medium and large datasets, espacially if they must be combined with interactivity.

2.2 Information visualization on mobile devices

[van Tonder and Wesson, 2008] address amongst other issues the problems and shortcomings of interfaces for mobile devices referring to small displays, slow hardware and new interaction procedures and discuss possible means to deal with these limitations.

In their work [Moelker and Wijbrandi, 2012] described the state of the art for 3D data visualization on mobile devices. They researched the currently supported HTML5 functionality and performed a benchmark which compared the performance of native (OpenGL) applications with web-based (WebGL) applications. Their conslusion is that WebGL is a good way to create 2D and 3D graphics but still lacks the speed of native OpenGL implementations. Regarding the speed of Javascript they state that its about ten times slower than native applications. Due to the lack of support and general performance issues they conclude that these techniques are not a feasible option for complex visualizations. However despite these drawbacks they acknowledged using web applications could save development effort if an application has to run on serveral platforms.

[Levkowitz and Kelleher, 2012] state that computing is going through a paradigm change. Due to a high propagation of cloud-based compute services and very capable mobile devices the combination of both provide the best computing resources. They argue that with the spreading of HTML5 and it's graphical capabilities (coupled with a cloud-based platform for the "heavy lifting"), mobile clients will enable high-performance graphics for most users most of the time. They further assert that their vision of a near future will change the display of the growing amount of data we face every day.

[Osebitz, 2015] surveys Java-based visualization libraries for Android mobile applications. He compares features like chart types and interaction possibilities as well as testing the performance in regard to rendering time and memory usage for different datasets varying in size. While Osebitz compared only Java-based libraries and this thesis focuses on web-based libraries, the scenarios are somewhat similar. And both thesis assess a library which implements the processing programming language where the same (scenario) code should be able to run on all covered platforms.

CHAPTER 3

Method

In the first part of this section it is explained how the libraries are found and analyzed, following an explanation on how the libraries are going to be compared.

3.1 Groundwork

First to find libraries for the comparision I used the two most popular search engines Google and Bing [eBizMBA Inc., 2013] as well as the help of my advisor Bilal Alsallakh. Used keywords were a combination of the following (as well as common abbreviations and different spellings):

- Library
- (Information) Visualization
- Javascript
- WebGL
- SVG
- Canvas

Libraries for plugins like Flash, Silverlight and Java are neglected, as are libraries for server-sided technologies. After that step the remaining libraries were grouped by the basic technology they use. Every library is then individually examined to explore the possibilities they provide.

Out of every category one library is then selected as a representative. The representatives were then compared to each other using three scenarios.

3.2 Scenarios

To compare the different libraries three scenarios are used where each of this scenarios focus on one of the following aspects: performance, interactivity, and extensibility

3.2.1 Performance

All libraries implement the parallel coordinates visualization which is created for five different sizes from the data set: 250, 500, 1000, 2500 and 5000. The execution time needed by the browser to initialize each of the visualization is recorded.

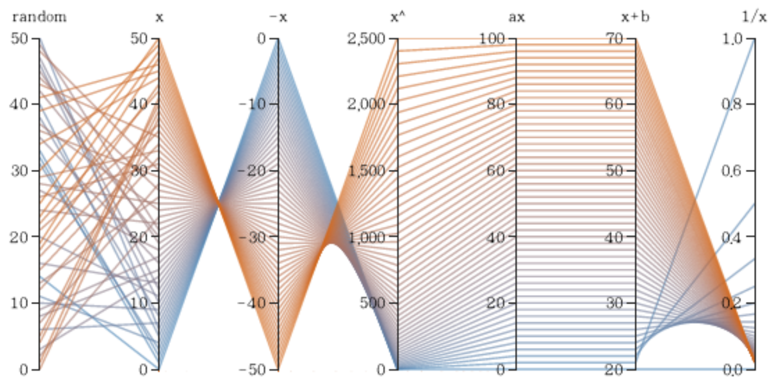


Figure 3.1: Example: Parallel Coordinates [Yug (via Wikimedia Commons), 2015]

This test is performed on different operating systems and devices. The test devices cover the following:

- Desktops:
  - **Mac OS X** (10.10.3) / **Safari** (8.0.5)
    CPU: 2,3 GHz Intel Core i7
    Memory: 16 GB 1600 MHz DDR3
    GPU: NVIDIA GeForce GT 750M 2048 M
  - **Windows** (7) / **Internet Explorer** (11)
    CPU: 3.3 GHz Interl Core i5
    Memory: 8 GB DDR3
    GPU: AMD Radeon HD 6800 Series
  - **Linux** (ArchLinux) / **Chrome** (43.0)
    CPU: 3.4 GHz AMD A10-5700 Quad-Core
    Memory: 8 GB 667 MHz DDR3
    GPU: Radeon HD 7660D
- Tablets:
  - **Android** (5.0.2) / **Chrome** (43)
    CPU: 1.3 GHz quad-core Nvidia Tegra 3
    Memory: 1 GB
    GPU: ULP GeForce

  - **iPad Mini 2** (8.1.1) / **Safari** (8.1.1)
    CPU: 1.3 GHz dual-core Apple Cyclone
    Memory: 1 GB LPDDR3 DRAM
    GPU: PowerVR G6430
  - **Windows** (8.1) / **Internet Explorer** (11)
    CPU: 1.7 GHz Intel Core i5
    Memory: 4 GB 1600 MHzr DDR3
    GPU: Intel HD Graphics 4000

- Smartphones:
  - **Android** (5.0.2) / **Chrome** (42)
    CPU: 2.2 GHz Qualcomm MSM8974 Quad Core
    Memory: 2 GB RAM
    GPU: Adreno 330
  - **iPhone 5** (8.1.2) / **Safari** (8.1.2)
    CPU: 1.3 GHz dual core
    Memory: 1 GB LPDDR2-1066 RAM
    GPU: PowerVR SGX543MP3
  - **Windows Phone** (8.1) / **Internet Explorer** (11)
    CPU: 1.2 GHz quad core ARM Cortex-A7
    Memory: 1 GB
    GPU: Qualcomm Adreno 305

The subsets of data used for the performance evaluation are drawn from a public data set which contains over 5000 entries of information about daily mortality, air pollution, and weather data. [Department of Biostatistics, Johns Hopkins Bloomberg School Of Public Health, 2005] Each entry consists of a date, the number of cardiocascular deaths, the mean monthly temperature in degrees Fahrenheit and the mean ozone level. The data was preprocessed for every library so that necessary data transformation required by these libraries do not impact the measured execution time.

3.2.2 Interaction & Animation

For this scenario the focus is on implementing a TreeMap visualization (Figure 3.2) and comparing what kind of interaction and animation options the libraries offer.

In case of the interaction possibilities special attention was paid to the kind of event handling the libraries support:

- Low level
  User Interface (UI) based Event (i.e the user clicks on a specific element)

- High level
  Specific tasks (i.e the user highlights an element or move a node in a graph)

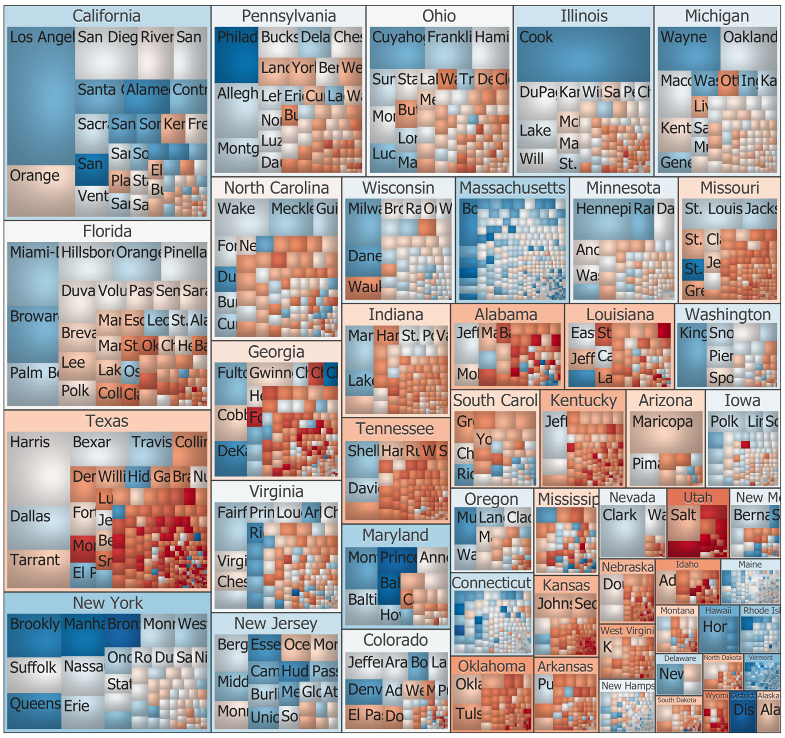
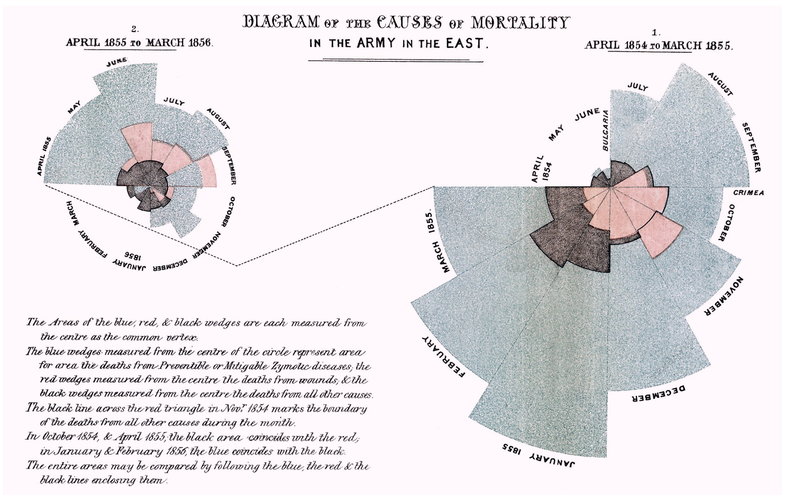


Figure 3.2: Example: TreeMap [Luc Girardin (via Wikimedia Commons), 2012]

The data set used for this scenario lists public earnings and expenditures of the city of Linz (Austria) in 2012. [KDZ - Zentrum für Verwaltungsforschung, 2012] The data set contains 2160 entries, listing the amount of money and the specific areas. The data was preprocessed for every library so that necessary data transformation required by these libraries do not impact the measured execution time.

3.2.3 Extensibility

In the last scenario a new type of visualization had to be realised that is not already supported out-of-the-box by the visualization library. So to test the extensibility of the libraries a polar area diagram is created as displayed in Figure 3.3.



The Areas of the blue, red, & black wedges are each measured from the centre as the common vertex.
The blue wedges measured from the centre of the circle represent area for area the deaths from Preventible or Mitigable Zymotic diseases; the red wedges measured from the centre the deaths from wounds; & the black wedges measured from the centre the deaths from all other causes.
The black line across the red triangle in Novr 1854 marks the boundary of the deaths from all other causes during the month.
In October 1854, & April 1855, the black area coincides with the red; in January & February 1856, the blue coincides with the black.
The entire areas may be compared by following the blue, the red & the black lines enclosing them.

Figure 3.3: Example: Polar Area Diagram [Florence Nightingale (via Wikimedia Commons), 1858]

CHAPTER 4

Overview of the Surveyed Web-based Visualization Technologies

This section provides an overview of the surveyed graphical libraries grouped by the technologies they used and gives a short description of each of these libraries. If a library supports multiple technologies it still only appears in one section. The section is choosen based on the keyword used to find the library via the search engines. [1] [2] [3]

4.1 SVG

4.1.1 Fundamentals

Scaleable Vector Graphics (SVG) is a markup language for two-dimensional graphics. These graphics can also be dynamic and allow interaction. SVG is an open standard and is designed to work with other W3C specifications and standards efforts, like CSS, DOM and XML. [W3C, 2011]

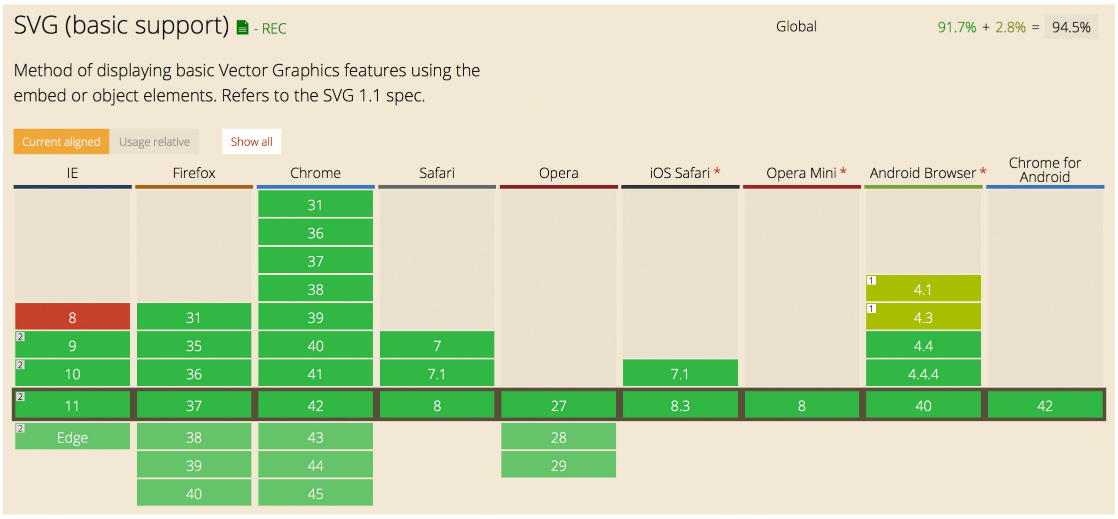
As seen in figure 4.1 the browser support for SVG is pretty good. Modern desktop and mobile browsers support this technology.

Compared to Canvas and WebGL the code for a SVG visualization can be generated on server-side, so the client simply has to display the SVG without spending additional computational time for generating the visualization. As stated in [Kee et al., 2012] a downside of SVG is that the developer can't control when rendering occurs.

[1]Regarding not specified browser support: See the Browser Support Section of every technology to see the supported versions.

[2]Regarding interaction: The library gets a checkmark for this section, if it supports some kind of interaction like dragging, zooming, clicking and so on.

[3]Regarding events: Low-level events: This kind of event gets fired, if a user interacts with an element (like clicking on an ui element). High-level events: Events, that get fired if a certain task is completed (like a node was moved in the graph)


SVG (basic support) - REC

Global 91.7% + 2.8% = 94.5%

Method of displaying basic Vector Graphics features using the embed or object elements. Refers to the SVG 1.1 spec.

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		38					[1] 4.1	
8	31	39					[1] 4.3	
[2] 9	35	40	7				4.4	
[2] 10	36	41	7.1		7.1		4.4.4	
[2] 11	37	42	8	27	8.3	8	40	42
[2] Edge	38	43		28				
	39	44		29				
	40	45						

Figure 4.1: Browser Support SVG [caniuse.com, 2015b]

Legend		
Green	..	Full support
Olive	..	Partial support (i.e. not supporting masking)
Red	..	No support

4.1.2 Libraries

4.1.2.1 D^3 - Data-Driven Documents

“D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG and CSS. D3’s emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.” [Bostock, Mike, 2013a]

4.1.2.2 Protovis

“Protovis composes custom views of data with simple marks such as bars and dots. Unlike low-level graphics libraries that quickly become tedious for visualization, Protovis defines marks through dynamic properties that encode data, allowing inheritance, scales and layouts to simplify construction.” [Bostock, Mike, 2013b]

Superseeded by d3.js

Browser Support	Only “modern browsers” specified
Line Charts	✓
Tree Maps	✓
Radial Diagrams	✓
Node-Link	✓
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	SVG
Development status	Active (Last activity: Jan. 2014)
License	BSD 3-clause
Interaction	✓
Support of events	Low-level: ✓, High-level: ✓

Table 4.1: D3.js Overview

Browser Support	Only “modern browsers” specified
Line Charts	✓
Tree Maps	✓
Radial Diagrams	✓
Node-Link	✓
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	SVG
Development status	Inactive (Superseded by D3.js)
License	BSD 2-clause
Interaction	✓
Support of events	Low-level: ✓, High-level: ✓

Table 4.2: Protovis Overview

4.1.2.3 Raphaël

“Raphaël is a small JavaScript library that should simplify your work with vector graphics on the web. If you want to create your own specific chart or image crop and rotate widget, for example, you can achieve it simply and easily with this library.

Raphaël uses the SVG W3C Recommendation and VML as a base for creating graphics. This means every graphical object you create is also a DOM object, so you can attach JavaScript event handlers or modify them later. Raphaël’s goal is to provide an adapter that will make drawing vector art compatible cross-browser and easy.” [Baranovskiy, Dmitry, 2013]

Browser Support	Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera 9.5+ and Internet Explorer 6.0+.
Line Charts	×
Tree Maps	×
Radial Diagrams	×
Node-Link	×
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	SVG
Development status	Active (last activity: December 2013)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ✓

Table 4.3: Raphaël Overview

4.1.2.4 Bonsai

"Bonsai is a graphics library which includes an intuitive graphics API and an SVG renderer." [uxebu, 2013]

Browser Support	Safari 5+, Chrome 20+, Firefox 18+, Opera 12+, IE 9+
Line Charts	×
Tree Maps	×
Radial Diagrams	×
Node-Link	×
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	SVG
Development status	Active (last activity: November 2013)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.4: Bonsai Overview

4.1.2.5 Vega

"Vega is a visualization grammar, a declarative format for creating, saving and sharing visualization designs.

With Vega you can describe data visualizations in a JSON format, and generate interactive views using either HTML5 Canvas or SVG." [Trifacta Inc., 2013]

Browser Support	Not specified.
Line Charts	×
Tree Maps	✓
Radial Diagrams	×
Node-Link	✓
Number of supported chart types	8
Technologies	Canvas, SVG
Development status	Active (last activity: October 2013)
License	BSD 3-caluse
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.5: Vega Overview

4.2 Canvas

4.2.1 Fundamentals

The canvas element is part of the HTML5 specification and provides a bitmap canvas. Through scripting it is possible to dynamically create two-dimensional shapes and bitmap images. [W3C, 2014b]

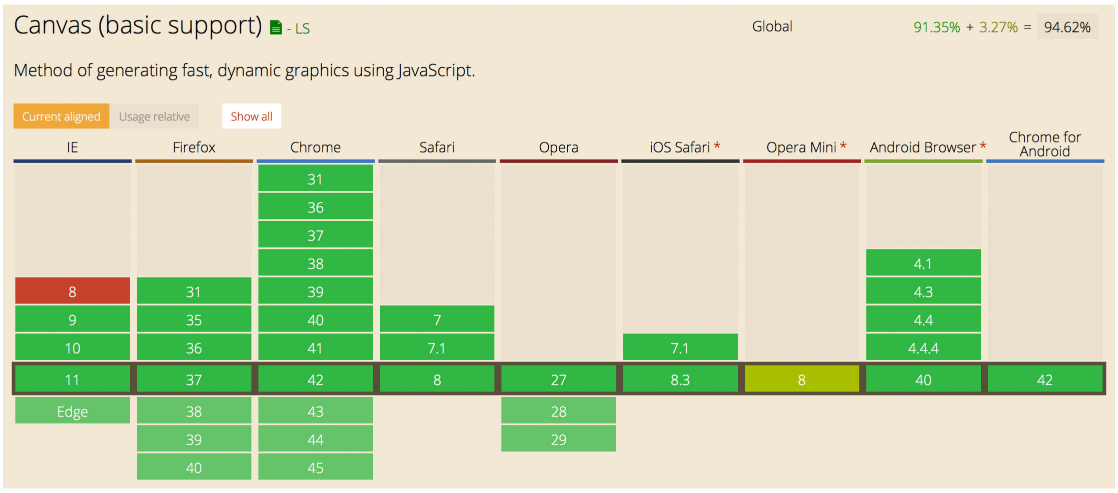
Like SVG the browser support for canvas is very good. All relevant modern desktop and mobile browser support this technology. (See figure 4.2)

Compared to SVG the developer has control over when rendering should occur, also the performance is better. [Kee et al., 2012]

4.2.2 Libraries

4.2.2.1 Processing.js

"Processing.js is the sister project of the popular Processing visual programming language, designed for the web. Processing.js makes your data visualizations, digital art, interactive animations, educational graphs, video games, etc. work using web standards and without any plug-ins.



Canvas (basic support) - LS

Global 91.35% + 3.27% = 94.62%

Method of generating fast, dynamic graphics using JavaScript.

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		38					4.1	
8	31	39					4.3	
9	35	40	7				4.4	
10	36	41	7.1		7.1		4.4.4	
11	37	42	8	27	8.3	8	40	42
Edge	38	43		28				
	39	44		29				
	40	45						

Figure 4.2: Browser Support Canvas [caniuse.com, 2015a]

Legend		
Green	..	Full support
Olive	..	Partial support (i.e. unable to play animations or run other more complex applications)
Red	..	No support

You write code using the Processing language, include it in your web page, and Processing.js does the rest." [Processing.js team, 2013]

Browser Support	Not specified.
Line Charts	×
Tree Maps	×
Radial Diagrams	×
Node-Link	×
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	Canvas
Development status	Active (last activity: November 2013)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.6: Processing.js Overview

4.2.2.2 sigma.js

"sigma.js is an open-source lightweight JavaScript library to draw graphs, using the HTML canvas element. It has been especially designed to:

Display interactively static graphs exported from a graph visualization software - like Gephi
Display dynamically graphs that are generated on the fly" [Alexis Jacomy, 2013]

Browser Support	Safari 5+, Chrome 20+, Firefox 18+, Opera 12+, IE 9+
Line Charts	×
Tree Maps	×
Radial Diagrams	×
Node-Link	✓
Number of supported chart types	1
Technologies	Canvas
Development status	Active (last activity: July 2013)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ✓

Table 4.7: sigma.js Overview

4.2.2.3 canvasXpress

"CanvasXpress was developed as the core visualization component for bioinformatics and systems biology analysis at Bristol-Myers Squibb. It supports a large number of visualizations to display scientific and non-scientific data. CanvasXpress also includes a standalone unobtrusive data table and a filtering widget to allow data exploration similar to those only seen in other high-end commercial applications." [Isaac Neuhaus, 2013]

4.3 WebGL

4.3.1 Fundamentals

WebGL is a rendering API for (hardware-accelerated) 3D graphics and is designed as a rendering context for the canvas element. The API is derived from OpenGL ES 2.0 specification. [Khronos Group, 2015]

Browser Support	Firefox 1.5+, Opera 9+, Safari 3+, Chrome 1+, IE 6+
Line Charts	✓
Tree Maps	✓
Radial Diagrams	×
Node-Link	×
Number of supported chart types	22
Technologies	Canvas
Development status	Active
License	GPLv3
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.8: canvasXpress Overview

WebGL has the worst browser support of the three technologies. Firefox and Opera have only a partial support and Safari and Internet Explorer only support WebGL in their latest version. From the popular mobile browsers only iOS Safari and IE Mobile support WebGL in the latest versions. Mobile Chrome and the Android Browser for Android 5.x have only partial support, the Android Browser up until Android 4.4.4 does not support WebGL at all. (see figure 4.3)

Since WebGL leverages hardware acceleration it obviously provides the best performance. It is also the only technology of the three mentioned that support 3D. But on the other hand developing WebGL visualizations is more elaborating. [Kee et al., 2012]

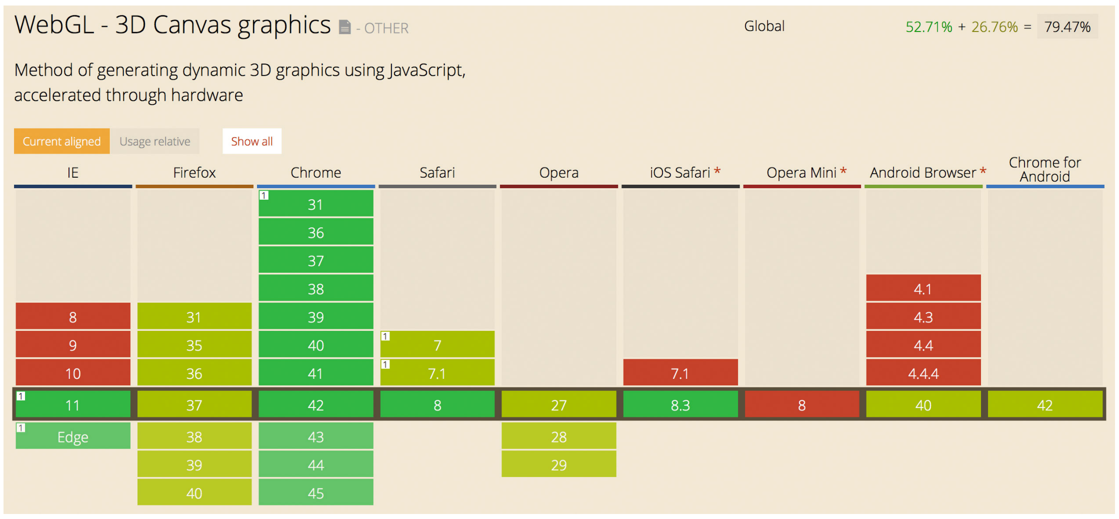
4.3.2 Libraries

4.3.2.1 JavaScript InfoVis Toolkit

"The JavaScript InfoVis Toolkit provides tools for creating Interactive Data Visualizations for the Web." [Sencha Labs, 2013]

4.3.2.2 Superconductor

"Superconductor is a web framework for creating data visualizations that scale to real-time interactions with up to 1,000,000 data points. It compiles to WebCL, WebGL, and web workers to unleash the power of parallel hardware for fast and cross-platform data visualization." [The Superconductor Team, 2013]



WebGL - 3D Canvas graphics - OTHER

Global 52.71% + 26.76% = 79.47%

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

Current aligned Usage relative Show all

IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		36						
		37						
		38					4.1	
8	31	39					4.3	
9	35	40	7				4.4	
10	36	41	7.1		7.1		4.4.4	
11	37	42	8	27	8.3	8	40	42
Edge	38	43		28				
	39	44		29				
	40	45						

Figure 4.3: Browser Support WebGL [caniuse.com, 2015c]

Legend		
Green	..	Full support
Olive	..	Partial support (i.e. not all users with these browsers have WebGL access)
Red	..	No support

4.3.2.3 Fluiddiagrams

"FluidDiagrams is a cross-platform, web-based information visualisation framework using JavaScript and WebGL." [Wright, Benedict, 2014]

Browser Support	Not specified.
Line Charts	×
Tree Maps	✓
Radial Diagrams	✓
Node-Link	✓
Number of supported chart types	10
Technologies	WebGL, Canvas
Development status	Active (Last activity: January 2014)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.9: JavaScript InfoVis Toolkit Overview

Browser Support	Not specified.
Line Charts	×
Tree Maps	✓
Radial Diagrams	×
Node-Link	×
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	WebGL
Development status	Active (Last activity: November 2013)
License	BSD 3-clause
Interaction	✓
Support of events	Low-level: ✓, High-level: ×

Table 4.10: Superconductor Overview

Browser Support	Not specified.
Line Charts	✓
Tree Maps	×
Radial Diagrams	×
Node-Link	✓
Number of supported chart types	Not specified - Framework for developing all kinds of charts
Technologies	WebGL
Development status	Inactive (Last activity: 2013)
License	MIT
Interaction	✓
Support of events	Low-level: ✓, High-level: ✓

Table 4.11: Superconductor Overview

CHAPTER 5

Comparison

5.1 1st Scenario

5.1.1 SVG

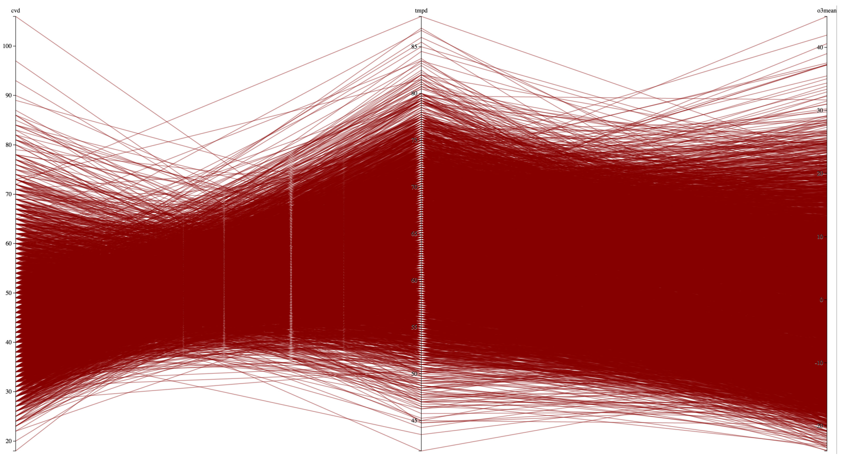


Figure 5.1: 1st Scenario using SVG

	Chrome	Safari	Internet Explorer
250	24ms	8ms	21ms
500	32ms	10ms	36ms
1000	44ms	13ms	41ms
2500	88ms	23ms	80ms
5000	197ms	42ms	146ms

Table 5.1: Results from first scenario using SVG in desktop browsers

	Chrome	Safari	Internet Explorer
250	185ms	31ms	81ms
500	218ms	48ms	109ms
1000	364ms	56ms	121ms
2500	799ms	95ms	200ms
5000	1703ms	167ms	273ms

Table 5.2: Results from first scenario using SVG in tablet browsers

	Chrome	Safari	Internet Explorer
250	104ms	87ms	173ms
500	160ms	124ms	236ms
1000	242ms	170ms	334ms
2500	488ms	234ms	702ms
5000	1066ms	343ms	1318ms

Table 5.3: Results from first scenario using SVG in smartphone browsers

5.1.2 Canvas

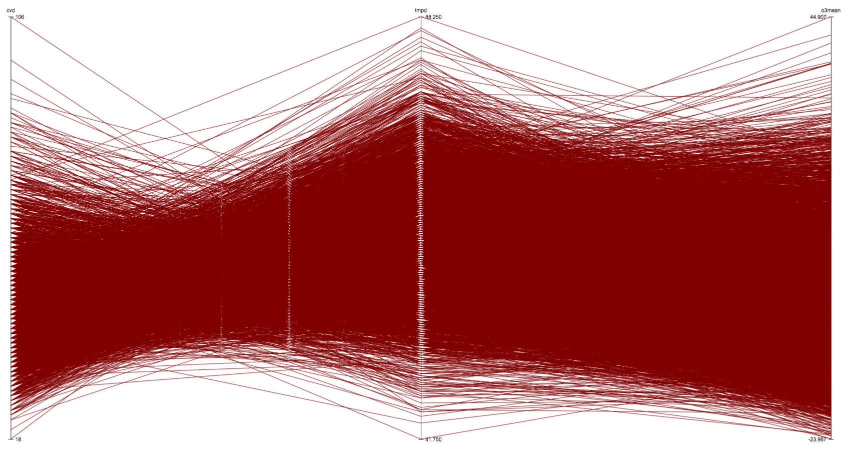


Figure 5.2: 1st Scenario using Canvas

	Chrome	Safari	Internet Explorer
250	7ms	4ms	4ms
500	9ms	6ms	6ms
1000	15ms	9ms	10ms
2500	30ms	19ms	24ms
5000	45ms	59ms	50ms

Table 5.4: Results from first scenario using Canvas in desktop browsers

	Chrome	Safari	Internet Explorer
250	63ms	23ms	10ms
500	68ms	31ms	16ms
1000	118ms	49ms	26ms
2500	230ms	107ms	35ms
5000	437ms	191ms	74ms

Table 5.5: Results from first scenario using Canvas in tablet browsers

	Chrome	Safari	Internet Explorer
250	65ms	39ms	35ms
500	71ms	59ms	105ms
1000	161ms	94ms	158ms
2500	260ms	227ms	275ms
5000	377ms	332ms	499ms

Table 5.6: Results from first scenario using Canvas in smartphone browsers

5.1.3 WebGL

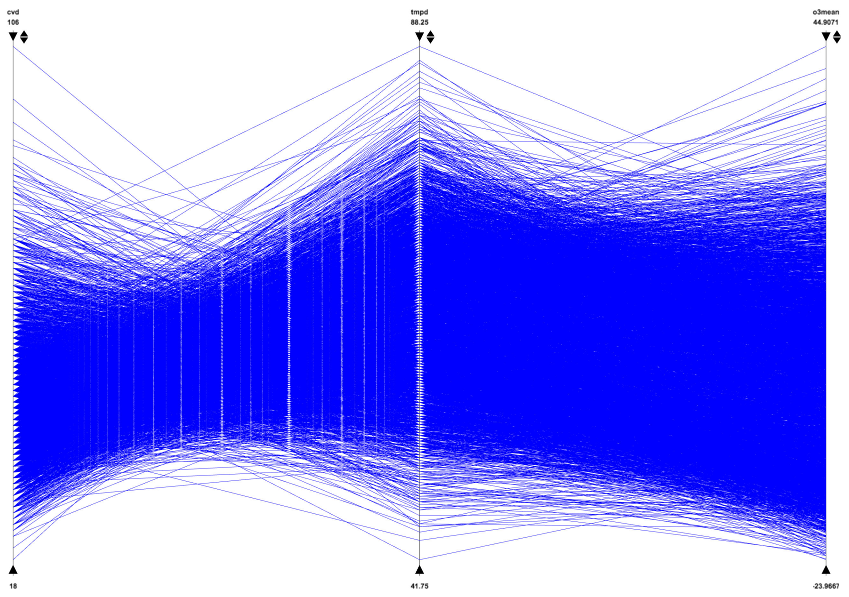


Figure 5.3: 1st Scenario using WebGL

	Chrome	Safari	Internet Explorer
250	196ms	125ms	211ms
500	269ms	161ms	263ms
1000	399ms	217ms	374ms
2500	843ms	466ms	740ms
5000	1425ms	916ms	1360ms

Table 5.7: Results from first scenario using WebGL in desktop browsers

	Chrome	Safari	Internet Explorer
250	1151ms	316ms	478ms
500	1702ms	444ms	604ms
1000	2944ms	759ms	716ms
2500	8244ms	2637ms	1336ms
5000	25048ms	4818ms	2346ms

Table 5.8: Results from first scenario using WebGL in tablet browsers

	Chrome	Safari	Internet Explorer
250	962ms	524ms	2903ms
500	1199ms	789ms	3261ms
1000	1967ms	1058ms	4625ms
2500	3818ms	2254ms	10204ms
5000	6663ms	3730ms	19122ms

Table 5.9: Results from first scenario using WebGL in smartphone browsers

5.1.4 Summary

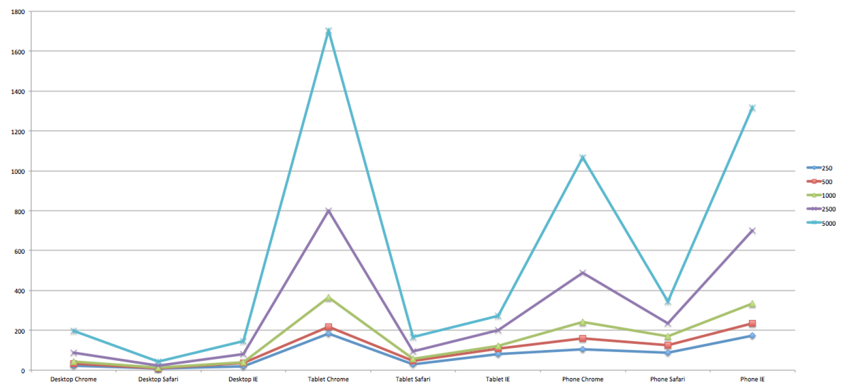


Figure 5.4: SVG Results of the 1st Scenario

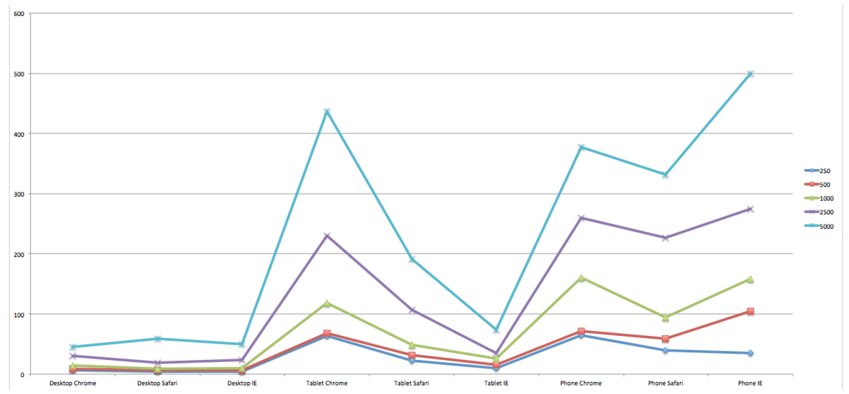


Figure 5.5: Canvas Results of the 1st Scenario

The Canvas implementations are nearly always the fastest, SVG comes in second and WebGL in last. While SVG is only a little slower, I suspect the gap of WebGL comes from the overhead of a 3D API used in a 2D way only and the architecture of the Fluiddiagrams library.

In regard to the browsers Safari seems to be the fastest across all devices and across all technologies. Especially using SVG and larger data sets Safari outdistance the other browsers. Using Canvas technology the Internet Explorer is sometimes slightly faster than Safari.

5.2 2nd Scenario

5.2.1 SVG

D3.js provides low-level mouse/touch and keyboard-based events and, depending on the used layout, high-level events as well. Because SVG is build upon the Document Object Model most

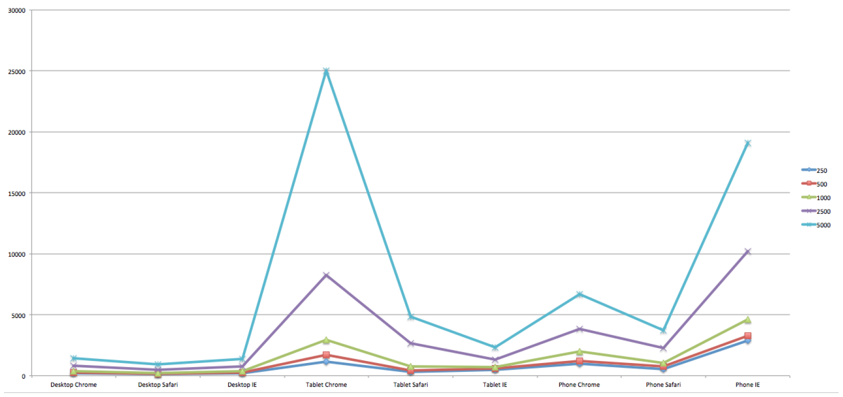


Figure 5.6: WebGL Results of the 1st Scenario

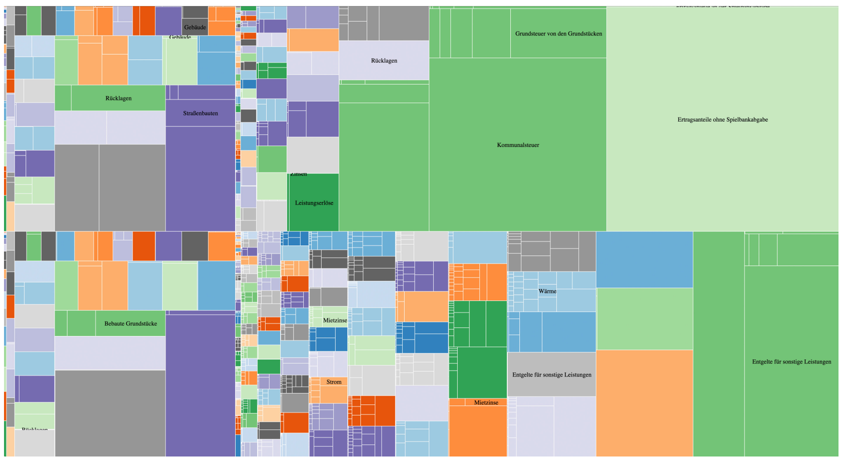

Gebäude
Rücklagen
Straßenbauten
Rücklagen
Grundsteuer von den Grundstücken
Kommunalsteuer
Ertragsanteile ohne Spielbankabgabe
Leistungserlöse
Bebaute Grundstücke
Mietzinse
Strom
Wärme
Entgelte für sonstige Leistungen
Mietzinse
Entgelte für sonstige Leistungen

Figure 5.7: 2nd Scenario using SVG

events can be mapped directly to a single element, so matching clicked elements - and thanks to D3.js also the associated data - is really easy.

To animate a chart the library provides a transition() module. You can either specify a custom tween function, or use the module to interpolate the value of a specific attribute. This makes it very easy and straight forward to orchestrate a series of sub-animations to became a whole.

5.2.2 Canvas

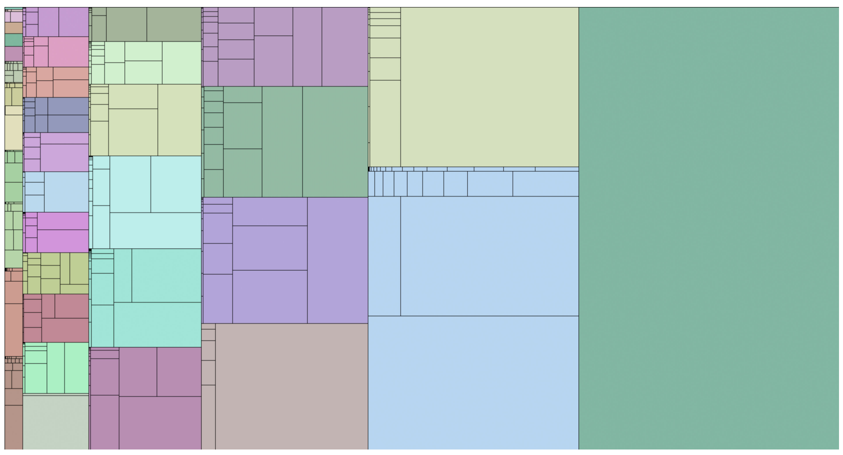


Figure 5.8: 2nd Scenario using Canvas

Processing.js provides only mouse/touch-based and keyboard-based events. For example if you want to get an element which was clicked, you have to match the coordinates of the current mouse position with the bounds of the element yourself.

The library executes the draw() function in a loop. This can be used to animate the visualization. If no animation is needed the noLoop() function can be used so the draw() function is only called once. You have to track the animation progress yourself and adapt the draw() function accordingly.

5.2.3 WebGL

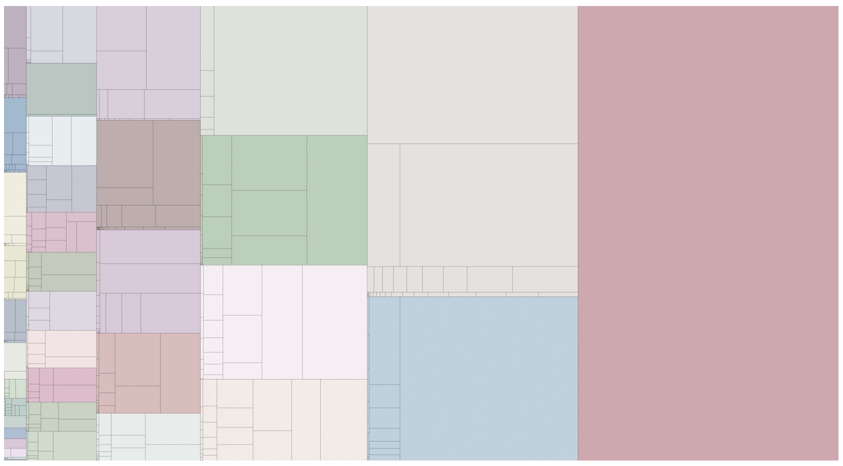


Figure 5.9: 2nd Scenario using WebGL

Like D3.js Fluiddiagrams provides low-level events and high-level ones depending on the layout. But unlike D3.js it is not that simple to map single chart elements to a mouse click. Theoretically you would only have to connect the data to the according mesh and Fluiddiagrams would assign this information to the event callback. But there seems to be a bug in this procedure and the wrong data is supplied.

Animation in Fluiddiagrams is done with the tween.js library. This library calls a custom function every few milliseconds for a specified duration and provides a set of interpolated values which can be used to repaint the canvas area.

5.3 3rd Scenario

5.3.1 SVG

With only 88 lines of code the SVG implementation with D3.js has the smallest footprint. This library provides an excellent documentation and a large number of examples. There are also a vast number of helper functions, like range converter, built in to assist in the implementation. Compared to the other two implementations, this one took the least amount of time to be build.

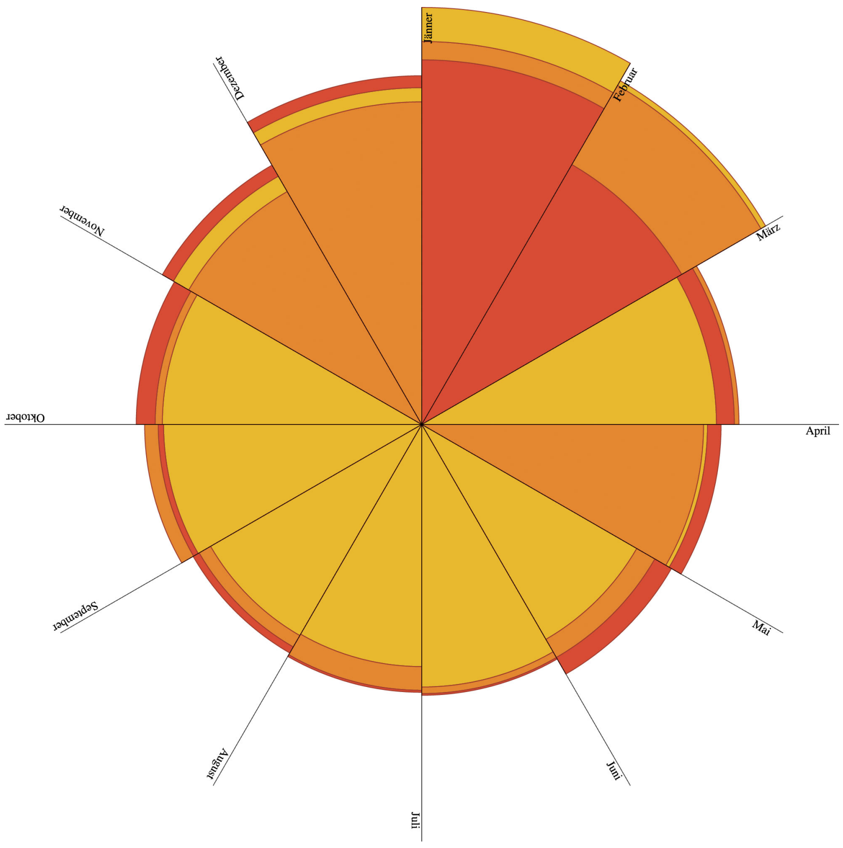


Figure 5.10: 3rd Scenario using SVG

5.3.2 Canvas

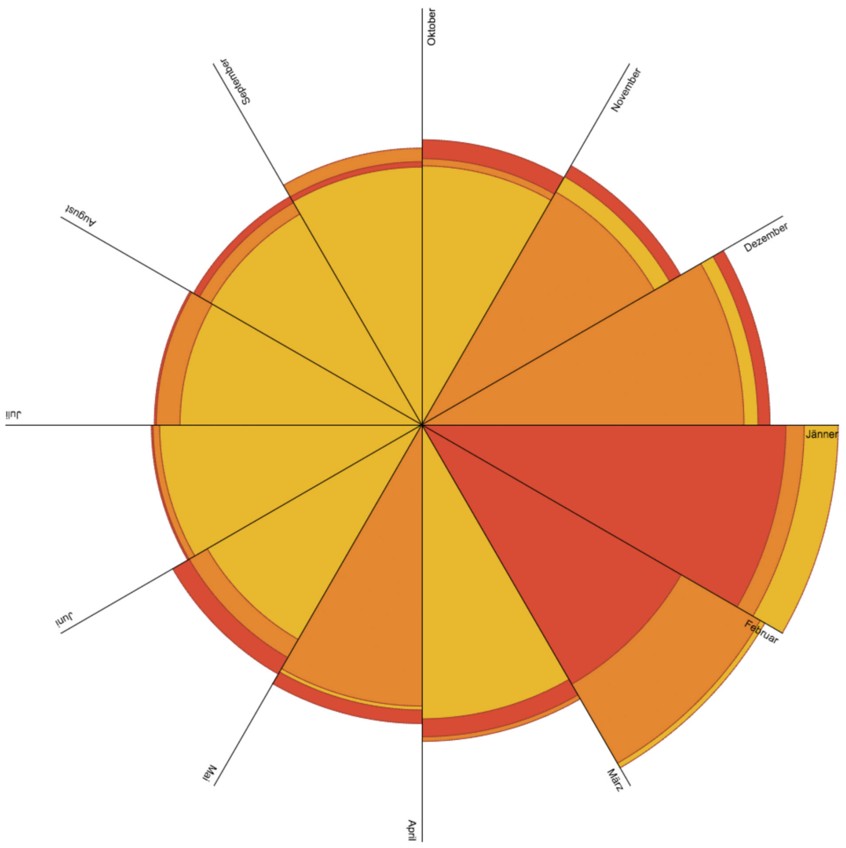


Figure 5.11: 3rd Scenario using Canvas

This implementation consists of 165 lines of code. Because Processing.js is a port of the programming language Processing, a large number of tutorials and examples - not specifically made for Processing.js - could be used to help create visualizations. In the beginning using Processing seemed a bit peculiar but soon it became very straight forward to create the chart in a short time.

5.3.3 WebGL

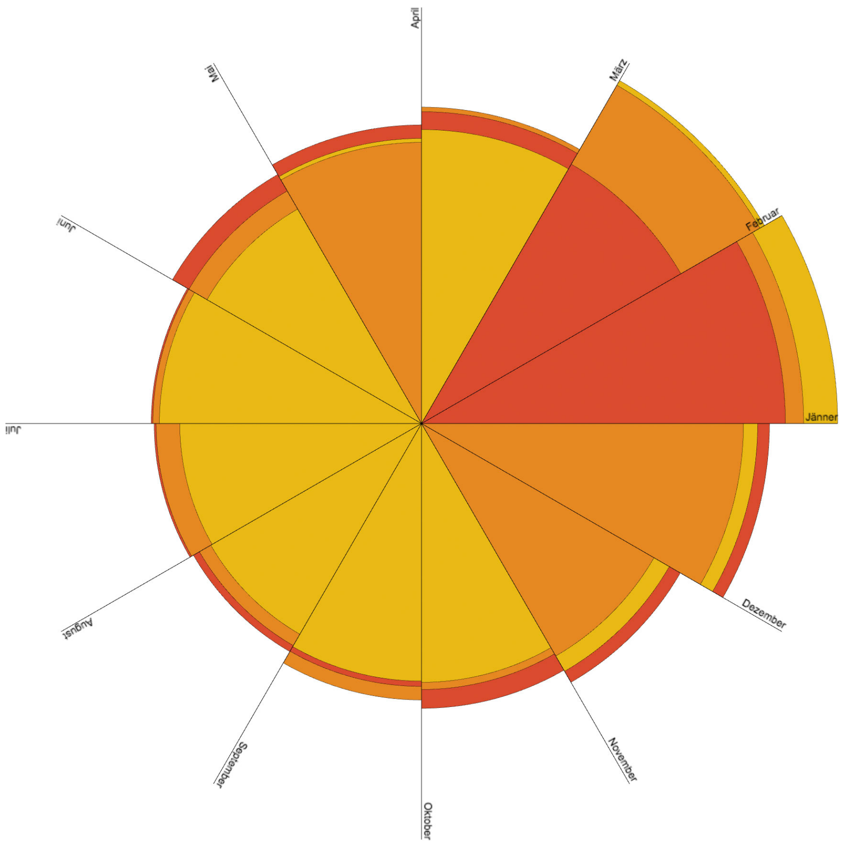


Figure 5.12: 3rd Scenario using WebGL

For the implementation of the 3rd Scenario I produced 204 lines of code. The Documentation for the Library is good, but compared to the SVG and Canvas libraries there are only a few tutorials/examples available. Fluiddiagrams has the steepest learning curve of the three libraries, but the framework provides a good structure for implementing clean and reusable extensions.

CHAPTER 6

Conclusion

This thesis gives a short overview of different libraries using different technologies for implementing charts with a special focus on mobile devices.

All selected libraries run without a problem on mobile devices, but vary in their runtime performance, with Canvas being faster than the other two. Even if an SVG visualization could became a problem to the browser because every part of a chart is a node in the Document Object Model, the flexibility of D3.js and the ease with which one can create a new visualization outweights this problem in most cases. Also making a visualization accessible is more easy with SVG than Canvas or WebGL. The WebGL Library Fludidiagrams may not be as fast as the other libraries, but would also provide the possibilities to create true 3D charts.

Each of the tested libraries support animation and some kind of interaction with the visualization. Also with all of these libraries it is possible to extend them and create new charts.

In further work additional libraries could be tested to get a more comprehensive overview. Furthermore other aspects could be included in the overview, for instance accessibility.

Bibliography

Adobe (2015). http://www.adobe.com/products/flash.html. Adobe Flash. Access: 2015-10-28.

Alexis Jacomy (2013). http://sigmajs.org/. sigma.js. JavaScript library to draw graphs. Accessed: 2013-12-04.

Baranovskiy, Dmitry (2013). http://raphaeljs.com/. Raphaël - JavaScript Library. Accessed: 2013-12-04.

Bostock, Mike (2013a). http://d3js.org/. D3 - Data-Driven Documents. Accessed: 2013-12-04.

Bostock, Mike (2013b). http://mbostock.github.io/protovis/. Protovis. A graphical approach to visualization. Accessed: 2013-12-04.

caniuse.com (2015a). http://caniuse.com/#feat=canvas. Accessed: 2015-05-01.

caniuse.com (2015b). http://caniuse.com/#feat=svg. Accessed: 2015-05-01.

caniuse.com (2015c). http://caniuse.com/#feat=webgl. Accessed: 2015-05-01.

Department of Biostatistics, Johns Hopkins Bloomberg School Of Public Health (2005). http://www.ihapss.jhsph.edu/. Internet-based Health & Air Pollution Surveillance System. Access: 2015-05-07.

Dragland, Åse (2013). http://www.sintef.no/home/corporate-news/big-data–for-better-or-worse/. Big Data – for better or worse. Accessed: 2015-05-03.

eBizMBA Inc. (2013). http://www.ebizmba.com/articles/search-engines. Top 15 Most Popular Search Engines | December 2013. Accessed: 2013-12-04.

Florence Nightingale (via Wikimedia Commons) (1858). http://commons.wikimedia.org/wiki/File:Nightingale-mortality.jpg. Accessed: 2015-05-01.

Iliinsky, Noah (2012). http://blog.visual.ly/why-is-data-visualization-so-hot/. Why Is Data Visualization So Hot? Accessed: 2015-05-03.

Isaac Neuhaus (2013). http://canvasxpress.org/. canvasXpress. A powerful canvas graphing utility. Accessed: 2013-12-04.

Jobs, Steve (2010). https://www.apple.com/hotnews/thoughts-on-flash/. Thoughts on Flash. Access: 2015-05-03.

Johnson, D. W. and Jankun-Kelly, T. J. (2008). A scalability study of web-native information visualization. In *Proceedings of Graphics Interface 2008*, GI '08, pages 163–168, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.

KDZ - Zentrum für Verwaltungsforschung (2012). http://www.offenerhaushalt.at/download/finanzdaten/top/linz/
Eingaben/Ausgaben der Stadt Linz 2012. Access: 2015-05-07.

Kee, D. E., Salowitz, L., and Chang, R. (2012). Comparing interactive web-based visualization rendering techniques.

Kemp, Simon (2015). http://wearesocial.net/blog/2015/01/digital-social-mobile-worldwide-2015/. Digital, Social & Mobile Worldwide in 2015. Access: 2015-05-03.

Khronos Group (2015). https://www.khronos.org/registry/webgl/specs/latest/1.0/. WebGL Specification. Access: 2015-05-03.

Lammarsch, T., Aigner, W., Bertone, A., Gärtner, J., Miksch, S., and Turic, T. (2008). A comparison of programming platforms for interactive visualization in web browser based applications. In Banissi, E., Stuart, L., Jern, M., Andrienko, G., Marchese, F. T., Memon, N., Alhajj, R., Wyeld, T. G., Burkhard, R. A., Grinstein, G., Groth, D., Ursyn, A., Maple, C., Faiola, A., and Craft, B., editors, *Proceedings of 12th International Conference on Information Visualisation (IV08)*, page 194–199. IEEE Computer Society Press, IEEE Computer Society Press. <p>Vortrag: 12th International Conference on Information Visualisation (IV08), London, UK; 2008-07-09 &ndash; 2008-07-11</p>.

Levkowitz, H. and Kelleher, C. (2012). Cloud and mobile web-based graphics and visualization. In *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on*, pages 21–35.

Luc Girardin (via Wikimedia Commons) (2012). http://commons.wikimedia.org/wiki/File:US_Presidential_Elec
Accessed: 2015-05-01.

MDN (2015). https://developer.mozilla.org/en-us/docs/web/javascript. JavaScript. Access: 2015-10-28.

Moelker, R. R. and Wijbrandi, W. E. (2012). Html5 data visualization capabilities of mobile devices. In *Proceedings 9th Student Colloquium 2011-2012*, pages 23–28.

Oracle (2015a). https://www.java.com/en/download/faq/java_mobile.xml. How do I get Java for Mobile device?. Access: 2015-05-03.

Oracle (2015b). https://www.oracle.com/java/index.html. Java. Access: 2015-10-28.

Osebitz, C. (2015). State of the art in java-based information visualization on mobile devices. Bachelor's Thesis.

Processing.js team (2013). http://processingjs.org/. Processing.js. A port of the Processing Visualization Language. Accessed: 2013-12-04.

Sencha Labs (2013). http://philogb.github.io/jit/. JavaScript InfoVis Toolkit. Create Interactive Data Visualizations for the Web. Accessed: 2013-12-04.

The Superconductor Team (2013). http://superconductor.github.io/superconductor/. Superconductor. Bit Data visualization for the web. Accessed: 2013-12-04.

Trifacta Inc. (2013). http://trifacta.github.io/vega/. Vega. Accessed: 2013-12-04.

uxebu (2013). http://bonsaijs.org/. Bonsai. A lightweight graphics library with an intuitive graphics API and an SVG renderer.. Accessed: 2013-12-04.

van Tonder, B. and Wesson, J. (2008). Using adaptive interfaces to improve mobile map-based visualisation. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, SAICSIT '08, pages 257–266, New York, NY, USA. ACM.

W3C (1999). http://www.w3.org/tr/html4/. HTML 4. Access: 2015-10-28.

W3C (2011). http://www.w3.org/tr/svg11/intro.html. Scaleable Vector Graphics (SVG) Specification. Access: 2015-05-03.

W3C (2014a). http://www.w3.org/tr/html5/. HTML5. Access: 2015-10-28.

W3C (2014b). http://www.w3.org/tr/html5/semantics.html#the-canvas-element. HTML5 Specification. Access: 2015-05-03.

W3C (2015). http://www.w3.org/tr/css/. CSS. Access: 2015-10-28.

Wright, Benedict (2014). http://projects.iicm.tugraz.at/fluiddiagrams/. FluidDiagrams. Accessed: 2015-05-01.

Yug (via Wikimedia Commons) (2015). http://commons.wikimedia.org/wiki/File:Parallel_coordinates-sample.png. Accessed: 2015-05-01.