

Laura Meskanen-Kundu

Making Data Accessible: An Overview of Interactive Data Visualization Using D3.js as Applied to a Scientific Dataset

Making a Static Visualization Interactive

Helsinki Metropolia University of Applied Sciences

Engineering

Media Engineering

Thesis

19 April 2015

Author(s) Title	Laura Meskanen-Kundu Making data accessible: An overview of interactive data visualization using D3.js as applied to a scientific dataset Making a static visualization interactive
Number of Pages Date	40 pages 19 April 2015
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Digital Media
Instructor(s)	Kari Salo, Principal Lecturer
<p>Technology is moving at a very fast pace, but data is still represented as tables, static graphs and infographics that do not create an impact on the population at large. Excluding the scientific and educational communities, to the common individual information should be displayed in an entertaining manner.</p> <p>This project set out to fulfill this goal by using known technologies from D3js, design guidelines, CSS3 animations, and HTML5 elements to real scientific data from the United States National Climate Data Center. The final product is a one page web application displaying 3,000,000 years of global temperatures in a visual format. The data was plotted using D3js, made interactive with JavaScript and laid out using Twitter Bootstrap.</p> <p>What can be concluded is that it is possible to create interactive content with current technologies, but the process is still only achievable after extensive study of the technologies involved. Further development has to be made for data interactive tools to become easier to use and to produce large-scale interactive web applications involving data display and analysis. The advancement of interactive visualizations are also relevant as studies have shown that engaging lectures lead to a statistically significant higher average on unit exams compared with traditional didactic lectures. This could be hypothesized to be the same for interactive data and this was confirmed by a small questionnaire.</p>	
Keywords	D3.js, scientific data, interactive web application, JavaScript, HTML5, CSS3

Contents

Abbreviations, Acronyms and Terms	1
1 Introduction	1
2 Web Programming	4
2.1 Short History of Web and Web Programming	4
2.2 What is Web Programming Today	6
2.2.1 Mark-up of Today: Focus on JavaScript	7
2.2.2 Best Practices in JavaScript	10
2.2.3 Layout and Design: Focus on Designing for Data	10
2.3 Reasons for Choosing D3.js in this Thesis Project	14
3 D3.js	17
3.1 Methods in D3	18
3.2 Vector Graphics: Focus on SVG	19
3.3 D3 Selections	20
3.4 Data-Joins	20
3.5 Scaling and Axes in Charts	21
3.6 Loading External Data	23
3.7 Adding interactions and Animating	23
4 Creating a One Page Web Application	24
4.1 Methods and Materials	24
4.1.1 Phase One: Researching for a Dataset	24
4.1.2 Phase Two: Formatting the Dataset	27
4.1.3 Phase Three: Multiple Temperature Data onto a Single Graph	30
4.1.4 Phase Four: Creating Interactions and Mouse Events	31
4.1.5 Phase Five: Styling and Appearance	34
4.1.6 Phase Six: Final Touches and Uploading to Remote Server	35
4.2 Results and Discussion	36
5 Conclusion	40
References	41

Abbreviations, Acronyms and Terms

API (Application Program Interface)

Set of commands, functions, routines, tools and protocols which programmers can use when building software. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

Big Data

“Data of a very large size, typically to the extent that its manipulation and management present significant logistical challenges.” [58.]

Cascading Style Sheets (CSS)

Used to format the layout of web pages, they can be used to define text styles, table sizes, and other aspects of web pages that previously could only be defined in a page's HTML [17].

Code Libraries

In computer science, a library is a collection of implementations of behavior, written in terms of a language, which has a well-defined interface by which the behavior is invoked [1].

Data abstraction

Reduction of a particular body of data to a simplified representation of the whole. Abstraction, in general, is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics.

Data cleansing

The process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database. Used mainly in databases, the term refers to identifying incomplete, incorrect, inaccurate and/or irrelevant parts

of the data and then replacing, modifying, or deleting this dirty data or coarse data.

D3.js (Data-Driven Documents)

D3.js is a JavaScript library for manipulating documents based on data. D3 helps bringing data to life using HTML, SVG, and CSS. D3's emphasis on web standards gives the user 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. [4.]

DOCTYPE

A document type declaration, or DOCTYPE, is an instruction that associates a particular SGML or XML document (for example, a webpage) with a document type definition (DTD) (for example, the formal definition of a particular version of HTML).

Document Object Model (DOM)

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. [25.]

Font-end

Front-end is a term used to characterize a program interface or service relative to the initial user of this interface or service. In this case the user can be a human or a program and front-end is the interface a user interacts directly. [3] For example a front-end is a text editor on a device where a user inputs data.

Framework

Describes a given structure of "how" code should be present. It is more like a code-template, along with some helpers, and constructors to solve and simplify a specific problem or bring architecture in "order". Examples, "Backbone", "AngularJS" and "Batman" [26; 34.]

gitHub

GitHub is a web-based revision control hosting service for software development and code sharing.

HyperText Markup Language (HTML)

A set of standards, a variety of SGML, used to tag the elements of a hypertext document. It is the standard protocol for formatting and displaying documents on the World Wide Web [12]. The publishing format for the web, including the ability to format documents and link to other documents and resources [11].

HyperText Transfer Protocol (HTTP)

The standard protocol for transferring hypertext documents on the World Wide Web [13]. Allows for the retrieval of linked resources from across the web [11].

JavaScript (ECMAScript)

A programming language commonly used in web development and originally developed by Netscape. While JavaScript is influenced by Java, the syntax is more similar to C and is based on ECMAScript, a scripting language developed by Sun Microsystems. It is used to add dynamic and interactive elements to websites [18]. ECMAScript and JavaScript can be used interchangeably.

jQuery

JavaScript library that uses CSS selectors to access and manipulate HTML elements (DOM Objects) on a web page. It provides a companion UI (user interface) framework and numerous other plug-ins [27].

Linter

Linting is the process of running a linter program that will analyze code for potential errors. Linter is now applied generically to tools that flag suspicious usage in software written in any computer language.

Mock-up

A scale or full-size model of a design or device, used for teaching, demonstration, design evaluation or promote. It is sometimes the result of wireframes. It is a working sample

built for reviewing format, layout, content, testing or displaying.

Same-origin policy

In computing, the same-origin policy is an important concept in the web application security model. Under the policy, a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin. An origin is defined as a combination of an URI scheme, hostname, and port number. This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model. [36.]

Sublime Text 3

Sublime Text is a cross-platform text and source code editor, with a Python application programming interface (API). Sublime Text is proprietary software. Its functionality is extendable with plugins. Most of the extending packages have free-software licenses and are community-built and maintained.

SVG (Scalable Vector Graphics)

Description of a vector image in Extensible Markup Language (XML). Any program such as a web browser that recognizes XML can display the image using the information provided in the SVG format.

Tuts+

An online education website and membership focused on creative skills.

Twitter Bootstrap

Bootstrap is a free and open-source collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions.

Uniform Resource Identifier (URI)

A kind of “address” that is unique to each resource on the web [11].

Web programming

Web programming, also known as 'web development', "is a broad term for the work involved in developing a web site for the Internet (World Wide Web) or an intranet (a private network). Web development can range from developing the simplest static single page of plain text to the most complex web-based Internet applications, electronic businesses, and social network services. A more comprehensive list of tasks to which web development commonly refers may include web design, web content development, client liaison, client-side/server-side scripting, web server and network security configuration, and e-commerce development." [10.]

Wireframes

A design tool used in web development. It is a visualization tool for presenting proposed functions, structure and content of a website. It is typically created in black and white or shades of grey, using placeholders for images. Wireframes avoid the visual design of the site and are more concerned with the organization of the content and features.

World Wide Web Consortium (W3C)

The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the web [15].

1 Introduction

Technology is moving at a very fast pace and breakthroughs are happening daily. Why then is data still being represented in ways that have not quite progressed fast enough for current standards in the information era? Figures are still being represented in tables, static graphs and infographics. These are inactive in nature and do not allow the observer to interact with them. Progress has run from analyzing data with mathematical models and placing them into different variations of non- or color-coded graphs and tables. While this, in most cases, is still practical and sufficient in higher educational and scientific communities the population at large prefers a more entertaining means. A bigger picture has to be seen when it comes to displaying information. The world has moved on from reading and observing to seeing and interacting. Individuals grab information when it is given in a format that allows them to play with it. The world has moved to touch technologies, access to information is growing and so is the demand for interesting ways of displaying it. Transitioning from paper to the Internet with interactive content interest has spiked on infographics and information in general. In figure 1 Google trends displays the search interest of the word infographic and how interest has peaked from 2009. In figure 1 numbers represent search interest relative to the highest point on the chart. If at most 10% of searches for the given region and time frame were for "info," we'd consider this 100. This doesn't convey absolute search volume. So does data analysis have to progress in the same pace, from simple graphs to on screen graphics and even special effects?

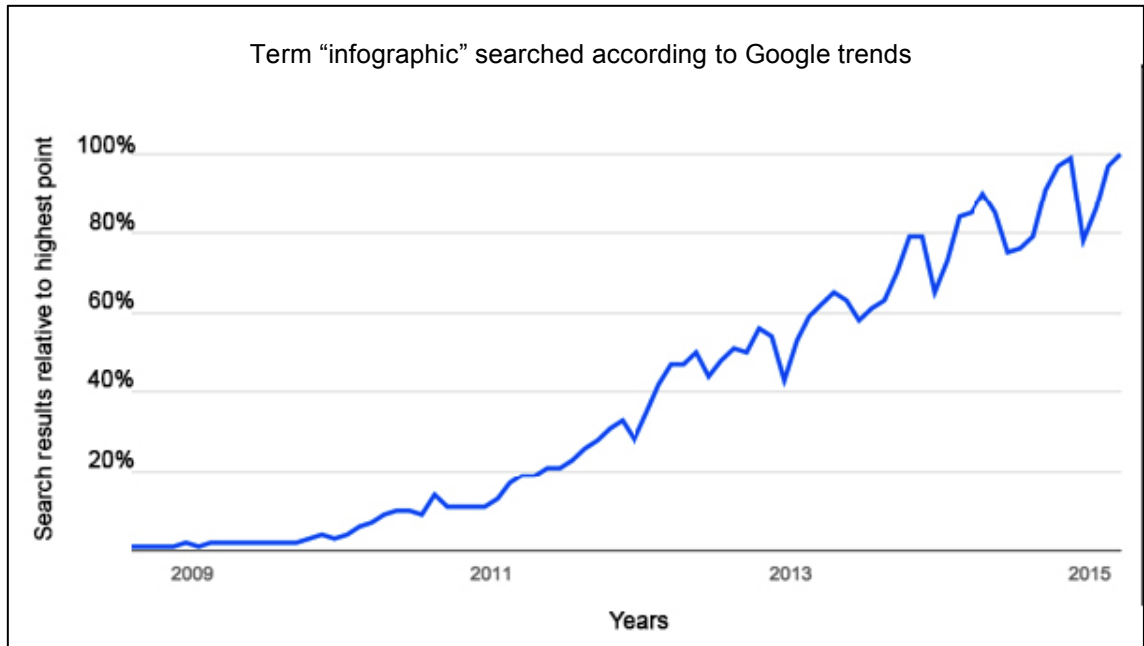


Figure 1: Search interest relative to the highest search volume. Copied and modified from Google trends. [46.]

This change is being made by new web based *coding libraries* (ready made implementations of behavior in a certain programming language) allowing anyone with knowledge of programming to create interactive content of analyzed data. [1] One great example of a *front-end* library is *D3.js* that allows anyone with sufficient knowledge of a few well-known technologies to create powerful and robust graphical forms and visualizations that can be customized. [2.] Other libraries include Dygraphs, ZingChart, jqPlot, jpGraph, Highcharts, Crossfilter, Flot, and Raphaël [5]. Technologies are clearly being developed to fill this void in current data visualization practices.

This paper is going to focus on *D3.js* and its ability to create a highly interactive one-page web application displaying open source scientific data on earth's temperature measurements 3 million years to present day [7; 8; 9]. This involved using a large dataset that cannot be considered *big data* by current standards, but that subjected to similar problems that would come with big data. Nevertheless, the aim was to see how far current technologies have come in creating an interactive experience from raw data points that will be interesting, accessible and useful to the general public. This was measured with a questionnaire at the end of running the application and results are available to the public. This thesis project set out to solve the current lack of interactive online data visualization, which is a problem for the average consumer wanting a quick and fun way to consume quantitative information. The aim was to produce an entertain-

ing one-page web application with current technologies that would also be informative and more conveying than a traditional static graph.

2 Web Programming

To understand this interactive web application built in D3.js and the scientific data analyzed, the intricacies of D3.js and *web programming* must be known. Thereby basic concepts and building blocks are presented in the first few sections. Web programming is one of these key elements in creating current interactive data visualizations. Also known as front-end programming, this is the creation of a web site for the Internet (World Wide Web) or an intranet (a private network) that is at simplest a static single page of plain text or as complex as a web-based Internet application. [10.] The following sections will go in-depth into the apprehending of this field of programming.

2.1 Short History of Web and Web Programming

“Tim Berners-Lee invented the World Wide Web in 1989, about 20 years after the first connection was established over what is today known as the Internet” [11]. As a software engineer at CERN (particle physics laboratory near Geneva, Switzerland) he wanted to tackle the problem of exchanging data and results by scientists around the world. He documented what was to become the World Wide Web and made a proposal of specifications and set of technologies that would make the Internet truly accessible and useful to people (figure 2). In this proposal lies the three fundamental technologies that remain the foundation of today’s web: *HTML: HyperText Markup Language*, *URI: Uniform Resource Identifier* and *HTTP: Hypertext Transfer Protocol*. In April 1993 the World Wide Web technology was available for anyone to use on a royalty-free basis.

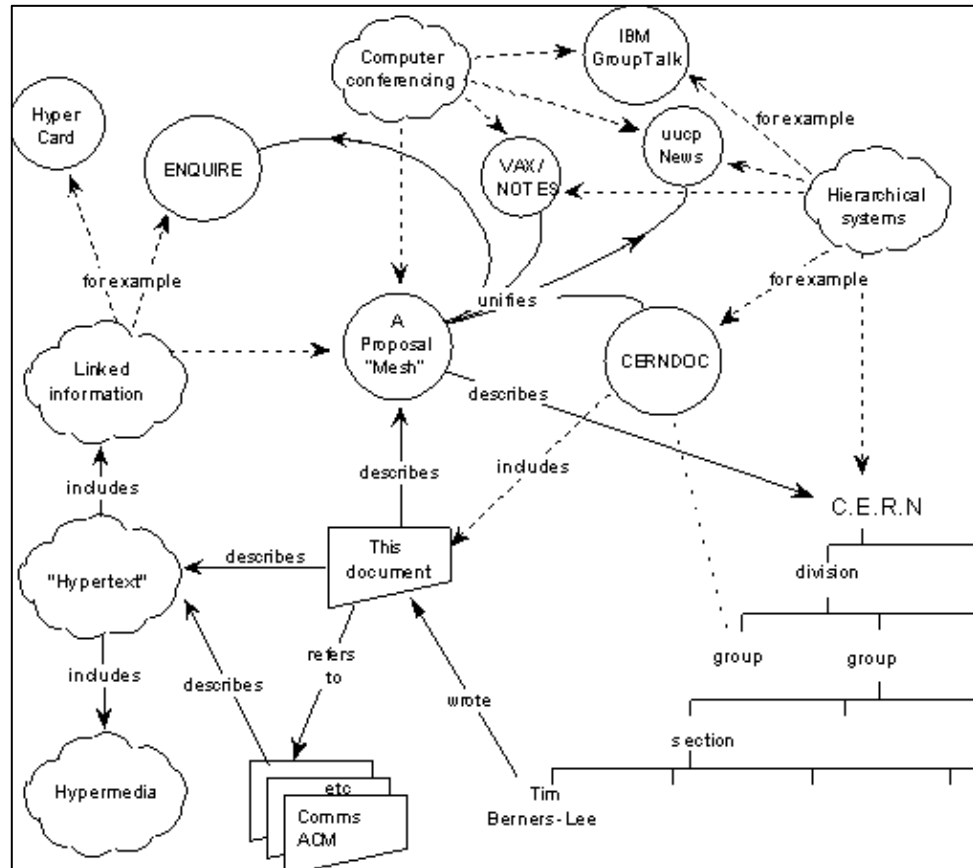


Figure 2. Sir Tim Berners-Lee's proposal on 12th March 1989. Image copied from World Wide Web Foundation [11].

The web started its growth, with engineers and scientists using it, but it was realized that for the web to reach its full potential its underlying technologies had to be standardized. This is how in 1994 Tim Berners-Lee founded the *World Wide Web Consortium* (W3C) to find consensus around the specification and guidelines [11]. What came next were the hobbyists, as well as not quite specialized programmers, who started trying new graphical ways of building webpages. This was the cave man era of the web where everything was wild and free. The web consisted mostly of static sites with small hacks and tweaks to make cool things happen. Some are still online and functioning, such as warnerbros.com/spacejam/movie/jam.htm [14]. As seen these were mostly flashy unreadable pages constructed with tricks and hacks to make everything fall into place.

Even though the Internet has progressed far when looking into the past, there are still frontiers to cover. The 1990's might have been the cave man era of the Internet, but now it is the Wild West with progress being slow. Web programming relies on the tech-

nologies that use W3C's specifications, and unfortunately these are implemented and propagated slowly into browsers and servers. The major difficulties of the twenty-first century are not the lack of specifications, but the absence of their implementations in technologies web programmers rely on. [16.] With the rapid growth of the web with over 1 trillion public pages (in 2008) and 1.7 billion people on the Web (in 2009), and not fully understanding how all the pieces work together, the Web Science Trust (WST) was started. With the future looking promising, as more and more devices are connected, it is still far from reaching its full potential. [11.]

2.2 What is Web Programming Today

Web programming can be split into two major groups: client-side and server-side coding. Client-side coding consists of what is mostly observed about the web and what the user sees clearly in a web browser. These technologies include HTML, CSS to JavaScript and graphics. Server-side coding on the other hand includes technologies such as Java, PHP, Python and Scala. Client-side coding is executed and stored on a local client, whereas server-side code is not available to a client and is executed on a web server. In other words, the server-side is more hidden to the user as code is running elsewhere, while the client-side gets executed on a user's local machine. [10.]

Web based technologies have developed rapidly and the knowledge base is huge when it comes to mastering the art. The importance of the three fundamental technologies that started it all (HTML, URI and HTTP) is crucial. New on the scene are CSS (cascading style sheets) and *JavaScript*. Also the knowledge of cross-browser, cross-platform, cross-device functionality, accessibility, templates, CMS, web *frameworks*, different programming languages, usability and performance testing are part of today's web programming. The world has moved away from hobbyist front-enders to professionals who need to know the nitty-gritty, outs and ins of what it means to develop a front-end interface. It is a great task and a challenging one in the ever-developing industry. In figure 3 one can see the fast development of technologies. This graph is a great example of how programmers have to know at least a little about everything and a lot about nothing.

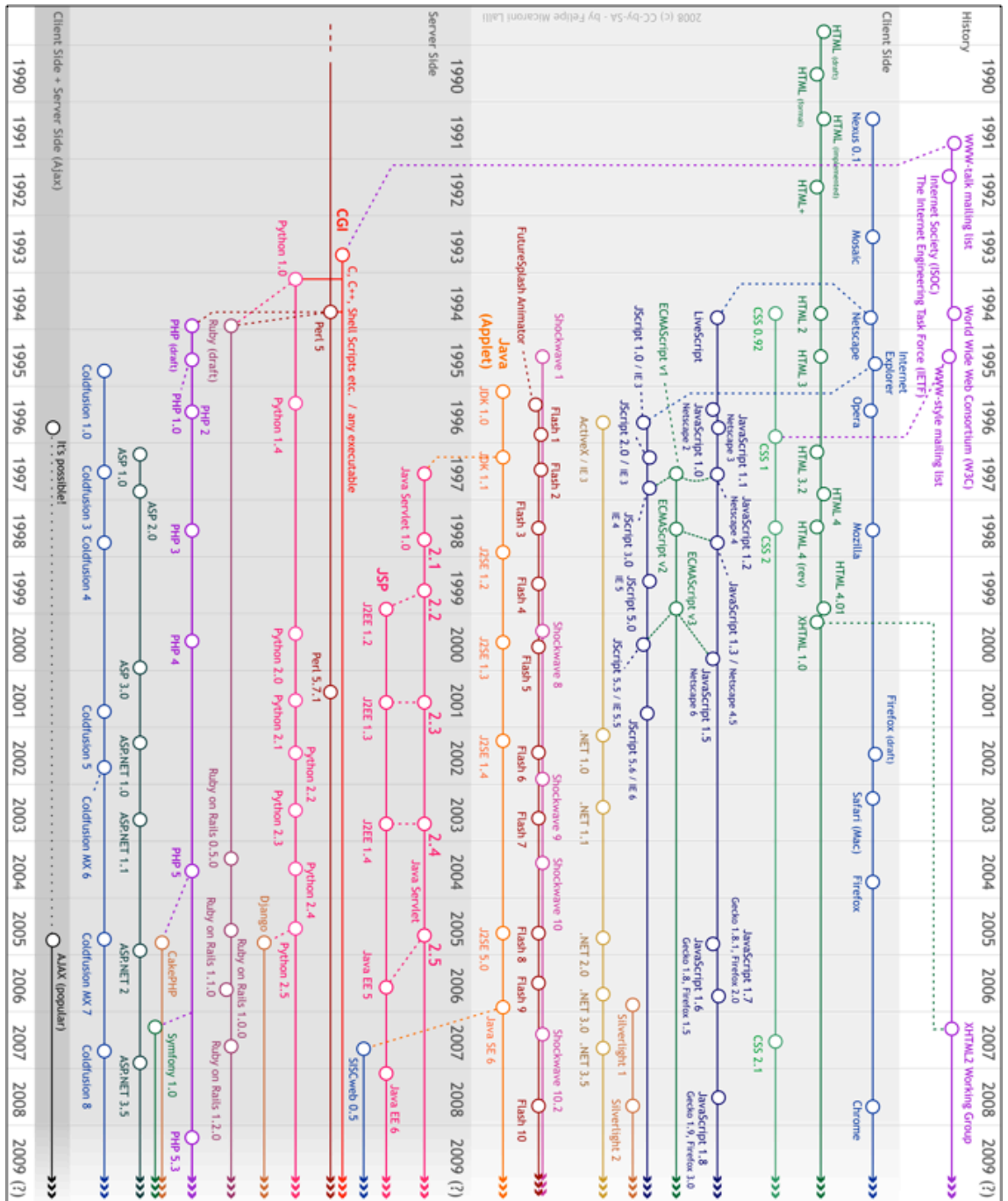


Figure 3. Timeline of web technologies from 1990-2009. Courtesy of Felipe Micaroni Lalli [10].

2.2.1 Mark-up of Today: Focus on JavaScript

There are three key elements or pillars of front-end development today. These can also be called the best practices when it comes to mark-up and coding for the web. Firstly there is the separation of presentation, content and behavior. Secondly, mark-up

should be well formed, semantically correct and generally valid. Thirdly, JavaScript should progressively enhance the experience. [19.]

The first component of any web page is the tag-based mark-up language HTML and it always defines the structure and outline of a document. The second component is the presentation information contained in CSS. CSS describes all presentation aspects of the page via a description of its visual properties. Finally JavaScript code enhances the overall user and browser-based experience through attaching to events and controlling the behavior layer. [19.]

The key difference with the early days of the web to modern times is the amount of JavaScript that can be found on websites. Robert Ward and Martin Smith already predicted this in their paper “JavaScript as a first programming language for multimedia students” in 1998, when they reported experiences and findings on the first delivery of JavaScript modules and the future the web offers graduates: [20, 250.]

“...it seems that that the Web is now becoming the prevalent delivery medium for multimedia products and Web development a main area of employment for multimedia graduates...” [20,250]

JavaScript was introduced in 1995 with the Netscape Navigator browser to add programs into web pages. Even though its name might suggest a connection to Java it has little to nothing in common with it. If anything JavaScript is more like C and similar name was inspired by marketing considerations, rather than good judgment. After the adoption outside of Netscape a document to standardize the language was published called the ECMAScript standard. [22; 23, 857] ECMAScript and JavaScript can be used interchangeably and are names for the same language. There have been many versions of JavaScript, with ECMAScript version 3 dominating in 2000-2010. Propositions on version 4 were made and abandoned in 2008 and today developers are writing version 5 code which is supported by all major browsers (see figure 3). While traditionally JavaScript has been thought of as a client-side language, it can be used in databases, such as MongoDB and CouchDB, for scripting and querying. [21,6.] Netscape introduced an implementation of the language for server-side scripting with Netscape Enterprise Server in December 1994, soon after releasing JavaScript for browsers [45, 2]. Only now after a decade has it picked up interest in implementations such as

Node.js that is a powerful environment for programming JavaScript outside of the browser.

JavaScript is an interpreted, object-based, event-driven programming language embedded into HTML pages. It has a built-in object based hierarchy and it is untyped, so variables and types do not have to be declared. It contains the full range of operators, logical operators and supports functions, but not procedures. “It uses dynamic binding so object references are checked only at run-time... It has a full range simple data types, and supports local and global variables but does not support user-defined data types. It supports the concept of uni- and multidimensional arrays (although elements are actually instances of an inbuilt Array object)...” [20,250.]

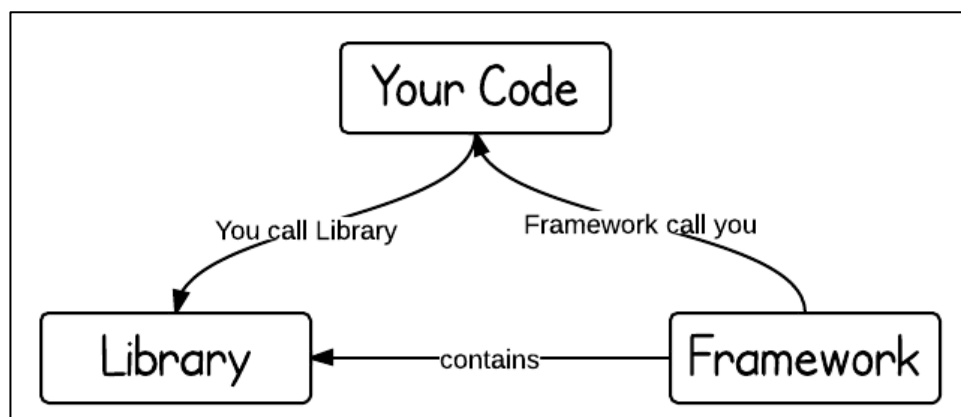


Figure 4: The difference between a library and a framework explained very simply. Copied from programcreek.com [35.]

It is important to note that no one writes pure JavaScript anymore and most programmers rely on libraries and frameworks. This is because advanced JavaScript programming can often be very difficult and time-consuming to work with. For example the complex handling of browser differences is a good reason why a lot of JavaScript (helper) libraries have been developed to deal with these difficulties. The two terms library and framework are overlapping words that have common characteristics. Even though the question is highly subjective, it can be said that frameworks give a structure of "how" programmers should present their code and library is a toolkit, which highly abstracts different layers [34]. Simply put a library slots into an existing architecture and adds specific functionality, while a framework gives an architecture such as a file structure that is meant to be followed and as such meant to handle all common requirements [26]. The difference can also be defined by the “inversion of control”. This means, when calling a library, the coder is in control. However with a framework, the

control is inverted: the framework calls the coder, (see figure 4). All the control flow is already in the framework, and there is just a bunch of predefined white spots that can be filled out with own code. A library on the other hand is a collection of functionality that can be called. [35.] In this study libraries have been used and some of the most popular JavaScript libraries are *jQuery*, Prototype and MooTools. All of these have functions for common JavaScript tasks like animations, *Document Object Model (DOM)* manipulation, and Ajax handling. In section 4 it is explained how jQuery and D3.js have been used to implement functionality within this project. [27.]

2.2.2 Best Practices in JavaScript

Even without being an advanced programmer there are some best practises to take into account when writing JavaScript. Many opinions are out there and search results come up with multitudes of links. The reading of credited sources such as w3schools, code.tutsplus and some frameworks readme's are a great starting point. There are some pillars of practices that include avoiding global variables, declaring local variables, placing declarations on top, initializing variables, never declaring numbers, strings or Boolean objects, avoiding the use of `newobject()`, being aware of JavaScripts quirky automatic type conversions and its funny `'==='` comparison instead of `'=='` comparison operator that converts (to matching types) before comparing. Also using parameter defaults avoids breaks in code. Ending switches with defaults and avoid using `eval()` altogether is good practice. [37.] *Tuts+* has an even longer list of good points and explanations for each in their "24 JavaScript Best Practices for Beginners" post from June 2009 [39]. Isobar's "Front-end Code Standards & Best Practices" also offers a comprehensive guide to overall best practices in HTML, CSS and JavaScript. [19.]

2.2.3 Layout and Design: Focus on Designing for Data

Equal to code is the layout and design of a webpage. The word design comes from the Latin word *designare*, which means to mark out, devise, or choose. Designing for web is its own unique ballgame when compared to print. Print involves seeing, while web is seeing the doing. This means that the responsibility of a web designer is to lead the user through the website that is the doing. "The experience of a website is defined by the interaction the user has with it." [28, 8.]

The design process can be defined in six stages: defining goals and strategy, research, information architecture, sketching, *wireframes* and *mock-ups*. Defining goals and strategy involves asking questions such as “Why does this website need to exist?” A designer should be able to define the goal of the website. Research is done to understand what visitors to a site might be expecting and it can be helpful to create scenarios and characters for the process. Information architecture goes deeper into designing the structure of a website before considering the visuals. This is providing optimal navigation paths in getting from point A to point B intuitively. In this stage developers come to early usability testing and it runs all the way to wireframes and mock-ups. Wireframes, prototypes, and mock-ups are to explore different design options and functionality for the site. It allows making changes before writing code and creating graphics. This is important as no one wants to write code again realizing its un-usefulness. [28, 11-20.]

Even though web design is thought of as being completely separate from traditional design, it still has basic elements that follow it closely. For example, just like any other form of design, there is a grid, which is an organizing principle in graphic design. It is ingrained into current practice and design education. In “Making and Breaking the Grid”, by Timothy Samara, various grid layouts are explored, which have been used for centuries and are still as fresh and used in modern day websites. Bootstrap, a CSS library, is an example where a 12-grid layout and its knowledge is key to building websites with the technology. [29, 8-9,88-89.]

To design means to plan. The process of design is used to bring order from chaos and randomness. Order is good for readers, who can more easily make sense of an organized message... good design ... changes with time. It is apparent that style and fashion are aspects of design that cannot be ignored. [30,1.]

As in the legendary design bible “The Elements of Graphic Design” all the traditional elements can also be seen in online websites. These include understanding space, unity, page architecture, color and type [30]. To be able to design great sites, it is crucial for a web programmer to also understand something about design even if they have not gotten a degree in it. The fundamentals of design come into play when coding any project meant be of convenience to its users. “But the success of a page is only as good as the power with which it communicates and the effortlessness with which it does it.” [30, 201.] As a great example of creativity and following traditional graphic

design principles James Cheshire mapped the world's population in figure 5. "Omitting any shorelines or country boundaries, he drew horizontal latitude lines that zigzag upward in black according to the population at that spot, with spikes at major cities colored gold. The approach highlights Earth's densest areas—crowded Asia stands out particularly—more clearly than a table of numbers ever could." [40.]

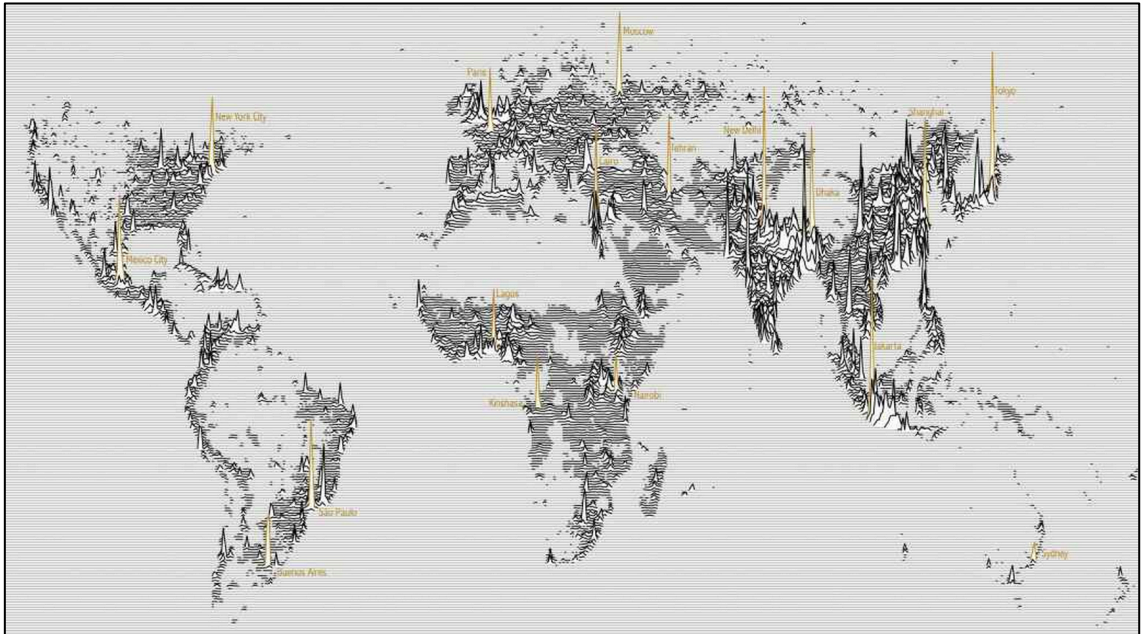


Figure 5: James Cheshire mapped the world's population in a new way. Copied from popular Science (2014) [40.]

In a 2014 study on how designers design for data it was found through observation that "designers would first draw high-level elements of their design such as the layout and axes, followed by a sketching in of data points based on their perceived ideas of data behavior" [31, 19]. What was most interesting was that the designers' inference about the data was often inaccurate. As would be expected, to design well for data designers have to spend a substantial amount of time in understanding its intricacies. This was also visible in the study as "the amount of data exploration and manipulation was related to the level of a designer's experience working with data". Inexperienced designers "data encodings remained unchanged until assumptions about data behavior were shown to be incorrect". [31, 20.] It was noted that designers have a critical need to be able to define and modify data abstractions, but unfortunately these types of tools are still in their infancy. Currently the only way to minimize ineffective or incompatible visualizations in the planning and implementing phases is by understanding data behavior and structure early on in the design process. Unfortunately this can be tough as visual-

ization creation tools, for example NoPumpG, SAGE and Tableau, make importing data easy, but tend to limit the design space. Visualization programming environments, for example Processing, D3.js, Protovis, VTK, can produce sophisticated, flexible, and creative visualizations, but they require skill and time to learn. [31, 19.] Currently the need for “data abstraction tools that attempt to provide the freedom to explore alternative interpretations of data, including alternative data structures and deriving new data” is missing in the arsenal of tools for creating data visualizations for today’s audiences. [31, 22] This might be the reason why data is still represented in the static representations forms as expressed in the introduction of this paper.

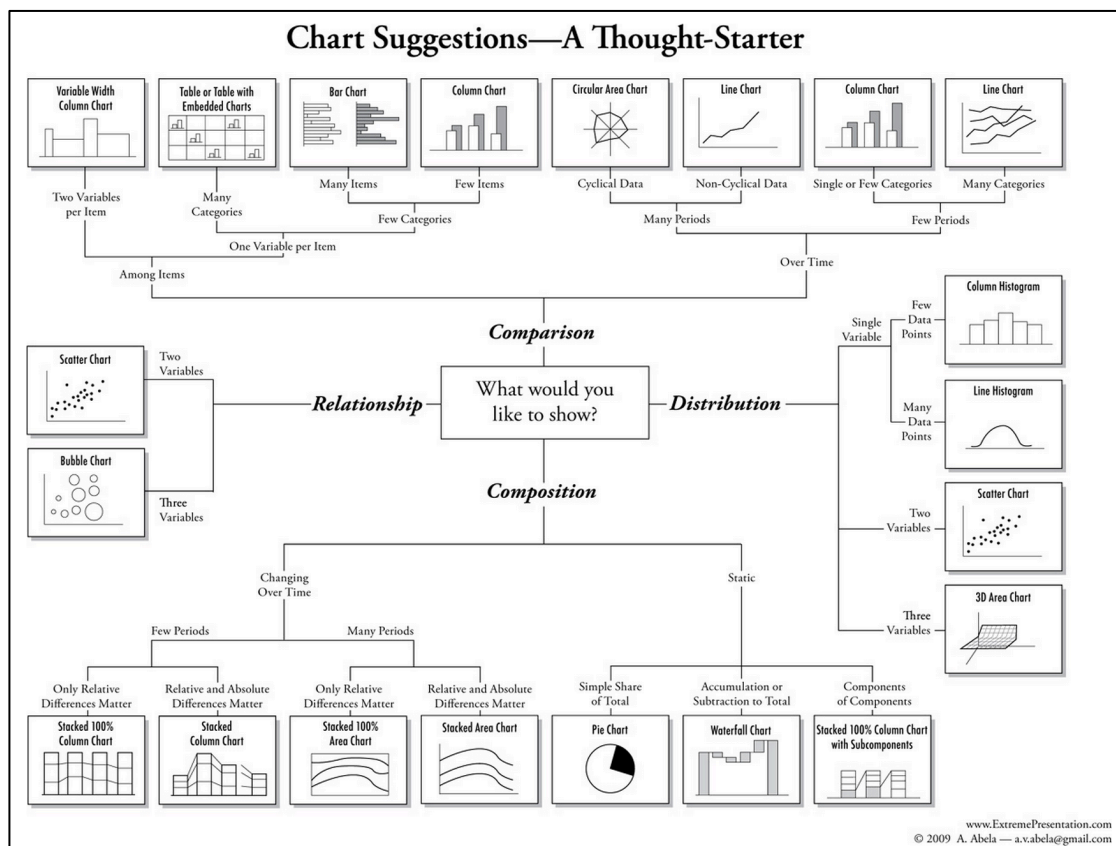


Figure 6: Diagram to help in the choosing of a form. Copied from King (2014) [2], Created by Dr. Andrew Abela, marketing professor and dean, Catholic University of America in Washington, D.C.

A good quote about the importance of data visualization as expressed by Edward R. Tufte in *The Visual Display of Quantitative Information*:

Excellence in statistical graphics consists of complex ideas communicated with clarity, precision, and efficiency... Graphics *reveal* data. Indeed

graphics can be more precise and revealing than conventional statistical computations. [32, 13.]

There are some key elements to designing good visualizations with data. The visualization should show the data as best it could. It should “induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production” or anything else. It should not distort what the data has to say and present many numbers concisely and make large datasets coherent. Good visualizations encourage the eye to compare different pieces of data and reveal the data at several levels of detail, from overview to fine structure. What is shown has to serve a clear purpose with description, exploration, tabulation or decoration. It also needs to be “closely integrated with the statistical and verbal descriptions of a data set.” [32, 13] Interestingly enough even in 1982 John W. Turkey realized that the lack of quantitative skills of professional artists was a cause for graphics that lie. This is because mostly designers, with education in fine art, have done this task, generally with no experience with the analysis of data and statistics. [32, 79.] The core skill is the knowledge of raw mathematics along with a visual eye. Discerning among the multitude of statistical methods requires knowledge and learning how different types of data behave and are best represented. Dr. Andrew Abela’s model in figure 6 helps designers in choosing a form for data, but the rest is up to the knowledge of the designer. To make the most use of the diagram, Lauri Nummenmaa’s book on statistical methods explores distinct data analysis methods and calculations. It explains the idea behind statistics from collecting relevant data in order to make predictions of the population. It also explains what is a significant enough dataset that it can be compared to its real population. It goes into dependent and independent variables in datasets and what is significant enough correlation continuing to explain statistical mathematical models and probabilities. As an example, in section 4 that describes the thesis project in more detail, two curves have been displayed to give the user an opportunity to observe two sets of data in one graph. [33, 3-10,284.]

2.3 Reasons for Choosing D3.js in this Thesis Project

D3 was published in 2011 and it quickly becoming a popular tool for creating data visualizations online, but it was not the first or only tool out there. Most importantly I chose D3 for the project as it had a number of example code blocks and books available. It

was important to be able to have some references and ideas on what was possible to accomplish when it came to making online data visualizations. This was the first time delving into this subject and as such needed all the support I could get. Also choosing a challenging project was personally important and D3 was definitely this with the flexibility of doing everything and customizing where needed. The other options out there that could battle with D3js included less known and newer libraries such as zingCharts, dygraphs and highcharts. Thinking now highcharts and zingCharts would have been better choices for this project to create the most amount of interaction and animation within the graph with less writing. Interactivity is inbuilt in these libraries while in D3js it has to be written. [59.] Nevertheless D3 ended up being a good challenge for the amount of literature on the topic. Other choices included jqPlot, jpGraph, Crossfilter, Flot, and Raphaël. jqPlot was small and did not have much to offer when it came to customization, jpGraph was PHP based and since this was not a strong skill it was marked off. Crossfilter is an extension library for D3 to explore large multivariate datasets and therefore would still require building the visualization in D3. Flot is modest with their documentation being very limited and example base small. Raphael is a small JavaScript library that works with vector graphics on the web, but it also lacked enough examples. Table 1 below shows the most important features in each when compared to the needs of the thesis project. A more thorough chart can be found online. [59.]

Table 1: Chart on JavaScript data visualization libraries (mentioned in this section). jpGraph not given values because it is a PHP library and Crossfilter is an add-on. Modified from Wikipedia [59].

Library	Line graph support	License	Interactivity		Rendering Technologies		
			Mouse over	on-Click	HTML Canvas	SVG	VML
D3	Yes	BSD-3	Yes	No	No	Yes	No
zingCharts	Yes	Free with a link	Yes	Yes	Yes	Yes	Yes
dygraphs	Yes	MIT	Yes	Yes	Yes	No	No
highcharts	Yes	License required	Yes	Yes	No	Yes	Yes
jqPlot	Yes	MIT or GPLv2	Yes	Yes	Yes	No	No
jpGraph	-	-	-	-	-	-	-

Crossfilter	-	-	-	-	-	-	-
Flot	Yes	MIT	Yes	Yes	Yes	No	No
Raphaël	Yes	MIT	Yes	No	No	Yes	No

For now D3 is the most known library and noted for its flexibility. It works seamlessly with existing web technologies, and can manipulate any part of the document object model (DOM). It is as flexible as the client side web technology stack (HTML, CSS, SVG).

“This gives it huge advantages over other tools because it can look like anything you want, and it isn’t limited to small regions of a webpage like Processing.js, Paper.js, Raphael.js, or other canvas or SVG-only based libraries.” [60.]

What D3 has is great documentation, examples, community, and the accessibility of Mike Bostock. These have all played major roles in its rise to prominence. With all great libraries there are also drawbacks. One major one is, also clearly visible in this thesis project, the DOM manipulation being extremely slow for large numbers of entries and SVG performance limitations when dealing with large quantities of elements. Another handicap is the high learning curve, but with great community support, learning is much easier than with less known and documented libraries. To conclude D3 is not “a graphics library, nor is it a data processing library. It doesn’t have pre-built charts that limit creativity. Instead, it has tools that make the connection between data and graphics easy.” [60.]

3 D3.js

The D3 library is extensively used in the project that is part of the 4th section of this paper. It was first developed by Stanford graduate student Mike Bostock in 2011, along with his advisors Jeffery Heer and Vadim Ogjevetzsky. The library is a freely available extension of JavaScript and the basic idea behind D3 was to provide a way to join data with elements on a web page and then manipulate the elements based on that data. [2.] The library enables the developer to directly set the attributes of graphical elements in SVG (vector based graphics written with code) according to data [2, 6]. It uses SVG, HTML5, CSS and JavaScript to power visualizations (see figure 7).



Figure 7: The core web based technologies that power working with D3 and make it an easier tool to learn [60].

It is important to comprehend what D3 is not a tool for. It is not the tool for figuring out the best method of displaying data or the story a data has to tell. Only after figuring out the data set, its intricacies and story does the library come in handy. Also the library is not for scraping data off the web. For such work Python has a couple of libraries that work. This tool is also not good for sorting through data or doing basic data analysis. Because D3 has a high learning curve, it is not built for quickly making bar charts, maps, or other standard charts. The tool was designed to be powerful and robust, thus to be used in creating multitude of graphical forms and to have full flexibility in the customization process. [2, 8.]

Also important is to note that D3.js is there to help tell a story. Section 2.2.2 explains the process of creating a story from information with layout and design. To design with data is to design data in mind. The steps to creating an amazing looking data visualization are for example the following: 1) Ask a question that can be answered with data. For example, how long is the average lifespan of people in the world? Or, how much on average are nations in debt around the world? There are many places to look for data

and some of these are the United Nations, the World Bank, the International Monetary Fund, and the Organization for Economic Co-operation and Development (OECD). 2) Gather the data into a usable format and prepare it for processing. This might involve scraping data from an existing API or from Excel charts and making sure there are no missing values because those are hazards for analyzing or visualizing a given dataset. This is called data cleansing and some techniques include examining the validity of measurements, accuracy of a measurement, syntax errors in measurements, missing values, consistency of measurements making sure that they fall between a reasonable space and uniformity (measured in same units). There are many more methods and tools to use in searching for these data mishaps. [41.] 3) Pre-analyzing data to see what graphing model is best suited (see figure 4 and section 2.2.3 for reference). Also model the data quickly in other tools (for example Excel) to see a glimpse of it in different formats. 4) Start to plan the visualization and possible animations that are wanted in order to help the future user understand the story of the given dataset. 5) Work in D3.js starts and the coding begins. In the sections that follow D3 is shortly gone through.

3.1 Methods in D3

D3.js offers a lot of different methods in working with data. There is a list below taken from the libraries API reference pages on *github* [42]. Behaviours allow dragging and zooming of DOM elements created by D3. The core is where the data is placed into selected DOM elements, where transitions are created, data loaded, colors added to the visualization and all the standard techniques used in getting the data onto the HTML page. Geography, as the name suggests, is part of map visualizations. Geometry can create voronoi-, quadtree-, polygon- and hull layouts. Layouts can create several standard graphs without much coding on a programmer's part. There is a standard histogram layout, pack (recursive circle-packing), partition (node tree into a sunburst or icicle), traditional pie chart, stack, tree, and treemap (display a tree of nodes). [4.] Scales lets a developer work with continuous input domains, such as numbers, discrete input domains, such as names or categories or time domains. SVG binds visual elements to data and the DOM. The time method has its own group as it allows, for example, time format conversions and placing of intervals.

- Behaviours — reusable interaction behaviors

- Core — for example selections, transitions, data, localization, colors
- Geography — project spherical coordinates, latitude and longitude math
- Geometry — utilities for 2D geometry, such as Voronoi diagrams and quadtrees
- Layouts — derives secondary data for positioning elements
- Scales — converts between data and visual encodings
- SVG — utilities for creating Scalable Vector Graphics
- Time — parse or format times that can compute calendar intervals

On the API pages developers can also find how arrays are used, about bundle layouts, chord layouts, cluster layouts and about colors. There is information also about namespaces, importing data in different formats (json and CSV), math that can be done on data and even tutorials.

3.2 Vector Graphics: Focus on SVG

There are two types of graphics: vectors and rasters. A raster image is made up of pixels that can only be of a certain color and that together form an image in known formats such as JPG, GIF and PNG. A good way of testing if an image is a raster image is by zooming into it. If it starts to become “pixelated” the viewer start to see the building blocks of squares of one color. Then it is a raster image. A vector image is a mathematical model that tells a computer what shapes it contains. It is the computers task to tell the pixels on the screen to render themselves according to these instructions. Therefore a user can zoom into a vector image indefinitely without any pixilation. [2.]

```
<svg width="500px" height="500px">  
  
  <circle cx="120" cy="50" r="20" fill="darkmagenta" stroke="black" stroke-width="5"/>  
  
  <circle cx="15" cy="50" r="10" fill="darkmagenta" stroke="black" stroke-width="5"/>  
  
  <circle cx="60" cy="100" r="50" fill="darkmagenta" stroke="black" stroke-width="5"/>  
  
</svg>
```

Figure 8: creating three SVG circles. Copied from King (2014) [2]

SVG is a vector format that has been standardized for the web and works on all major browsers except for Internet Explorer 8 or earlier. It can be written in a human-readable programming language that is a lot like HTML. It contains opening tags and closing tags, parents, children, and in fact when building a webpage, SVG can be inserted directly into the HTML and the graphic will appear in the browser. In figure 8 above illustrates an example snippet of how an SVG circle can be created. Ellipses, polygons, lines, paths, and SVG text can easily be created. Also SVG has a slew of style properties that can be used for designing graphics. Some of these are color and transparency, stroke properties, drawing order and groups and transformations. Because of its web standard status SVGs can be assigned classes and then styles in CSS. [2.]

3.3 D3 Selections

To place data into a web page, D3 has to be given an element to select and place that information into. There are two methods for this: `d3.select()` and `d3.selectAll()`. `d3.select()` selects one element at a time and creates a selection out of the first match it finds in the DOM. Importantly D3 always employs CSS selectors to make selections, and the created SVG elements should also have CSS selectors to style and select them. Selections return arrays that can be assigned to variables. D3 uses the same type of chaining as jQuery making it easier to do multiple things in sequence to a selection. It is also possible to append and remove selections. The method that is more used than any is `d3.selectAll()` and it creates a selection of multiple elements. The power lies in the fact that it can create a selection of elements that does not exist. Selections only become useful when they are used with Data-Joins.

3.4 Data-Joins

Data-Joins is joining data with something and these are elements on a web-page. When using the magical combination of `data()` and `enter()` on a selection, it creates a new object for every point in the dataset (see figure 9 for a diagram of the methods in action).

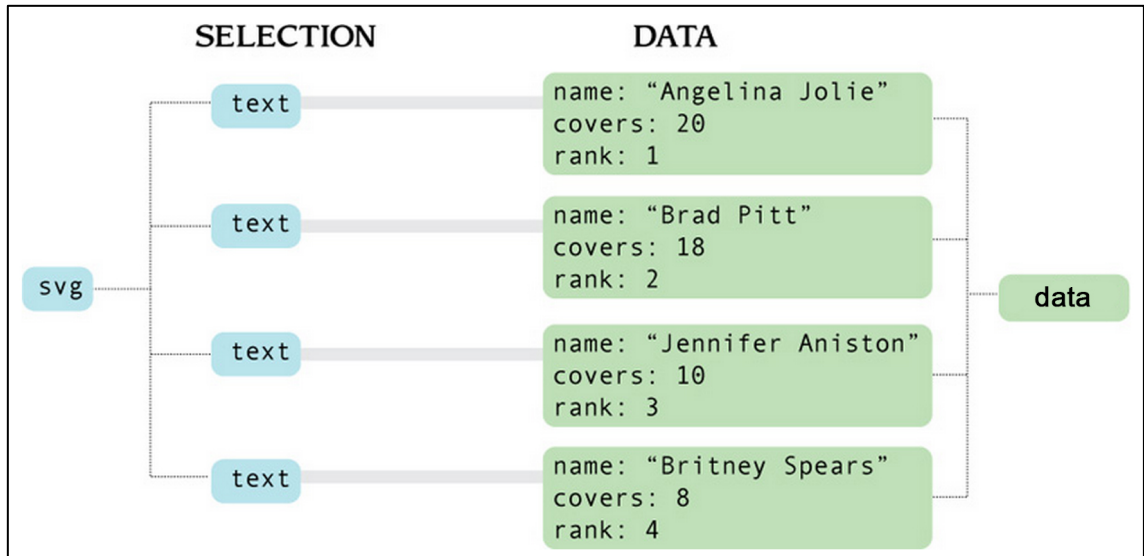


Figure 9: Data-Joins in action. Modified from King (2014) [2]

To create even more powerful code, a programmer can use anonymous function to access bound data. There are conventions when using anonymous functions to access this bound data. Firstly an argument always takes an argument `d` and it has a special meaning. For each of the elements in a selection, `d` represents the bound data point. D3 also gives access to the index of each data point within the data array by introducing another argument to the anonymous function - `i`. These methods are at the heart of any D3 project. [2.]

3.5 Scaling and Axes in Charts

Scaling is convenient in D3 as it allows setting up a function that essentially maps data to pixels. These scales can be linear, logarithmic or ordinal (mapping non-numeric values to pixels). Even though most math can be done in pure JavaScript and scales are not necessary, they make programming easier. As an example, if specifying `d3.scale.linear().domain([0,10]).range([10,100])` it means to take an array inside `.domain()` and mapping it to the array in `.range()` in a linear way. Figure 10 shows this graphically. The range scales the data points so they are displayed on the chart from minimum to maximum according to the element they are to place into. The domain is the minimum and maximum of the actual dataset.

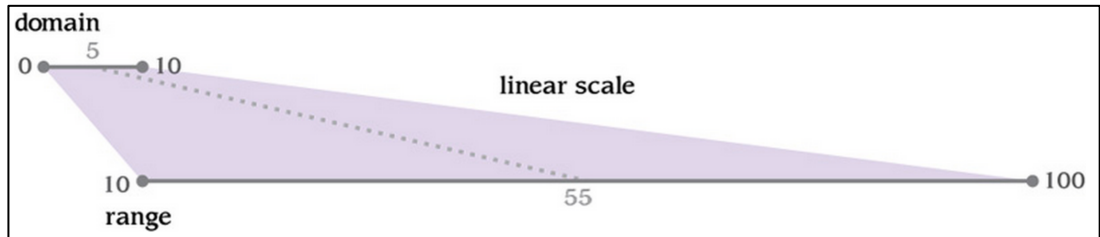


Figure 10: Scaling an array by domain and into a range linearly. Copied from King (2014) [2]

To keep data neatly organized, smart margin conventions are used. To create these there is a method with four properties. The SVG element that contains all the data points is given margins in terms of width and height. An example of this can be found in the API references for D3 in github, and figure 11 tries to explain it in a graphical format.

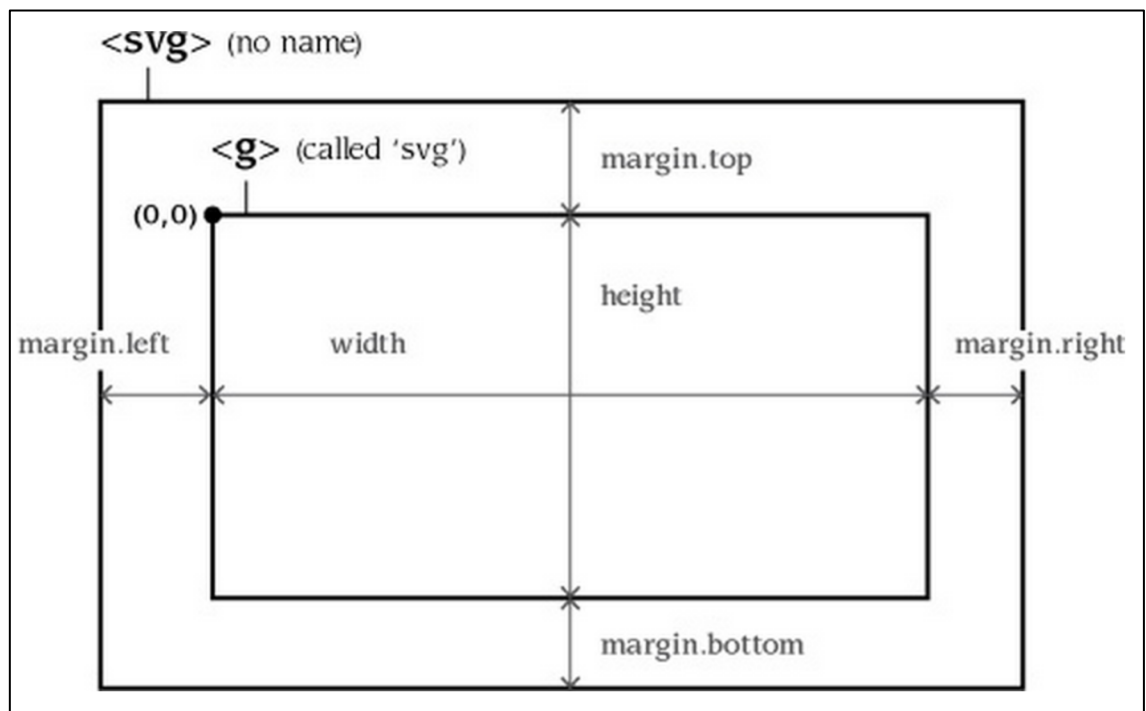


Figure 11: margin convention in D3. Copied from: King (2014) [2]

The axes in D3 are created by calling the generator for axes. To do this a function is defined and initialized with `d3.svg.axis()`. The axes can be customized and there are plenty of ways to do this accessibly in the API references. Finally to add ordinal scales use the same `domain()` and `range()`, but instead apply them to `d3.scale.ordinal()`. With

the `map()` method in JavaScript it is possible to act on arrays and reorganize them or even extract subsets of them with relative ease, dumping them into a new array. [42.]

3.6 Loading External Data

D3 can handle many formats for data except the richly formatted spreadsheet files, `.xls` and `.xlsx`. Below is a table of the formats excepted by D3. In section 4 `.json` is used, and the method to get to the data is `d3.json()`. To access these datasets on a local machine a developer needs to set up a local server. D3 interprets that data, and processes requests asynchronously. [2; 42.]

Table 2: Table showing the different formats of data that D3 is able to parse.

Format	Function	Description
<code>.txt</code>	<code>d3.text()</code>	Plain text file
<code>.csv</code>	<code>d3.csv()</code>	Comma-separated values
<code>.tsv</code>	<code>d3.tsv()</code>	Tab-separated values
<code>.json</code>	<code>d3.json()</code>	JSON blob
<code>.html</code>	<code>d3.html()</code>	HTML document
<code>.xml</code>	<code>d3.xml()</code>	XML document

3.7 Adding interactions and Animating

This is what makes this technology different from earlier generation graphing methods. It allows the display of data in a canvas on a web page and this technology is fundamentally interactive. This means that basic user interactions can be introduced using standard browser events. [43.] To specify animations (or other transitions) are done with the `.transition()` method, followed by a chain describing the end result. It can include any DOM transformation and the only things to worry about are “the values to transition from and to, the duration of the animation, and any delay that should happen before the animation starts.” [44.]

4 Creating a One Page Web Application

4.1 Methods and Materials

For this thesis paper a one page application was created in order to test the current technologies involved in producing interactive data visualizations. The application was constructed in six distinctive phases. Firstly, a proper dataset had to be found and somehow data displayed onto the viewport of the browser. Second, the data was constructed into a format that can be read by D3js and inputted into a D3 SVG element creating a graph. Thirdly, multiple datasets had to be plotted onto the built JavaScript graph function. Fourth, came the building of interaction and the logic of clicks and transitions from one state to another. Fifth, was the styling of elements and the built of interactions to be responsive on a wide range of devices. The sixth and final step was putting everything together, deploying everything onto a server with any finishing touches the app needed. [47.]

The project was built using D3js, but also employing jQuery and *Twitter Bootstrap* where needed. Google fonts along with Bootstrap gave a simple stylish layout. Social media buttons in the footer where provided in order to get questionnaire responses from the general public about the visualization application. CSS3 animations where also used and HTML5 written as the *DOCTYPE* entails in the main index.html. For the execution of all these technologies a development environment had to be built to house sufficient testing and debugging utilities. The terminal was used for initiating a local HTTP server as well as keeping version control. A gitHub repository was created to have access to the project from any machine. Coding was done mostly in *Sublime Text 3* with the help of *linters* for debugging. Custom JSON, JavaScript and CSS were the building blocks and the backbone was with the known technologies of D3, Bootstrap and jQuery.

4.1.1 Phase One: Researching for a Dataset

In phase one data was scouted for, researched and a prospective plan on how to display it was contrived. This was a time consuming process that involved exploring the Internet for open data and for figures that would be relevant and interesting to a wide

audience. Many datasets like CREO's extinction of species dataset were a prospect, but in the end these were left to be devoured for a later date [6]. When coming across Bintanja and van de Wal's (2008) scientific report of 3,000,000 of years global temperatures according to Ice Volume ($\delta^{18}O$) [8], it seemed perfect for an interactive visualization. Not only was it challenging to understand, but it also seemed the kind of data a common person might be interested in.

Now only if it was in a format that could be understood.

If searching for data was a challenge, converting it into an understandable visualization, and planning the intricacies around how it could be implemented with current technologies, was even harder. Discerning the data implied reading up on geography, glaciology, climatology and sedimentology. The dataset implies an understanding of the fields and goes deeper into earth's historical eras and phases. The measurements were based on a "comprehensive ice-sheet model and a simple ocean-temperature model", that were "applied to marine benthic oxygen isotopes (LR04 stack)". [8] This allowed the extraction of three-million-year mutually consistent records of surface air temperature, ice volume, and sea level. To create a more simplistic view of the large dataset of 30,000 data rows only surface air temperature (*Tsurf (changed to sTemp in json)*) and deep-ocean temperature (*Tdo (changed to tDo in json)*) were used. It is to note, "the reconstructed atmospheric surface air temperatures apply to all subarctic to arctic land masses (including continental shelves) north of about $\sim 45N$." [8] In figure 12 we can see core samples that are 1 million years old and see clearly changes in climate in the hues of the ice. Sediment core, taken with a gravity corer by the research vessel POLARSTERN in the South Atlantic; light/dark-colored changes are due to climate cycles of the Quaternary; basis age of the core is about 1 million years (length of each segment is 1 meter).

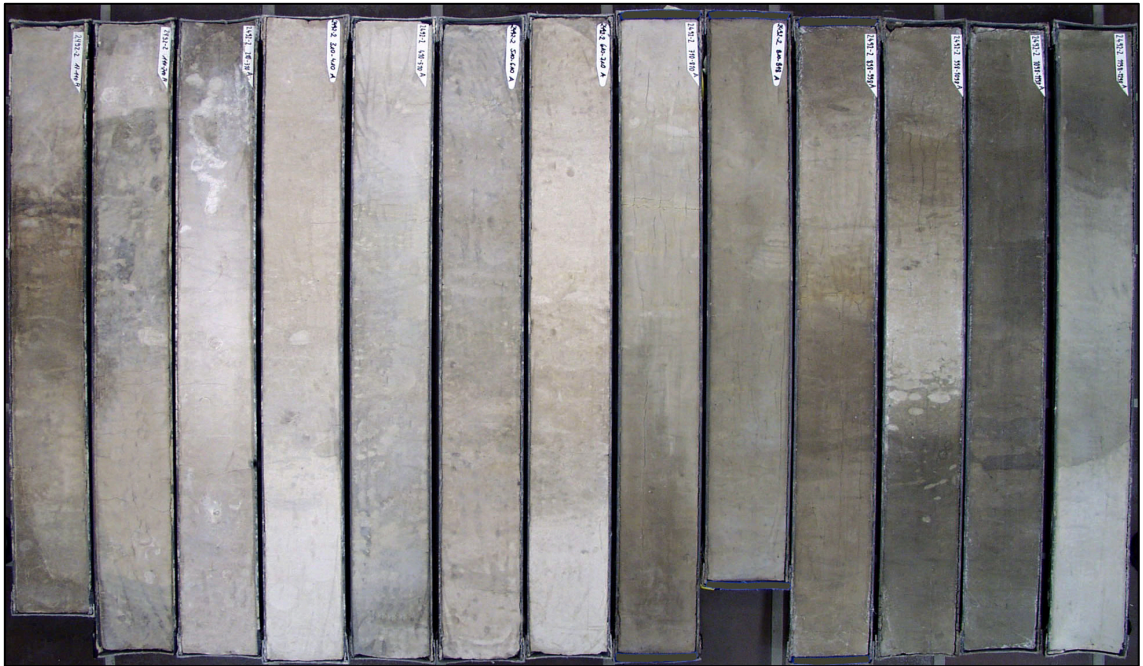


Figure 12: Sediment core, light/dark-colored changes are due to climate cycles. Taken by: Dr. Hannes Grobe, Geosciences (Marine Geology and Paleontology) [48.] Copied from Wikipedia [49]

Marine benthic oxygen isotope (LR04 stack) is part of the marine isotope stages (MIS) and tells about the alternating warm and cool periods in the earth's paleoclimate. These stages are deduced from oxygen isotope data from deep-sea core samples. Stages are either even numbers with high levels of oxygen-18 representing cold glacial periods or odd numbers representing warm interglacial intervals. Estimations of climate “are derived from pollen and foraminifera (plankton) remains in drilled marine sediment cores, sapropels, and other data that reflect historic climate; these are called proxies.” Cesare Emiliani’s pioneering work of the 1950s gave rise to the MIS timescale. It is now widely used in archaeology and other fields to express dating in the Quaternary period (the last 2.6 million years), as well as providing the fullest data for paleoclimatology or the study of the early climate of the earth. [49] The LR04 stack measures the ratio of stable isotopes, $^{18}\text{O}:^{16}\text{O}$ (oxygen-18:oxygen-16), from corals, foraminifera and ice cores. The definition is, in “per mil” (‰, parts per thousand):

$$\delta^{18}\text{O} = \left(\frac{\left(\frac{^{18}\text{O}}{^{16}\text{O}} \right)_{\text{sample}}}{\left(\frac{^{18}\text{O}}{^{16}\text{O}} \right)_{\text{standard}}} - 1 \right) * 1000\%$$

Here the standard is a known isotopic composition. [50.]

After understanding the scientific dataset a way to exhibit data on a webpage had to be found. This involved researching about the methods involved in parsing data in D3. Section 3.6 describes the different methods. After trying out a few, the top candidate became JSON for its flexibility and ease of use with JavaScript. In the next section and in figure 13 the intricacies of employing this format onto the given dataset is explored.

4.1.2 Phase Two: Formatting the Dataset

The large dataset was studied in Excel format in the first phase, but in order to manipulate it in D3 it had to be converted into a format that can be parsed. This became the next challenge. The dataset had to be edited with only the relevant inputs intact and produce a valid output. With over 30,000 rows of data and 9 columns, this was not a task to be done by hand. Conversion was going to involve time no matter what format was picked. In the end because JavaScript natively reads JSON, it became the format of choice. In section 3.6 and figure 12 alternate ways of inputting data into D3 are illustrated. Fortunately an online tool for converting data into one of several web-friendly formats (including HTML, JSON and XML) was found. [55] Also it was helpful that its clipboard memory was not restricted and could fit the entire dataset. With it, valid JSON was produced into a single object file with 30,000 rows.

```
<script type="text/javascript">
//load external data:
d3.json("../json/30000years_of_temperatures.json", function(error, json)
{
  if (error!=null) return console.warn(error);
  dataSet = json;
  visualizeit(dataSet);
});
function visualizeit(dataSet){
d3.select("body").select("div").data(dataSet).enter().append("span").text(function(d, i){ return d.Tdo });
};
</script>
```

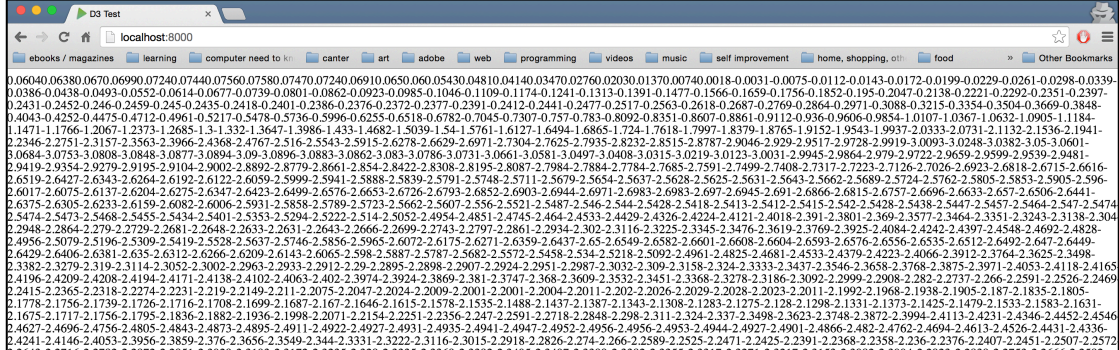


Figure 13: Phase one code where testing of loading of a dataset was rendered onto a page. Screen capture of first initial code commit [47.]

This large JSON was now going to have to be loaded from an external file because it was impossible to place it directly into the HTML page with 30,000 rows. Because of “same-origin policy” restrictions that are built into modern browsers, loading JSON from an external file required the running of a local server. When trying to develop locally these security restrictions will open a security exception. Currently there are two ways to solve this: 1. Change security for local files in a browser (access page as file:///example) or 2. Run files from a local server (access page as http://localhost/example). [36.] For this project the second method was used. The running of a local Python built-in HTTP server was executed with a terminal calling “python -m SimpleHTTPServer” in the project repository directory. After this it seemed to print out nicely all the elements within the file and the output looked like one long paragraph without any spaces or punctuation (see figure 13, above).

As JSON was nicely being printed from the external file, it was time to move to more challenges. Next came the implementation of an SVG graph with only one column of

data within the given JSON file. Next D3 needed to know where in the object to take the data from and what domains to give it when plotting. This was a big learning curve and several hours were spent reading example code and monkeying code in. In the end revelation struck with the result of a single line graph with data used from deep ocean temperatures. [51.]

```

...
var line = d3.svg.line()
  .x(function(d) { return x(d.Time); })
  .y(function(d) { return y(d.Tdo); });

var svg = d3.select("body").append("svg")
  .attr("width", width + margin.left + margin.right)
  .attr("height", height + margin.top + margin.bottom)
  .append("g")
  .attr("transform", "translate(" + margin.left + "," + margin.top + ")");
...
function visualizeit(data){
  x.domain(d3.extent(data, function(d) { return d.Time; }));
  y.domain(d3.extent(data, function(d) { return d.Tdo; }));

  svg.append("g")
    .attr("class", "x axis")
    .attr("transform", "translate(0," + height + ")")
    .call(xAxis);

  svg.append("g")
    .attr("class", "y axis")
    .call(yAxis)
    .append("text")
    .attr("transform", "rotate(-90)")
    .attr("y", 6)
    .attr("dy", ".71em")
    .style("text-anchor", "end")
    .text("Temperature");

  svg.append("path")
    .datum(data)
    .attr("class", "line")
    .attr("d", line);

```

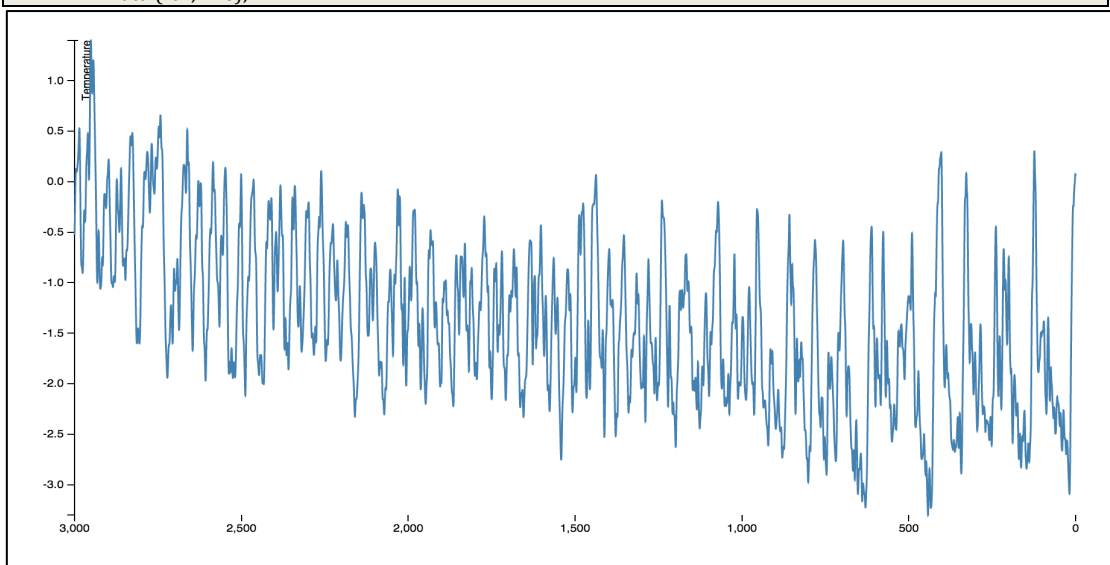


Figure 14: Data printed onto an SVG element with the help of D3 and code clipped from the relevant part. Screen capture of the second phase [47.]

As seen in figure 14 the data displayed nicely when D3 was given a clear understanding where to pick up the data points. Here a variable line is told that x is Time and y is deep ocean temperatures. When loading the data the SVG element is told to rotate on its axis -90 degrees to compensate for the fact that in computer graphics the origin is always the top left corner. For humans origin is displayed in the bottom left corner. The result was as in figure 14 a line graph with the correct axis representing the domain of deep ocean temperature values of 3 million years. In the next phase multiple data values are plotted.

4.1.3 Phase Three: Multiple Temperature Data onto a Single Graph

To compare visually deep-ocean temperature relative to present (degC) and atmospheric surface air temperature relative to present (degC), a multi-line graph had to be coded. The web had nice examples of this, but as with anything they were not perfect fits with the data at hand. [52.] A time taking task was getting measurements from the object and creating the right domains for the x and y. Finally the lines were plotted in a single group element with color being the defining factor in splitting the two data variables (see figure 15).

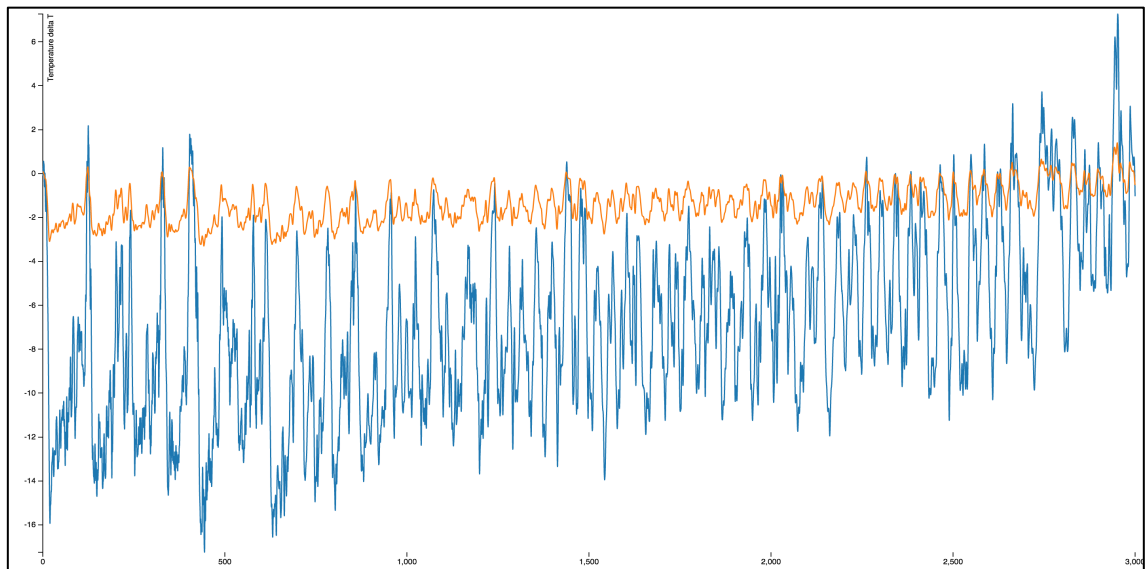


Figure 15: Output of the third phase of getting 60,000 points to show up on an SVG element. Screen capture of the second phase with multiple data [47]

As with all project, there were difficulties. The dataset chosen was quite large with 60,000 data points to create a curve from. As expected the loading of data created a substantial lag when loading the page with so much for the browser to draw. This became a huge problem throughout the next phases of development.

4.1.4 Phase Four: Creating Interactions and Mouse Events

Creating interactions and mouse events was the most time consuming phase. How to create interaction and change an SVG graphic according to clicks and mouse events? An initial starting screen had to be built with the logic that when a button was clicked it would load a graph with the data from the scientific dataset would appear. This was only the beginning as phase four ended up becoming the most time consuming of all the phases with bugs popping up after each new line of code. Time engrossing was creating loops of rendering animations within the D3 logic. Because of the rendering slowness of the large dataset, multiple smaller JSON files were made which were sections of the whole. This allowed for the illusion that the whole dataset was all the time being scrolled but actually a new dataset was loaded after an external popover.json file gave a Boolean variable to do so. A graph is animated once it is called by a runner function and popovers displayed with data transferred from an existing JSON file. This can be seen in more detail when reading the code clips in listing 1 and 2. A carousel effect was at first created with logic for a back button, but this was abandoned once noticed that the logic became too complex to execute in the required time for this thesis.

```

$( ".start" ).click(function() {
begin();
});

function begin() {
$( ".start" ).off("click");
$(".relative").hide();
var figure = $("#svgGraph");
figure.fadeIn( 1000 );
if(first) {
    firstRun("../json/1.json");
    first = false;
} else {
    runner("../json/1.json");
}
function firstRun(jsonFile) {
d3.json(jsonFile, function (error, data) {
    if (error != null) return console.warn(error);
    fillColorDomain(data);
    temperatures = generateTempColors(data);
    setAxisDomains(data);
...
function rotate() {
$("#svgGraph").hide();
$(".relative").show();
$( ".btn" ).click(function() {
    $("#svgGraph").show();
    begin();
});
...
function fillColorDomain(data) {
function runner(jsonFile) {
//load external data:
d3.json(jsonFile, function (error, json) {
    if (error != null) return console.warn(error);
    visualizeit(json);
});
};

```

Listing 1: Code clipping representing the runner function that switches between json files. For the full code check out github.com/laurames/d3-and-data [47].

With the help of jQuery the popover and runner functions could be constructed. The idea was to have all elements of the page already within the initial DOM structure and be called by JavaScript when needed. Unneeded elements would be hidden while the ones that were displayed were shown with CSS. HTML elements were given animations for a feel of transitions between stages of the visualization. Information of the presentation was in an external JSON file that could be read when needed.


```

$.getJSON( "../json/popovers.json", function( data ) {
  popoverJson = data;
});
var popover = fillPopover(1);
popover.data("page", 1);
popover.show();
}
function fillPopover(cur) {
  var popover = $("#popover-static");
  var popoverButton = popover.find('.btn-popover');
  popoverButton.unbind("click");
  var current = cur.toString();
  var json = popoverJson[current];
  popover.find("p").text(json.content);
  popover.find(".popover-title").text(json.title);
  popover.find(".popover").removeClass().addClass("popover " +
  json.placement);
  popover.css({"left": (json.x * width / 100), "top": (json.y * height /
  100)});
  if('last' in json) {
    popoverButton.on("click", function() {
      var nextPage = popover.data("page") + 1;
      loadNext(nextPage);
      popover.data("page", nextPage);
    });
  }
  if(!("end" in json)) {
    popoverButton.on("click", function() {
      popoverNextHandler(cur);
    });
  } else {
    popoverButton.text("Start over");
    popoverButton.unbind("click");
    popoverButton.on("click", function() {
      rotate();
      popoverButton.text("Tell Me More");
      popover.hide();
    })
  }
  popover.data("current", cur);
  return popover;
}
function popoverNextHandler(current) {
  var current = $("#popover-static").data("current");
  fillPopover(current + 1);
}
function loadNext(page) {
  runner("../json/" + page + ".json");
}
}

```

Listing 2: The code above gives a clearer understanding of how the popover function actually has the logic of how views are being carouselled. The logic was for elements to appear and disappear based on the popovers.json file that gets loaded and the Boolean value checked for what comes next. [47]

The major problem that arose in this phase was animating the large dataset without causing tearing. The execution of the animation gets called and the wrong interpolation of points might be the cause for this tearing phenomenon. For a smoother transition a custom interpolator function could be built in the future to test the scenario. Unfortunately the time constraint has not allowed this so far. Another hypothesis might be that because of the amount of points within the dataset the SVG drawing is slower than the

animation transition permits. The problem might also be highly linked to large taxing of processing power and this problem is common when working with *big data*. As with the first appearance of the term in 1997 by scientists at NASA, big data was described in the context of the problems they faced with it. For the scientists the amount of data was so large that they had the problem of visualizing it and ran into problems with the limitations of physical memory and great taxing of the capacities of the computer. In this thesis project the tearing might be linked to a similar fault in the optimization and largeness of the dataset taxing the browsers rendering capacity. In the browser application tearing might happen as a result of the large amount of taxing on the browser's processing power to draw all the required SVG elements according to the given data points. [58.] The problem persists within the current implementation of the visualization during the writing of this thesis, but investigation is being done to fix the transition effect when moving from one JSON file to the next.

4.1.5 Phase Five: Styling and Appearance

CSS styling of the application was done using Twitter Bootstrap. It is a "sleek, intuitive, and powerful front-end library for faster and easier web development." [53.] Currently it is the most popular HTML, CSS, and JS library for developing responsive, mobile first projects on the web. This can be seen as in March 2015, it was the most-starred project on GitHub, with over 78,000 stars and more than 30,000 forks. Furthermore its compatible with the latest versions of the Google Chrome, Firefox, Internet Explorer, Opera, and Safari browsers. It allowed for quick creation of buttons, popovers, and following a grid system by means of following good design principles.

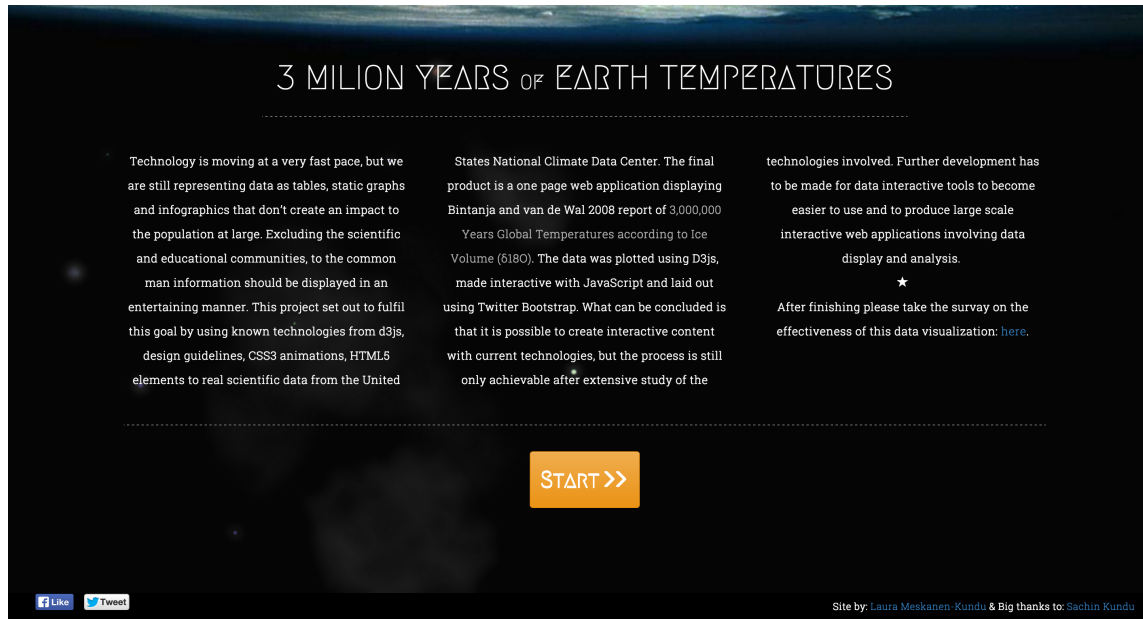


Figure 16: The page layout of the first initial starting screen of the one page application to visualize 3 million years of earth's temperature data.

The development of the initial page of the application had to be simple enough to attract users to click on starting the visualization (see figure 16). It also had to be informative about what the application is. All essential information was placed in the three split text column design and links were provided for the dataset and questionnaire. Sharing buttons to Facebook and Twitter were arranged in the footer to give the option for anyone to share the data and possibly provide more responses to the questionnaire. As of now a significant number of responses have not been given to the questionnaire to draw ample conclusions for this thesis, but from the one's received hypotheses have been drawn in 4.2. The questionnaire statistics have been made publically available for anyone to view once the questionnaire is answered and submitted. It will be interesting to see if the visualization in the long run has a positive impact or goes unnoticed.

4.1.6 Phase Six: Final Touches and Uploading to Remote Server

Final touches involved tweaking the CSS styling of the layout and the visualization's information popovers. The final code was pushed to gitHub and the project uploaded onto a server for future viewing by anyone. The JSON files were checked to make sure they displayed the desired data and they were put through an online validator. Small changes were made to main.css, runner.js and popovers.json to correct bugs and ty-

pos. Lastly, the application was tested by running it from a remote server and made live on the World Wide Web. [56.]

4.2 Results and Discussion

The produced application was a challenge to create from the perspective of implementation, but the results produced a fun and engaging data visualization. The one-page application (shown in figure 17) fills the void of today's need for interactive data visualizations and the problem of displaying large datasets to the common public. It can be said that a possible solution to displaying large datasets is to split data into relevant smaller chunks and stitching them together with data injection when called. This prevention of data overload was solved by not showing all values in one go, but by annotating the different portions of the dataset. As a positive consequence this makes the data intuitive for assimilation for the user. The interactive method used to display the otherwise inaccessible dataset also opens the application to a wider audience.

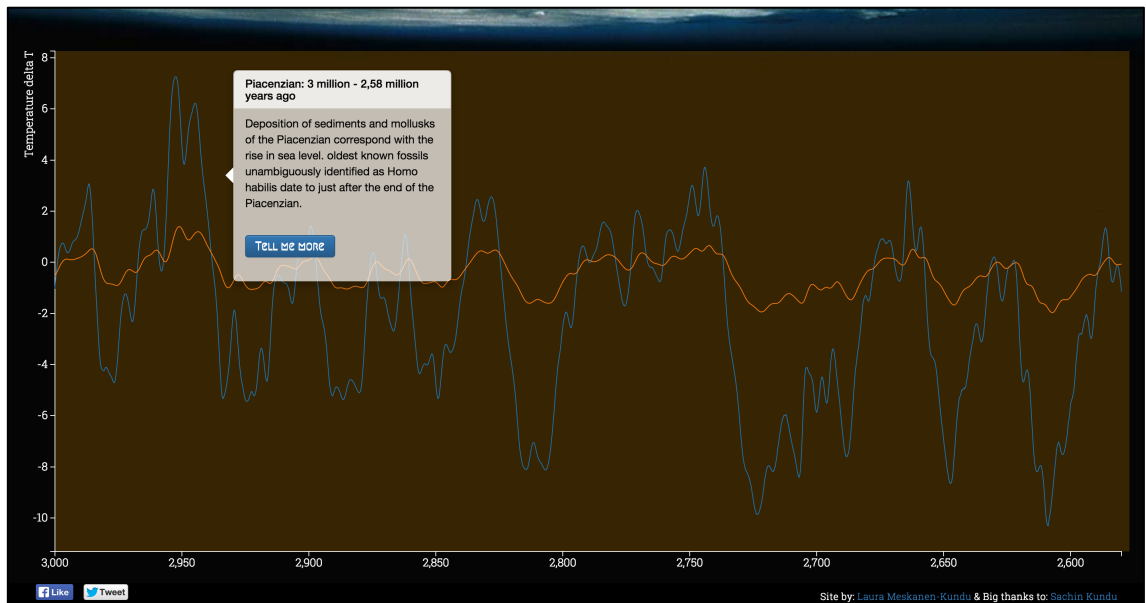


Figure 17: Capture of the application showing interactive information in a popover from the Piacenzian age: 3 million – 2.58 million years ago.

In a study done by Cynthia J. Miller, Jacquee McNear & Michael J. Metz, on a comparison of traditional and engaging lecture methods, it was found that “engaging lectures led to a statistically significant higher average on unit exams compared with traditional didactic lectures (8.6% higher, $P < 0.05$)”. More importantly it was discovered that en-

gaging improved long-term retention of learned information. [54.] Thus more engaging and interactive content should be the future of displaying information and providing data to the general public. Unfortunately further studies have to be made in the field of visualization for big datasets to move from using conventional tools, such as, tables or Excel. It was found that even with today's tools interactivity can be programmed, but the required learning curve is significant enough to put even professionals off the task. Tools such as D3 are the first building blocks to creating an ecosystem where anyone can start building interactive special effects visualizations, but more development is still needed for higher-level tools.

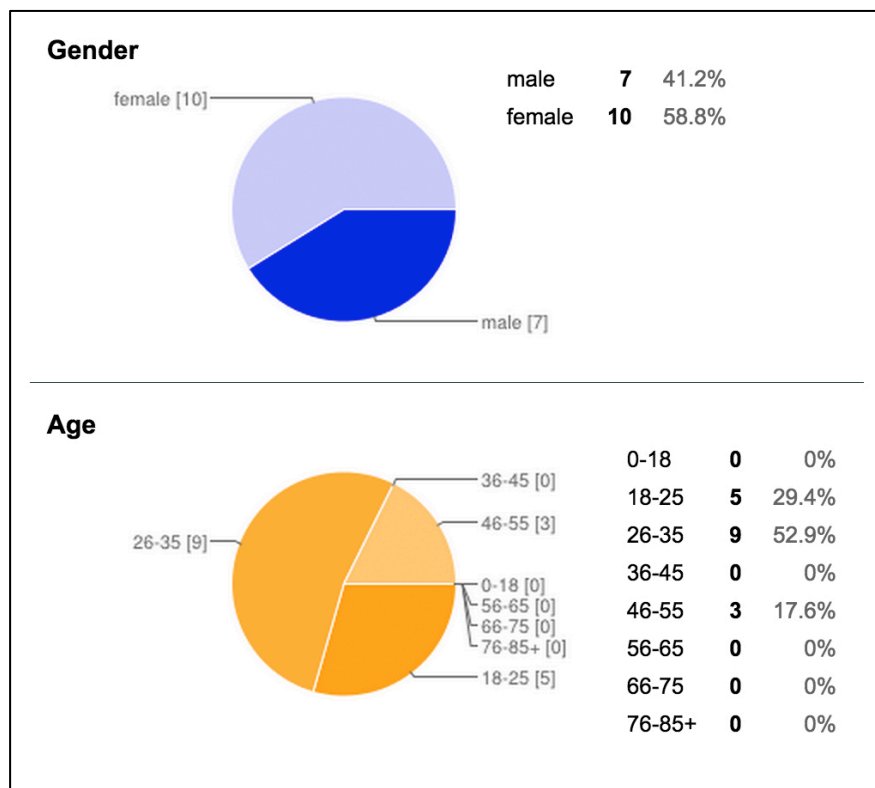


Figure 18: Gender and age distributions of the respondents to the survey after viewing the visualization. [57]

As stated in the introduction, a small survey of 17 responses was collected that consisted of people who had heard about the visualization through a Facebook post or word of mouth. The 17 responses were from the initial first three days the visualization was online. The respondents fit into three age categories between 18-25, 26-35 and 46-55. There was almost an equal amount of female and male responses in the sample group. It is due to note that the survey is open to the public and to new responses even after the submitting of this thesis. The results are also visible to anyone who takes the

questionnaire. Even though the take was not significant enough to make a conclusions of the population at large it can give a direction for a hypothesis. [57.]

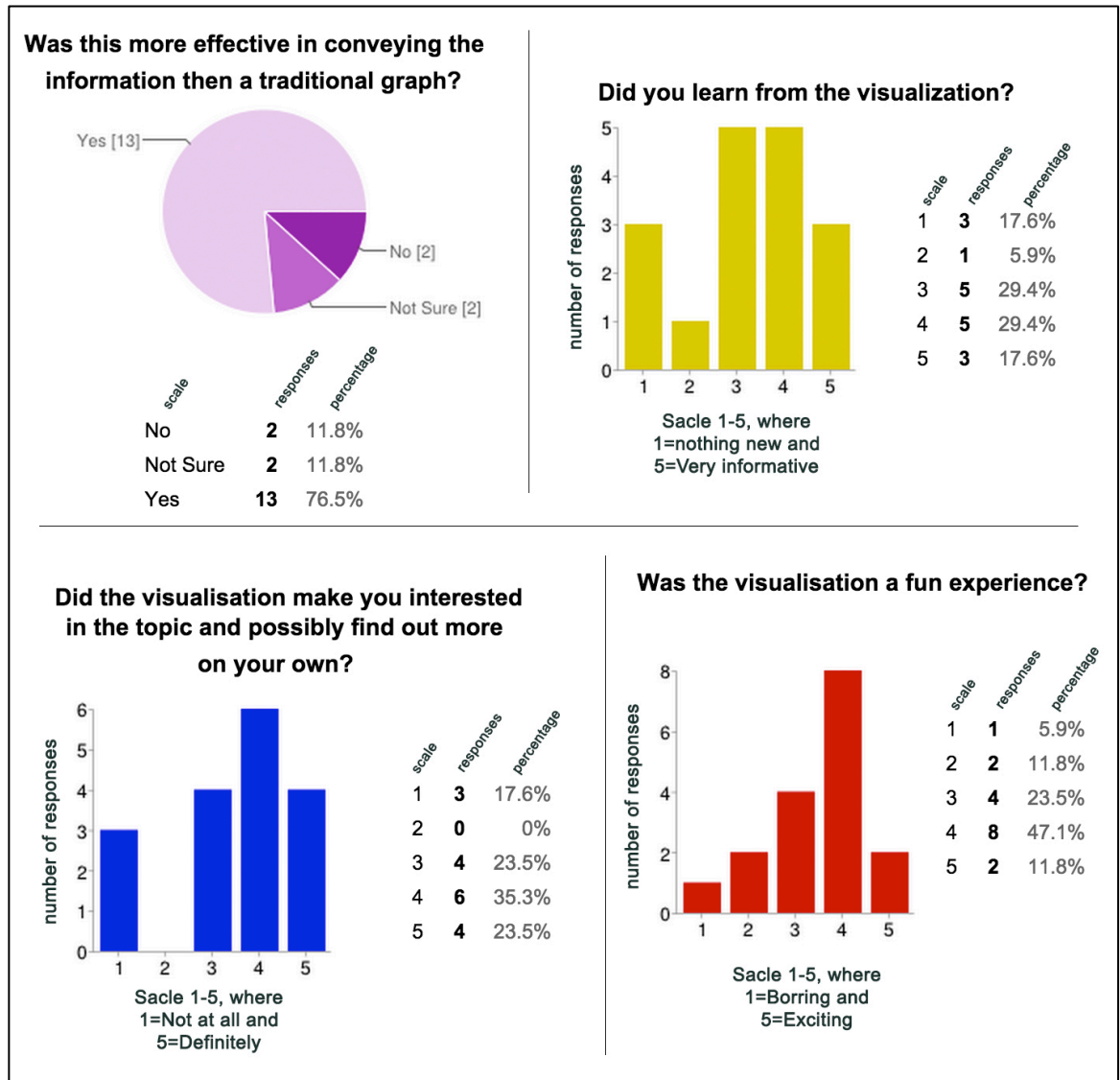


Figure 19: Graphs showing the results of the survey on the effectiveness of the visualization. [57]

The small sample group who answered the survey suggests that engaged learning compared to traditional didactic learning is superior in getting information across. This correlates with study on traditional and engaging lecture methods mentioned previously. [54.] As seen in the graphs above 76.5% said that the visualization was more effective in conveying the information than a traditional graph. Furthermore 47% said they learned from the visualization (yellow graph scale 4 and 5). Also 58.8% said that the visualization made them more interested in the topic and that they might research it on

their own time (blue graph scales 4 and 5). It is also noteworthy that 58.9% said that the visualization was a fun experience (red graph scales 4 and 5). The sample size being so small a hypothesis can only be formulated suggesting that interactive visualizations really are more effective in conveying information to the public at large when compared to normal static graphs. [57.]

5 Conclusion

An interesting effect to note from this study is that general population assumes that global temperatures have been rising steadily during the geological time frame of 3 million years to present. While human influence has affected global temperatures slightly the earth is still below average temperatures compared to the overall average within the time frame displayed in the data. Results like these come to light when data can be seen and perceived in an interactive context. The large scientific dataset of Bintanja and van de Wal's (2008) scientific report called 3,000,000 Years Global Temperatures according to Ice Volume ($\delta^{18}O$) [8] is hard to grasp when looked at in a table, but comes to light when split into respective earth ages, plotted and visualized. The future of data visualization has to move from its static roots to interactive content. The project conducted for this paper indicated that while interactive content can be produced with current technologies, advances have to be made for such visualizations to become accessible to masses. Also the small sample that answered the questionnaire in time for the writing of this thesis suggests that on average people were more interested in the information displayed interactively than when conveyed conventionally in static form. To conclude, data is knowledge, knowledge is power and sharing of power leads to peace. Even though the mission of this project was to study possibilities of creating interactive content with large datasets it has to be acknowledged that data is still not reaching the average consumer in an understandable form. Presenting data in a consumable format can have the next profound impact on the world.

References

- 1 Library (computing) [online]. Wikipedia; March 2015.
URL: http://en.wikipedia.org/wiki/Library_%28computing%29. Accessed 24 March 2015.
- 2 Ritchie S. King & Addison-Wesley. Visual Storytelling with D3: An Introduction to Data Visualization in JavaScript. Pearson Education; 2014.
- 3 Margaret Rouse. DEFINITION front-end [online]. WhatIs.com; May 2006.
URL: <http://whatIs.techtarget.com/definition/front-end>. Accessed 8 July 2014.
- 4 Mike Bostock. D3js.org [online]. URL: <http://d3js.org/>. Accessed 6 December 2014.
- 5 The 37 Best Tools for Data Visualization [online], CB Creative Blog; Nov 11, 2014. URL: <http://www.creativebloq.com/design-tools/data-visualization-712402>. Accessed 2 March 2015.
- 6 American Museum of Natural History [online]. Accessing the CREO extinctions database. URL: <http://creo.amnh.org/pdi.html#access>. Accessed 2 March 2015.
- 7 National Climate Data Center [online]. National Oceanic and Atmospheric Administration, Climate Reconstruction. URL: <http://www.ncdc.noaa.gov/data-access/paleoclimatology-data/datasets/climate-reconstruction>. Accessed 2 March 2015.
- 8 Bintanja, R. and R.S.W. van de Wal. North American ice-sheet dynamics and the onset of 100,000 year glacial cycles [serial online]. Nature 2008;454:869-872.
URL:ftp://ftp.ncdc.noaa.gov/pub/data/paleo/contributions_by_author/bintanja2008/bintanja2008.txt. Accessed 5 November 2014.
- 9 NASA. GLOBAL Land-Ocean Temperature Index in 0.01 degrees Celsius [online].
URL: http://data.giss.nasa.gov/gistemp/taledata_v3/GLB.Ts+dSST.txt. Accessed 2 November 2015.
- 10 Web development [online]. Wikipedia; April 2 2015].
URL: http://en.wikipedia.org/wiki/Web_development. Accessed 22 March 2015.
- 11 World Wide Web Foundation. History of the Web [online].
URL: <http://webfoundation.org/about/vision/history-of-the-web/>. Accessed 22 March 2015.
- 12 HTML [online]. Dictionary.com.
URL: <http://dictionary.reference.com/browse/html>. Accessed 22 March 2015.
- 13 HTTP [online]. Dictionary.com.
URL: <http://dictionary.reference.com/browse/HTTP?s=t>. Accessed 22 March 2015.

- 14 Reilly Lucas. 17 Ancient Abandoned Websites That Still Work [online]. URL: <http://mentalfloss.com/article/53792/17-ancient-abandoned-websites-still-work>. Accessed 12 March 2015.
- 15 The World Wide Web Consortium (W3C) [online]. URL: <http://www.w3.org/>. Accessed 12 March 2015.
- 16 Alexis Deveria and Lennart Schoors [online]. Browser support; February 2015. URL: <http://caniuse.com/>. Accessed 12 March 2015.
- 17 CSS [online]. TechTerms. URL: <http://techterms.com/definition/css>. Accessed 5 March 2015.
- 18 JavaScript [online]. TechTerms. URL: <http://techterms.com/definition/javascript>. Accessed 5 March 2015.
- 19 Front-end Code Standards & Best Practices [online]. Isobar; 2014 URL: <http://isobar-idev.github.io/code-standards/>. Accessed 9 April 2014.
- 20 Martin Smith & Robert Ward. JavaScript as a First Programming Language for Multimedia Students. ITiCSE '98 Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education. September 1998; 249-253.
- 21 Haverbeke Marijn. Eloquent JavaScript: A Modern Introduction to Programming. Second edition. No Starch Press; 2014.
- 22 ECMAScript [online]. wikipedia.com; March 2015. URL: http://en.wikipedia.org/wiki/ECMAScript#cite_note-Zakas-6. Accessed 11 March 2015.
- 23 Nicholas C. Zakas. JavaScript for Web Developers. Third edition. John Wiley & Sons; 2012. URL: <http://read.uberflip.com/i/113144-javascript-for-web-developers/44>. Accessed 13 December 2014.
- 24 Document Object Model (DOM) [online]. January 2005. URL: <http://www.w3.org/DOM/>. Accessed 21 March 2015.
- 25 Browser Object Model [online]. July 2012. URL: http://en.wikipedia.org/wiki/Browser_Object_Model. Accessed 19 January 2015.
- 26 Sanderson Steven. Rich JavaScript Applications – the Seven Frameworks (Throne of JS, 2012) [online]. Steven Sanderson's blog; 1 August 2012. URL: <http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/>. Accessed 19 April 2015.
- 27 JavaScript Libraries [online]. W3schools. URL: http://www.w3schools.com/js/js_libraries.asp. Accessed 21 March 2015.

- 28 Jeremy Osborn, Jennifer Smith & the AGI Creative Team. Web Design with HTML and CSS: Digital Classroom. Wiley Publishing, Inc; 2011.
- 29 Timothy Samara. Making and Breaking the Grid: A graphic Design Layout Workshop. Rockport publishers, Inc; 2005
- 30 Alex W. White. The Elements of Graphic Design: Space, Unity, Page Architecture, and Type, Second Edition. Allworth Press; 2011
- 31 Alex Bigelow, Steven Drucker, Danyel Fisher & Miriah Meyer. Reflections on How Designers Design with Data. AVI '14 Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces. May 2014; 17-24
- 32 Edward R. Tufte. The Visual Display of Quantitative Information: Second Edition, fifth printing. Graphics Press LLC; 2001
- 33 Lauri Nummenmaa. Käyttäytymistieteiden tilastolliset menetelmät. Julkaisukaupunki: Tammi; 2009
- 34 What is the difference between a JavaScript framework and a library? [online]. StackOverflow; 18 February 2015, URL: <http://stackoverflow.com/questions/11576018/what-is-the-difference-between-a-javascript-framework-and-a-library>. Accessed 19 April 2015.
- 35 Library vs. Framework? [online]. Programcreek.com. URL: <http://www.programcreek.com/2011/09/what-is-the-difference-between-a-java-library-and-a-framework/>. Accessed 19 April 2015.
- 36 Same-origin policy [online]. March 2015. URL: http://en.wikipedia.org/wiki/Same-origin_policy. Accessed 10 March 2015.
- 37 How to run things locally [online]. Theo Armour; Sep 2011. URL: https://github.com/mrdoob/three.js/wiki/How-to-run-things-locally/_history. Accessed 07 February 2015.
- 38 JavaScript Best Practices [online]. w3schools JavaScript tutorials. URL: http://www.w3schools.com/js/js_best_practices.asp. Accessed 29 January 2015.
- 39 Jeffrey Way. 24 JavaScript Best Practices for Beginners [online]; Jun 2009. URL: <http://code.tutsplus.com/tutorials/24-javascript-best-practices-for-beginners-net-5399>. Accessed 30 January 2015.
- 40 Katie Peek. World population mapped as peaks and valleys: one of our 15 favorite recent data visualizations [online]. Dec 2014, URL: <http://www.popsci.com/world-population-mapped-peaks-and-valleys>. Accessed 23 February 2015.
- 41 Data cleansing. Wikipedia; March 2015. URL: http://en.wikipedia.org/wiki/Data_cleansing. Accessed 24 March 2015.

- 42 D3. Js API Reference.
URL: <https://github.com/mbostock/d3/wiki/API-Reference>. Accessed 24 March 2015.
- 43 Mike Dewar. Getting started with D3. Published by O'Reilly Media, Inc.; 2012
- 44 Scott Becker. Learning D3 Part 3: Animation & Interaction [online]. July 2012.
URL: <http://synthesis.sbecker.net/articles/2012/07/10/learning-d3-part-3-animation-interaction>. Accessed 22 March 2015.
- 45 Server-Side JavaScript Guide [online]. Netscape Communications Corporation; 1998.
URL: <http://docs.oracle.com/cd/E19957-01/816-6411-10/contents.htm>. Accessed 21 March 2015.
- 46 Web search interest: infographics – Worldwide 2004, present [online]. Google Trends.
URL: <https://www.google.com/trends/explore#q=infographics>. Accessed 16 March 2015.
- 47 Laura Meskanen-Kundu. d3-and-data [online]. March 2015.
URL: [githubhttps://github.com/laurames/d3-and-data](https://github.com/laurames/d3-and-data). Accessed 15 April 2015.
- 48 Dr. Hannes Grobe [online]. Alfred Wegener Institute.
URL: <http://www.awi.de/People/show?hgrobe>. Accessed 1 April 2015.
- 49 Marine isotope stage [online]. Wikipedia; April 2015.
URL: http://en.wikipedia.org/wiki/Marine_isotope_stage. Accessed 6 April 2015.
- 50 Carol Kendall. Resources on Isotopes. Fundamentals of Stable Isotope Geochemistry. URL: <http://en.wikipedia.org/wiki/%CE%9418O>. Accessed 6 April 2015.
- 51 Line Chart [online]. D3 examples. URL: <http://bl.ocks.org/mbostock/3883245>. Accessed: 6 April 2015.
- 52 Multi-Series Line Chart [online]. D3 examples. URL: <http://bl.ocks.org/mbostock/3884955>. Accessed 5 January 2015.
- 53 Twitter Bootstrap [online documentation].
URL: <http://getbootstrap.com/>. Accessed 3 December 2014.
- 54 Cynthia J. Miller, Jacquee McNear, Michael J. Metz. A comparison of traditional and engaging lecture methods in a large, professional-level course. *Advances in Physiology Education* [serial online] 2013; 37(4):347-355.
URL: <http://advan.physiology.org/content/37/4/347>. Accessed 3 April 2015.
- 55 Shan Carter. Mr. Data Converter [online]. URL: <https://github.com/shancarter/Mr-Data-Converter>. Accessed 4 February 2015.
- 56 Laura Meskanen-Kundu. 3 Million Years Of Earth Temperatures. April 2015.
URL: <http://3milyears.lauramk.me/>. Accessed 19 April 2015.

- 57 Laura Meskanen-Kundu. How was "3 million years of Earth Temperatures"? [online]. March 2015.
URL: https://docs.google.com/forms/d/1IO_pfpqdK3F--9ZTX762PH-QWgYs1B2YfXkWsuXHbqs/viewanalytics. Accessed 17 April 2015.
- 58 Gil Press. 12 Big Data Definitions: What's Yours? [online]. March 2014.
URL: <http://www.forbes.com/sites/gilpress/2014/09/03/12-big-data-definitions-whats-yours/>. Accessed 17 April 2015.
- 59 Comparison of JavaScript charting frameworks [online chart]. Wikipedia; April 2015.
URL: http://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_frameworks. Accessed 19 April 2015.
- 60 Drew Skau. Why D3.js is So Great for Data Visualization [online]. January 2013.
URL: <http://blog.visual.ly/why-d3-js-is-so-great-for-data-visualization/>. Accessed 19 April 2015.