Data Visualisation

Exploring the possibilities of Data Visualization in e-inks displays and the EPUB format.

Ignacio Talavera Cepeda
Supervisor: Hans-Jörg Schulz

Aarhus Universitet

May 26, 2022

Abstract

The Electronic PUBlication (EPUB) format is based on the same three frameworks than the Web: HTML, CSS and JavaScript. However, very few scripting capabilities are usually included in these publications, and the Data Visualisation present in academic papers is limited to static images. We present a methodology for embedding visualizations using the D3 library, alongside a comprehensive explanation of how the EPUB format works, a study on compatibility, which other libraries are available, and a gallery of examples with their respective code.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Academic papers and publications are the main forms of knowledge sharing in the scholastic environment; but, even though visualizations are commonly embedded as figures to support the ideas and the research of the authors, the way users have to interact with those files, which we usually read via a browser of an e-reader, is very limited. Those files were designed to serve as a representation of a printed paper, therefore restraining the actual interaction and visualization capability that modern computer offers. The main usage of these type of files is not to be printed anymore.

This project aims to research the visualization possibilities that the EPUB format allows and how to adapt the visualizations to different technologies and systems used for reading academic publications. The specifications and boundaries of the EPUB format on different devices are explored, and a methodology for embedding visualizations in EPUB files is detailed, alongside examples of different types of visualizations.

1.1 Context

Electronic publications provide an opportunity to overcome the limitations of the physical format, and this is especially relevant in the academic environment, where dynamic information and data visualization is a key characteristic of the domain [1]. Very often, academic papers offer the reader different types of data visualizations that help the authors support their thesis or show their results. These visualizations are designed on a computer and embedded as an image (with a format either Scalable Vector Graphics or raster graphics) into the publication. This process, while necessary to adapt the visualization to the classical conception of publication, prunes some of the key features of electronic data visualizations, such as dynamic data input and interaction. And, even though it is a necessary process if the user wants to print the document, users usually consume papers and publications on computers, tablets or other electronic devices, where those capabilities could still be present [1].

There is a variety of devices with which readers can interact with academic publications. Browsers exploit the capabilities of modern computers and usually support multiple programming languages and engines. On the other hand, there are several e-reading applications in tablets and phones, which are also exploiting those capabilities, but with different screen sizes and form factors. And there are also e-ink devices, such as e-books, which are widely used but are limited in terms of hardware, visualization and interaction possibilities. However, they also provide benefits, like limited energy consumption and less eye strain when used for prolonged periods. They are relatively cheap devices, and quite popular nowadays [2][3].

As the way of consuming academic papers changes, it starts making sense to push the capabilities of the format in which those publications are read forward while maintaining the possibility of printing it into paper. The EPUB format allows the inclusion of JavaScript scripting, but there has not been any major concern in the industry to define guidelines and processes for exploiting this capability. And, even though the format allows this execution, each e-reader device has its own engine and its own set of JS capabilities that can be executed, and those that cannot.

Data visualizations are usually embedded into EPUB publications as images that have to be exported first. There are usually SVG files, which results in better quality and the possibility of resizing, but they can also be found in any of the raster graphics formats. Therefore, all the different aspects of the visualizations that could be enhanced via scripting are pruned, no matter if the publication is printed, converted into a PDF file or read through an EPUB reader.

1.2 Problem analysis & motivation

Even though charts and data visualization can be embedded as SVG or raster graphics into the EPUB format, this procedure denies the opportunity of having dynamic data, interactivity and other perks provided by JS scripting. This, combined with the lack of documentation about scripting in EPUB files, motivates this student project.

The problem faced consists of researching how to run JavaScript code into EPUB files, which options regarding scripting are allowed by the format and which are constrained by it or by the devices. The capabilities of several reading systems will be tested in this process. Following the progress in JavaScript scripting, the focus of it will be driven towards the creation of data visualizations, with special attention to documenting and offering guidelines to reproduce these visualizations or embed new ones. Therefore, this project aims to offer not only instructions on

how to implement data visualizations into EPUB files, but serve as a start point to bootstrap the discussion on the matter and push the current capabilities forward.

1.3 Objective

The objectives of this student work are the following:

- Research on how to run JavaScript code into an EPUB file, taking into account the specifications of the format and the integration possibilities that the format and the programming language allow between each other.
- Test the JS capabilities in several EPUB reading systems, both specific devices and reading software present in computers, phones and tablets, and present information on the compatibility of the most used EPUB reading systems.
- Experiment on the possibilities of creating data visualizations using JS scripting, testing the most popular libraries.
- Defining guidelines and processes to include data visualizations in EPUB files in a structured manner, with a special focus on the D3 library.
- Including examples of different visualizations and how to reproduce them.
- Impulse the discussion about scripting for data visualization in an environment where academic publications are often consumed via devices that offer scripting capabilities.

1.4 Phases of development

This project has followed a linear approach. The line of work started with running the first scripted content in the EPUB format, and from there different aspects have been approached, with incremental difficulty. Following this main line of work, a documentation effort has been done throughout the entirety of the project, which culminates in this report and in the related video.

The first part of the project started with a research phase on the format and the possibilities of JavaScript code embedding. In this context, the first JS script was executed. Those first scripted EPUB documents served as testing material for the compatibility aspect of the project. Some of the most popular options in e-book reading were tested, and the obtained results were noted and transcribed into a compatibility matrix.

Afterwards, and exploiting the capabilities of the systems that offered the most in terms of compatibility and performance, data visualizations were introduced using the D3 library. In this part of the process several techniques were iterated, as there was a jump in complexity from executing simple scripts to running a JS library, introducing data, and serialising the creation of different types of visualizations, parsing, etc.

When the creation of data visualizations was achieved, the attention shifted to two different efforts: creating a gallery of examples and testing different libraries than D3.

The final stage of this project has consisted of gathering the different conclusions and achievements obtained and finishing the documentation phase.

1.5 Resources used

Due to the nature of the EPUB format, the programming language for this project has been JavaScript, with the support of HTML and CSS for creating the documents to be converted to EPUB. For creating the EPUB files, two tools have been used: Calibre[1] and eCanCrusher[2]. Both are tools that transform zip files into EPUB. Calibre handles the creation of all the internal files that an EPUB file needs to comply with the standard, but it is also not developed with scripting in mind. eCanCrusher relies on the user to manage the internal structure of the EPUB and therefore was the final tool of choice for this project. D3 was the most used data visualization library in the project, but experimentation was also carried out with Apache ECharts[3], Plotly[4] and Vega[5].

[1] `https://calibre-ebook.com`
[2] `https://www.docdataflow.com/ecancrusher/`
[3] `https://echarts.apache.org/en/index.html`
[4] `https://plotly.com/javascript/`
[5] `https://vega.github.io/vega/`

The documentation process was carried out using a combination of Notion[6] and Overleaf[7], and the code development was done in a GitHub repository[8], following an iterative process of issuing new functionalities, pushing changes and self-reviewing them. That repository contains the most relevant files that were part of the research and the development, alongside the example gallery.

[6] https://www.notion.so
[7] https://www.overleaf.com/
[8] https://github.com/ignacioct/Visualization-EPUB

2 State of the Art

2.1 E-ink displays

The electronic paper displays, also known as electronic ink displays, are a type of display that tries to mimic the look of the ink over the regular paper [4]. Instead of emitting light, like the ordinary displays, the e-ink displays reflect the ambient light, which offers a more comfortable reading experience and provides a wider viewing angle. The current technology allows the displays to show static elements without an electric current, and thus the devices with these displays usually have a good autonomy, as they only need to use energy for changing the current page. The first coloured e-ink display was released by E Ink Corporation, co-founded in 1997 by MIT students and professors [5].

There are different applications for this type of display, being one of the most popular e-book readers. Since the first reader in 2004 [6], many different companies have drawn their attention to developing e-readers and pushing the technology forward. Amazon started producing the Amazon Kindle in 2007, while other options are the Barnes & Noble Nook [7], the PocketBook Inkpad [8] or the Kobo eReader [9]. Usually, the operative systems of the e-books are based on Linux. There are other products that have been released with e-ink displays, such as wristwatches [9], status displays [10], digital signage on cities [11, 12] and others[13].

This work focuses on the e-ink displays applied to e-book readers, as they are devices that are often used to consume academic publications and books. However, without taking into account factors such as the operative system, the form factor and the processing capability, all e-ink displays have similar capabilities, and therefore the further conclusions drawn in this work are also relevant for other devices.

2.2 E-ink displays on e-book readers

While being popular for their low power consumption, low cost and reduced eye strain, e-ink displays present few colours and low refresh rates, and most of them are only black-and-white. The e-reader as a device balances these trade-offs and takes advantage of the strengths, focusing its functionality on consuming books and academic publications.

However, and also due to the trade-offs that this technology presents, not many ways of interacting with the written contents have been explored. Even though EPUB reading requires, in theory, an engine capable of running HTML5, CSS and JS, very few devices offer a way of integrating the possibilities that are available on the modern website into the reading process. This is mainly for maintaining a low processing power requirement and keeping the devices both cheap and with large autonomy.

Regarding Data Visualisation, researchers have explored the alternatives to the use of different colour hues and saturation values in e-inks. The main approach is to use textures, which are different shapes used as patterns used in both the stroke or the fill of a data-representing mark (see Figure 1 and 2) [3]. These textures have different attributes that can be modified to obtain any kind of texture, being the most common the density, the shape size, the orientation, the shape type and the kind of grouping.

[9] `https://www.kobo.com`

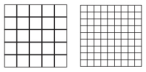
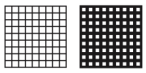
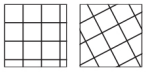
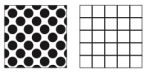
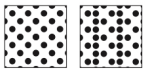
Texture property	Example Fill	Example Stroke	Length	Categorical Data	Ordered Data
Density			2–5	∼	✓
Shape Size			4–5	✗	✓
Orientation			2–4	✓	✗
Shape type			large	✓	✗
Grouping			small	✓	✗

Figure 1: Example of different texture properties applied to both strokes and fill [3].

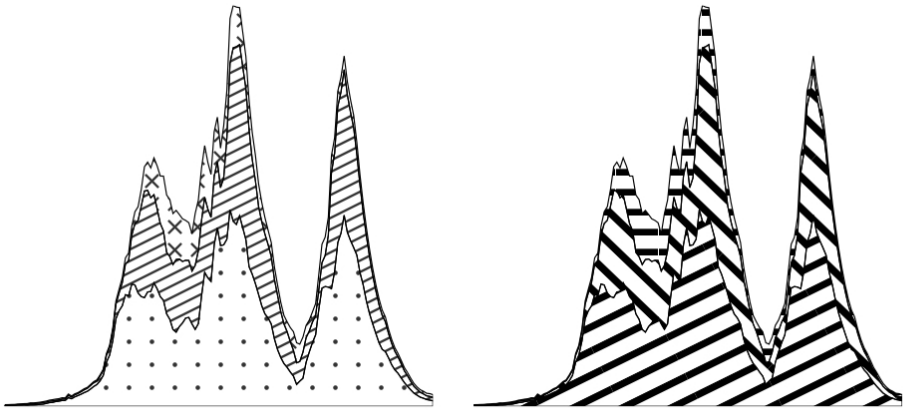


Figure 2: Experiment for replacing categorical color in an area chart.

Figure 2: Example of replacing categorical colour in an area chart [3].

To go even further in textures, Zhong proposes the iconic texture [14], a kind of texture that used icons as the shape attributes (see Figure 3). These icons have a semantically-resonant effect on the chart reading task.

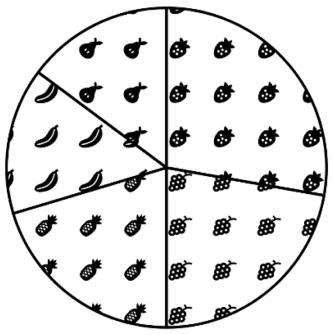


Figure 3: Example of iconic textures in a pie chart[14].

There are several tools to create textures and use them in visualizations. Texture.js[10] is a JavaScript library with extended functionality options, and Hero Pattern[11] is a website where users can design and download a texture from the browser. Both options could be used to create a data visualization for EPUB using textures.

2.3 Data Visualization in Javascript

Data Visualisation is an essential part of any data-oriented programming activity, even though the creation of useful, comprehensive visualizations was not well-established until recent times. One of the most important events in this area was the release of D3 in 2011, an open-source JS library built with and for HTML, CSS and JS itself with support for a variety of visualizations and tasks [15].

The popularity of JavaScript rises from its web-oriented design. Our communicating and sharing practices are entangled within the Internet, and that means that the future of Data Visualizations involves JavaScript, the most used scripting programming language on the web [16].

This field is currently one of the most demanded, especially thanks to the rise of Data Science. D3 emerged at the opportune time and used its potential and the market momentum to become the standard for Data Visualisation inside the browser and for all the technologies based on HTML, like EPUB.

2.4 Javascript on EPUB

An EPUB file may contain scripting capabilities, using the facilities defined in the HTML standard. When an EPUB file contains scripting, it is referred as a Scripted Content Document. [17]. There is a scripted property on the manifest of the EPUB file which indicates that a certain file is a Scripted Content Document.

While the EPUB Scripting Manifest [18] states that JavaScript code can be run within an EPUB file, each individual e-book reading device or software has its own limitations and capabilities, and they are usually vague and not well-defined. There is no process of scripting for EPUB files agnostic from the device used, and readers do not usually offer a console interface to check for coding errors, making this kind of development difficult and aproachless. There are, however, some examples available in the EPUB 3 Feature Matrix [19].

[10] `https://riccardoscalco.it/textures/`
[11] `https://heropatterns.com`

3 EPUB specifications & JavaScript integration

In this section of the report the characteristics of the EPUB format and how they relate with JavaScript scripting functionalities, in terms of compatibility, are explained. This includes information about the EPUB format and how it works, the specifications of the format regarding the scripting functionalities, and the initial snippets of code that were executed to test the main scripting functionalities that were available in different devices and e-book reading platforms. At the end of this chapter, the compatibility matrices summarise the obtained results.

3.1 EPUB format

EPUB, acronym for *Electronic PUBlication*, is an e-book file format that uses an extension with the same name. Since 2007, a technical standard published by IDPF regulates the format [20]. It is widely supported by many electronic readers, and compatible with a myriad of devices. It is often the choice for content packaging, in the context of books and other written publications. As opposed to PDF files, it runs on XML [21, 22].

The main features of the EPUB format are:

- Possibility to optimize the text layout for different display sizes, allowing a broad spectrum of form factors.
- Fixed-layout functionalities.
- Supports raster graphics and SVG images.
- Supports metadata and CSS styling.
- Bookmarking, highlighting and note-taking are directly supported by the standard.
- Supports MathML, for the inclusion of mathematical formulas.
- Digital rights management is supported directly by the format.
- JavaScript scripting can be executed.

The adoption of the EPUB format has been majoritarian. Except for Kindle, which uses its own extension, platforms like Google Play Book and Apple Books natively support the format. It is also compatible with browsers, like Microsoft Edge or Google Chrome, but it usually requires extensions to run. The requirements for running EPUB files are the same as the requirements for the modern web: HTML5, CSS and JS. The framework is very similar to web browsers.

Regarding the technicalities of the EPUB files, they can be easily explained as zip files with a fixed structure [23]. An EPUB publication is delivered as a single, unencrypted zipped archive containing a set of interrelated resources. They have some mandatory files to structure the content, and the content itself (see Figure 4).

- **mimetype**: this file states that the file is an EPUB publication. The content of this text file must be "application/epub+zip".
- **container**: this XML file, found in the mandatory META-INF directory, is used for bootstrapping, as it contains the location of the package document. Alongside the container, in the META-INF directory, there can be other optional files, like a signature file (containing the digital signatures of the container and the contents).
- **manifest**: list of publication resources, including HTML text chapters, images, multimedia, scripts, etc. The system will only process files that are in this list.
- **metadata**: file containing information about the content. The EPUB format requires a title, an identifier, a language and a modifier.
- **spine**: backbone of the document. Here, the reading system finds the default reading order of the chapters or sections in the publication (they are called spine items). Each spine item contains a reference to a manifest item, and they can be declared non-linearly.

- **Package Document**: also known as OPF file, it is an XML file that carries the bibliographic and structural metadata of the EPUB publication and therefore is the primary source of information on how to process and display the publication.

Both the content and the metadata files are combined into a single file. This file will be used along with this project, and it will be called content.opf

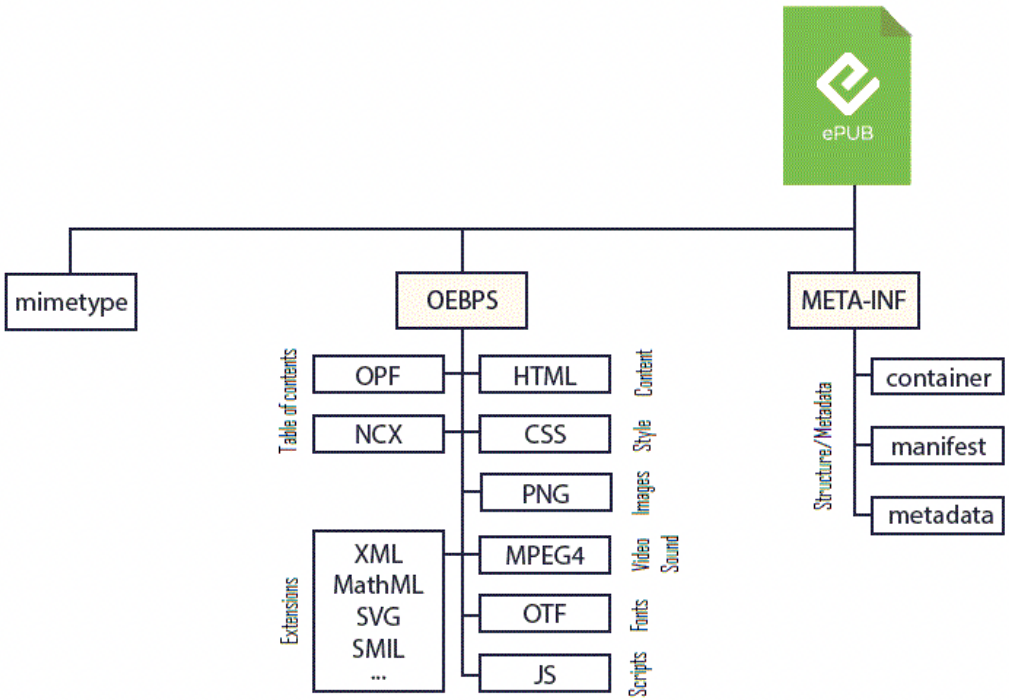


Figure 4: Anatomy of an EPUB file[23].

3.2 EPUB specification regarding JavaScript

EPUB files may contain scripting features using the facilities defined by the HTML, therefore allowing JavaScript code to be executed. To add a scripted element into an EPUB file, we have to declare it in the manifest of the file setting the scripted property as true. The scripted property of the manifest item element is used to indicate that an EPUB Content Document is a Scripted Content Document. In the code snippet below, you can see an example of a manifest file defining scripted content.

Listing 1: Example of the manifest section containing several scripted files.

```
<manifest>
 <item id="titlepage" href="titlepage.xhtml"
 media-type="application/xhtml+xml"/>
 <item id="html" href="gallery.xhtml" media-type="application/xhtml+xml"
    properties="scripted"/>
 <item id="ncx" href="toc.ncx" media-type="application/x-dtbncx+xml"/>
 <item id="page_css" href="page_styles.css" media-type="text/css"/>
 <item id="css" href="stylesheet.css" media-type="text/css"/>
 <item id="css" href="css/cole.css" media-type="text/css"/>
 <item id="echarts" href="script/echarts/echarts.js"
    media-type="text/javascript"/>
 <item id="bar" href="script/barchart.js" media-type="text/javascript"/>
</manifest>
```

As seen in the example, the document that has the scripted property is the one where the execution of the JavaScript code happens. In this example, even though the code lies in both *script/echarts/echarts.js* and *script/barchart.js*, the code is imported and executed from *gallery.xhtml*, and thus that is the file that has the scripted property set to true.

3.3 Reading systems

For this project, several reading platforms were taken into consideration. The main objective pursued was to test a variety of reading systems, including hardware devices, like e-book readers, and software systems, that allow EPUB reading on smartphones, tablets and personal computers.

- **Apple iBooks**[12]: platform available in Apple devices. It offers both a book store and the possibility to include personal EPUB files. Synchronisation between all Apple devices with the same account is made through Apple's cloud services, iCloud.
- **Calibre**[13]: software that allows to both create EPUB files from HTML and read any file. Specially useful, as the reader includes a console functionality, which can be used to debug the EPUB files created at some level.
- **Kobo Clara HD**[14]: e-reader developed by Rakuten, with a 6" glare-free high-definition 300 PPI e-ink display. Designed with the EPUB format in mind.
- **PocketBook**: it features a 7.8 inch e-ink display with a resolution of 1404 x 1872 and 300 PPI. The system has its own EPUB shop, and also allows to import personal EPUBfiles[8].
- **Adobe Digital Editions**[15]: Adobe's software for reading EPUB files.

3.4 Compatibility

In this subsection, the different elements taken into consideration for comparing the compatibility of the different reading systems with the EPUB format, focusing on the JavaScript scripting functionalities, are explained and discussed. In the end, the compatibility matrices show the results of the different experiments carried out with the different reading systems.

3.4.1 Navigation Property

The reading systems that support scripting are required to have the *navigator.epubReadingSystem* property, which allows querying the system about its scripting support and specific properties. There are several ways to check if this property is available, and it must be done in a way that the reading system supports. For example, if an alert is used in a system that does not support an alert system (as in the code below), the user will not receive an alert in the reading system stating that this property is available; not because it is not available, but because alerts are not supported.

Listing 2: Example of script that triggers an alert with information about the reading system.

```
<script type="text/javascript">alert("Name:" + navigator.epubReadingSystem.
name + "/version:" + navigator.epubReadingSystem.version +
"/layoutStyle" + navigator.epubReadingSystem.layoutStyle);</script>
```

3.4.2 Detecting Features

Through its *hasFeature()* method, *epubReadingSystem* allows content authors to query the characteristics of the scripting support in the system. Some of them are required to be report by systems that implement *epubReadingSystem* properties:

[12] https://www.apple.com/uk/apple-books/
[13] https://calibre-ebook.com
[14] https://us.kobobooks.com/products/kobo-clara-hd?utm_source=Kobo&utm_medium=TopNavTest&utm_campaign=Clara
[15] https://www.adobe.com/es/solutions/ebook/digital-editions.html

- **DOM-manipulation**: spine-level scripts may modify the attributes and CSS styles that affect the content layout.
- **Touch-events**: the device supports touch events.
- **Mouse-events**: the device supports mouse events.
- **Keyboard-events**: the device supports keyboard events.
- **Spine-Scripting**: scripting can alter any element in the whole document, and they are not constrained to the element they belong to.

3.5 Inclusion Models

Given the possibility of altering the spine of the document via scripting, and taking into account that some reading systems allow spine-scripting and others do not, two models for scripting inclusion can be devised:

3.5.1 Container-constrained

This inclusion model requires the script to be associated and embedded into a subcomponent, rather than it being embedded in the spine of the document. This component is usually an *iframe*. The scripted area has then an explicitly defined and bounding element, and thus the changes that it may generate can only affect that subcomponent.

Even though it seems like the best practice for most cases, wrapping each interactive component into an object can bear no resemblance to the content authority on the web, and iframes are uncommon in EPUB publications (sometimes they are even disallowed, like in the legacy EPUB 2 reading systems). Most reading systems have not been extensively tested with documents including *iframes*.

3.5.2 Spine-level

This is the inclusion model that is present in most of the script-aware EPUB 3 reading systems, and it is the dominant approach. Scripts can make changes in the Document Object Model (DOM) of the document. Inclusion of scripted elements becomes easier, and compatibility gets enhanced, but spine-level publications must try to minimize the number of changes to DOM elements, to improve readability, compatibility and processing times.

3.5.3 Compatibility of scripting features in different reading systems

To check the system properties of a given reading system, you can follow the procedures explained in the subsections about navigation properties and feature detection. In the compatibility matrix shown below, you can see the results of different JS scripting texting, finishing with the test of D3 visualisations (see Table 1 and 2).

Table 1: Compatibility matrix of different functionalities, first part.

Device/Platform	Countdown	Alert System	Button triggers a hello world	Button creates a text area
iBooks (Mac, iPad & iPhone)	Works	Works	Button is shown, no trigger	Button is triggering text
Calibre	Works	Does not work	Button is shown, no trigger	Button is triggering text
Kobo	Does not work	Does not work	Button is shown, no trigger	Button is turned into normal text
PocketBook	Works	Does not work	Button is shown, no trigger	Button is turned into normal text
Adobe Digital Editions	Does not work	Does not work	Button is shown, no trigger	Button cannot be pressed

Table 2: Compatibility matrix of different functionalities, second part.

Device/Platform	Checkbox	Select box	Graphics created with D3
iBooks (Mac, iPad & iPhone)	Checkbox cannot be selected	The select box can be used and it displays the contextual menu	Works
Calibre	Checkbox can be pressed and it triggers text	The select box can be used and it displays the contextual menu	Does not work
Kobo	Checkbox is not rendered	Select box is turned into a list	Does not work
PocketBook	Checkbox is not rendered	Select box is turned into plain text, only showing the first option.	Works
Adobe Digital Editions	Checkbox is not rendered	Select box is turned into a list	Does not work

3.5.4 Conclusions on compatibility

Different conclusions were drawn from these first experiments that were important for the subsequent development of this project:

- iBooks has wider compatibility, with its only drawback being that it is only supported on Apple devices.
- If something works in any Apple device, it will work in a very similar way in the others. All of the EPUB files are automatically synchronised between devices.
- Some code can be run into the Calibre reader, but it does not support D3 inclusion, which limits the capability to develop visualisations for that platform.
- PocketBook, while being more constrained in terms of JavaScript functionalities than iBooks, allows using the D3 library.
- Almost no results were obtained from Kobo nor from Adobe Digital Editions.

For the rest of the project, and given these results, the developed techniques will be tested in Apple iBooks and in the PocketBook reader.

4 Methodology for embedding visualizations using D3

In this chapter, the techniques to embed data visualisations in EPUB files using JavaScript and the D3 library are covered, including how to create the file itself, importing the library, working on the visualizations and feeding data into the visualization. The HTML elements that contain the visualizations are also explained.

4.1 Creation of the EPUB file

As stated in previous sections of this report, an EPUB file is a zip file with a specific structure that can be replicated. There is a template available on the GitHub page of this project [16] that can be used as a starting point to create your own file. The following explanation of this chapter is based on that file.

The file containing the HTML contents of the EPUB file is *content.xhtml*. In there, text, images and other multimedia content can be included as a normal HTML file, and it will be converted into an EPUB publication at the end of this process. Here is also the place for including scripted content. D3 is included in the head section, and other scripts can be included in both the head and the body section. It is recommended to use individual JS files for each visualization or functionality, and to include them in the body section, next to the HTML element that they are updating. It is also a good practice to keep all the JavaScript files in a *script* folder.

Afterwards, all those files have to be included into the manifest, which, in the case of this example, lies on the file *content.opf*. JavaScript files have to be included as follows:

Listing 3: Inclusion of a JavaScript file into the EPUB manifest.

```
<item id="d3" href="script/d3/d3.js" media-type="text/javascript"/>
```

Our XHTML file has to be included with the scripted property enabled:

Listing 4: Inclusion of a JavaScript file into the EPUB manifest.

```
<item id="html" href="content.xhtml"
media-type="application/xhtml+xml" properties="scripted"/>
```

Once all of the files are correctly declared in the manifest of the file, the spine section must lead to the first part of the EPUB file, for bootstrapping reasons. In the case of the example, there is a title page as the first element, but *content.xhtml* could directly be used as the starting point of the publication.

When the folder of the publication is finished, it must be converted to an EPUB file. Calibre can be used for this task, but it is not recommended, as it will probably break some of the dependencies that have been declared using the template, and the JavaScript functionality can break. Instead, the user can follow these processes, depending on the operative system of choice:

- Following a minimalist approach, the user can compress the folder as a zip file, and change its extension for EPUB. It is likely to work if the operating system does not include extra folders or files when compressing (like Mac). To avoid possible errors, a second approach can be used.
- Using eCanCrusher[17], which is available for Mac and Windows. By dragging and dropping the folder into the desktop icon, an EPUB file is returned. It can also be used to transform EPUB files back to compressed folders.

4.2 Inclusion of JavaScript in the EPUB file

In this subsection, a further look into a JavaScript file created for embedding scripted content, focusing on the data visualization use case, is given. The folder *minimal_ example*, that contains a basic bar chart visualization, will be followed, in which the main .xhtml file, containing the HTML content of the publication, is called *gallery.xhtml*.

[16] https://github.com/ignacioct/Visualization-EPUB/tree/main/epub_template
[17] https://www.docdataflow.com/ecancrusher/

4.2.1 Importing D3

An EPUB publication has no online capabilities, so all JavaScript functionalities must be downloaded and included in the folder of the publication. Those scripts have to be declared on the manifest to be used.

To use D3, it must be downloaded into a JS file and introduced in the folder. In this example, it is stored in the directory script/d3/d3.js. This JavaScript file contains a minimal but sufficient version of the D3 library, and it is distributed on the official website. In a similar way, any other library can be included in an EPUB publication.

This file has to be included in the manifest, as a javascript file, but it does not need the scripted property, as the execution of the code takes place in the XHTML file. The line where the D3 gallery is included in the manifest of the example is presented below:

Listing 5: Inclusion of D3 into the EPUB manifest.

```
<item id="d3" href="script/d3/d3.js" media-type="text/javascript"/>
```

4.2.2 Creating the visualization

In this example, there is a bar chart showing dummy data, and the visualisations can be seen in the Apple iBooks environment and in the PocketBook e-reader. It is defined in its own JavaScript file, script/bar_chart.js, and it is called through the XHTML file.

The XHTML file imports the D3 library in the head of the file, so it will be available by any subsequent scripts that may be executed. Afterwards, in the body section, an SVG element is defined to hold the visualization. It could be also a div element, or a canvas, as long as it is taken into account in the visualization script (this is further discussed in a section below). Then, the script that creates the bar chart is executed, and it effectively changes the SVG element above, showing the visualization.

Listing 6: Inclusion of a data visualization script in the main XHTML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <meta charset="utf-8" />
  <style>
   .bar {
    fill: red;
   }
  </style>

  <script src="script/d3/d3.js"></script>
 </head>
 <body>
  <h1>Bar Chart</h1>
  <svg xmlns="http://www.w3.org/2000/svg" width="500" height="400">
  </svg>
  <script src="script/bar_chart.js"></script>
 </body>
</html>
```

The data visualization is created at script/bar_chart.js. Firstly, the SVG element is accessed, and its dimensions are defined. Afterwards, we apply the different steps required to create this specific visualization. Each case is different, but it works in the same way as JavaScript coding for web browsers, so tutorials, examples and other online resources can be followed. The data points are included as a variable, which is handy for a simple visualization with few data points, but it is not suitable for visualizations with a lot of data points. How to implement the later type of visualizations is discussed in the data pipeline section.

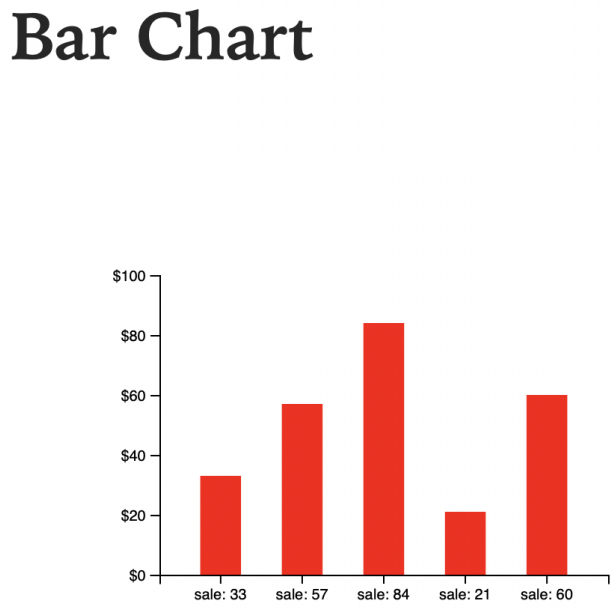


Figure 5: Bar chart visualization on Apple iBooks.

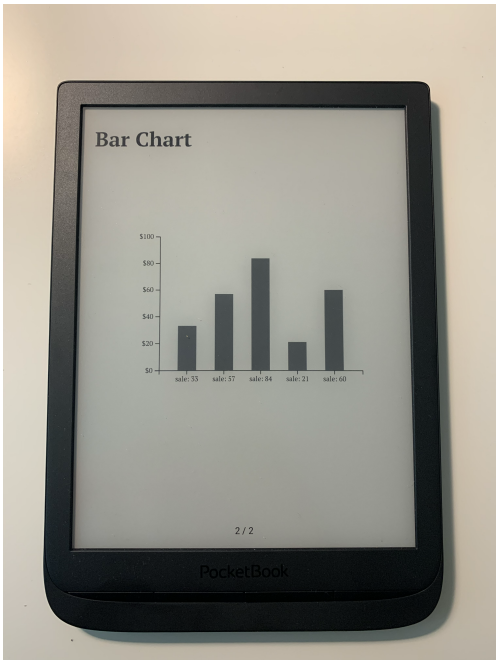


Figure 6: Bar chart visualization on PocketReader.

4.3 Data Pipeline

CSV files cannot be directly imported into an EPUB publication, as CORS[24] would prevent its use. The method *d3.csv()*, with which the file is loaded into the library, is expecting an URL, and it will not accept one that points to a local file. If the use case were web development, solutions like installing a web server or uploading the file online could solve this problem, but they are not feasible if the file must remain offline.

The chosen solution consists of parsing the CSV file as a string variable. We can introduce the content of a CSV file as a string variable into a JS file, as shown in the listing below:

Listing 7: Contents of a CSV file as a JavaScript string variable.

```
var file = `date,value
2011-04-28,135.98
2012-04-28,12
2013-04-28,456
2014-04-28,9
2015-04-28,27
2016-04-28,200`;
```

This variable can be parse using the following JavaScript function:

Listing 8: Parsing function for CSV string variables.

```
function csvToArray(str, delimiter = ",") {
 // slice from start of text to the first \n index
 // use split to create an array from string by delimiter
 const headers = str.slice(0, str.indexOf("\n")).split(delimiter);

 // slice from \n index + 1 to the end of the text
 // use split to create an array of each csv value row
 const rows = str.slice(str.indexOf("\n") + 1).split("\n");

 // Map the rows
 // split values from each row into an array
 // use headers.reduce to create an object
 // object properties derived from headers:values
 // the object passed as an element of the array
 const arr = rows.map(function (row) {
  const values = row.split(delimiter);
  const el = headers.reduce(function (object, header, index) {
   object[header] = values[index];
   return object;
  }, {});
  return el;
 });

 // return the array
 return arr;
}
```

Therefore, it is possible to introduce the contents of a CSV file into a data visualization by following these steps:

1. Copying the content of the CSV file into a string variable on a JavaScript file.
2. Declaring the *csvToArray()* function into the glsjavascript file of the data visualisation.
3. Import the file with the string variable into the HTML file.
4. Execute the function into the data visualization script and store its result into a variable.

That variable, called *data* in the provided example (the directory *data_pipeline* of the GitHub repository) can be then used normally with D3 function, and thus a data visualisation can be built.

4.4 Containers for the visualizations

There are two main elements which can contain a data visualization: SVG and canvas elements.

SVG elements are vector-based, so they do not lose resolution when resized. An SVG element is composed of multiple graphical elements that become part of the DOM tree of the page, and

for this reason, it is best suited for a limited number of elements. Data visualisations are easier to implement using this approach.

Canvas elements, on the other hand, are raster graphics images. It is processed as a single element, so it provides better performance for a large number of elements, as they get transformed into a single image. However, canvas can only be modified via scripting (SVG elements offer more input options). The created elements do not get added to the DOM, therefore being agnostic of DOM changes, which makes them more secure to use.

D3 can also be sued to create data visualisations in canvas elements. There is an example of how to create a visualization in this element, created with D3, in *canvas_ example/* (see Figure 7). D3 graphs can be embedded in canvas without changing the main code showed above, but using canvas element instead of SVG in the JavaScript code.

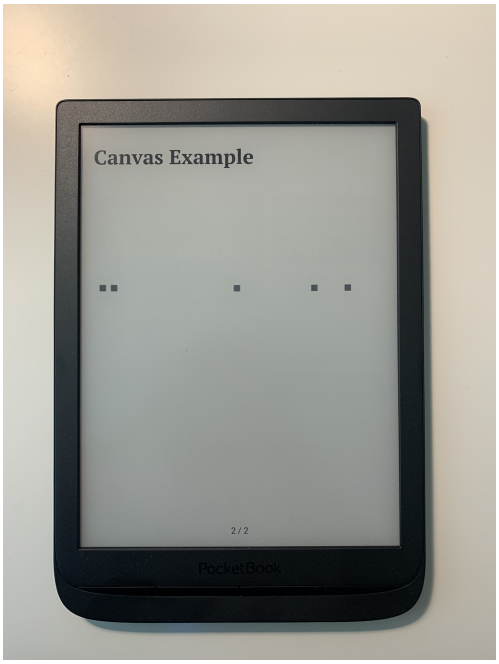


Figure 7: One-dimensional visualization using D3 over a canvas element.

5 Visualization in EPUB files using other libraries

D3 is one of the most popular data visualization libraries available on JavaScript, but there are other options that can be explored. In this section, three other popular libraries will be covered: Vega, Plotly and Apache eCharts. Vega and Plotly are compatible with Apple iBooks, but eCharts is not supported by any of the reading systems that have been covered in this project. Knowing which of their dependencies is not supported by the reading systems is a difficult and obscure task, but it is the most likely reason. These libraries are based mainly on D3, but include other dependencies to further push their capabilities.

5.1 Vega

Vega is based on D3, and it is intended to be a higher-level approach to Data Visualisation. Instead of it being based on the JavaScript syntax, it is a visualisation grammar; a declarative language for creating visualization designs. Visualizations in Vega are described in a JSON format, and generated using either canvas or SVG [25].

This library is a good option for generating visualizations in a serialised manner, from pre-generated JSON files. However, from the platforms that have been tried in this project, it is only supported by Apple iBooks. There is an example available on the GitHub page of the project, in *different-libraries/gallery-vega*. The JSON variable is included in the XHTML file, and from there Vega library generates the visualization, which can be seen below:

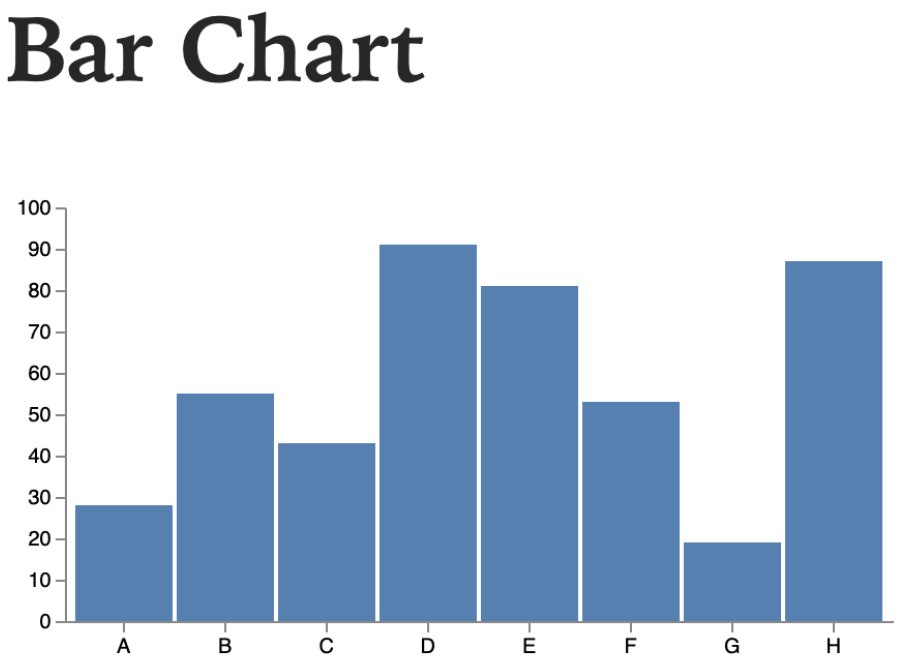


Figure 8: Bar chart visualization created using Vega.

5.2 Plotly

It is built on top of D3 and stack.gl as a declarative, high-level charting library. Plotly supports more than forty chart types, including 3D options, statistical graphs and SVG maps. Its syntax, while being based on D3, is slightly different, with a more straightforward approach [26].

An example of a mixed graph between a scatterplot and a line chart has been created and can be seen in *different-libraries/gallery-vega*. Of the platforms that have been tried in this project, it is only supported by Apple iBooks. The obtained visualization is interactive and offers a menu functionality.

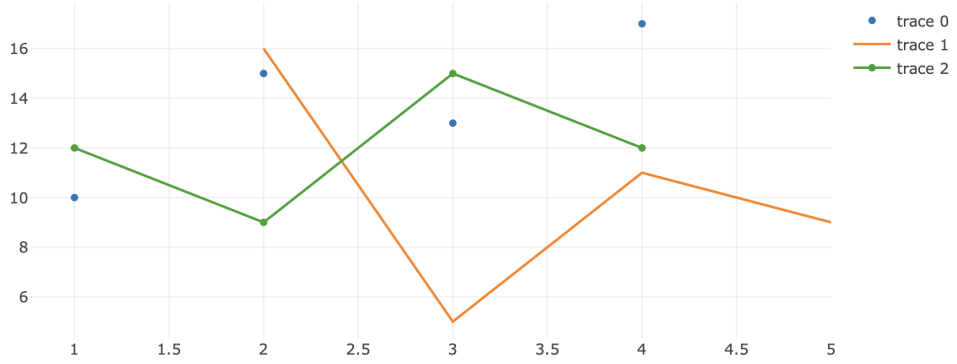


Figure 9: Line chart and scatter-plot visualization created using Plotly.

5.3 Apache ECharts

Apache ECharts is a charting and visualization library that offers interactive and customizable charts. It is written in JavaScript and based on ZRender[18], with a focus on being light-weighted [27]. Apache ECharts seems to not be compatible with any of the reading systems taken into account for this project, so no results have been obtained.

[18] `https://www.npmjs.com/package/zrender`

6 Gallery of examples

This section is dedicated to showcase several visualizations developed for EPUB files. The code used to create this publication is available on the GitHub page, in the *viz_ gallery* file [19]. For each visualization, a figure corresponding to the Apple iBooks output and the PocketReader e-reader is shown.

6.1 Scatterplot

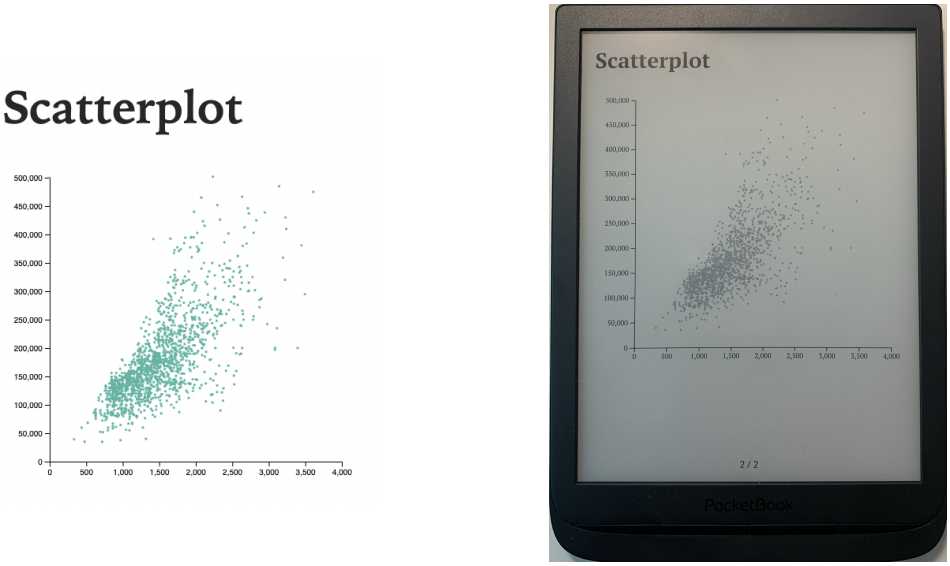


Figure 10: Scatterplot in iBooks and PocketReader

6.2 Line Chart

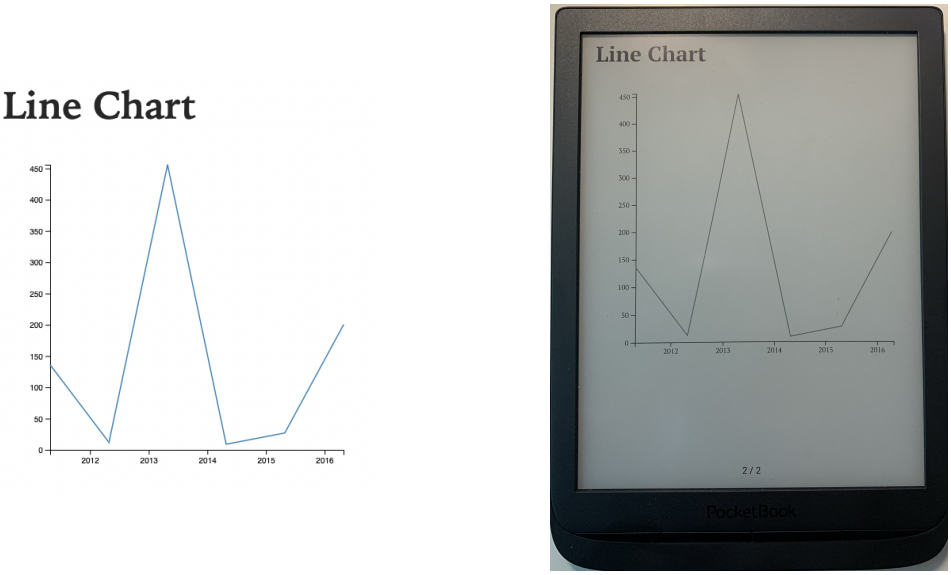


Figure 11: Line chart in iBooks and PocketReader

[19] `https://github.com/ignacioct/Visualization-EPUB/tree/main/viz_gallery`

6.3 Pie Chart

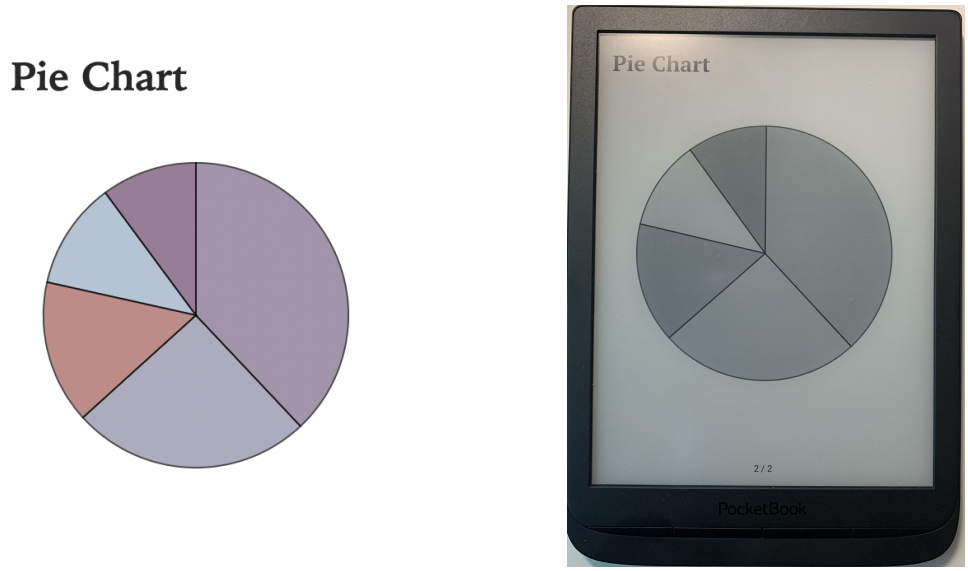


Figure 12: Pie chart in iBooks and PocketReader

6.4 Boxplot

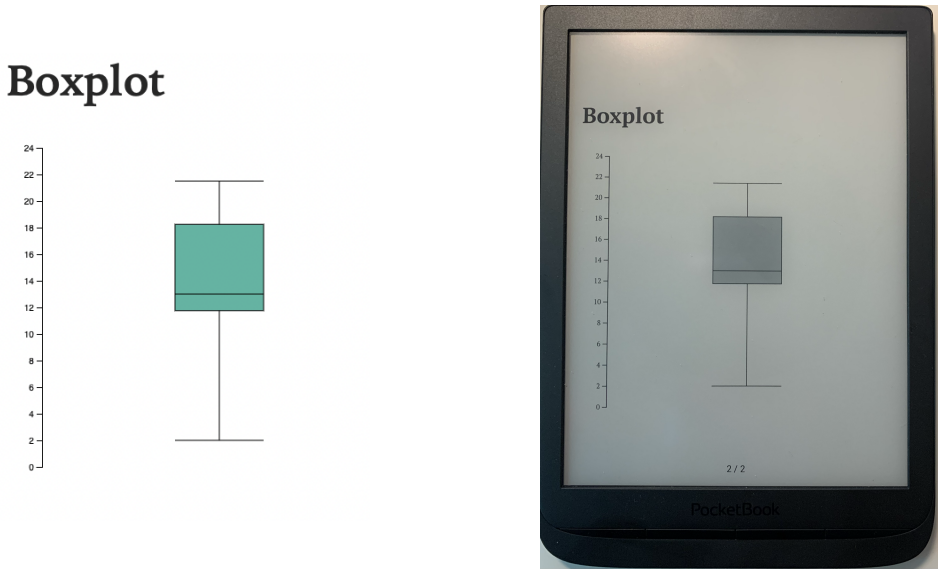


Figure 13: Boxplot in iBooks and PocketReader

6.5 Dendogram

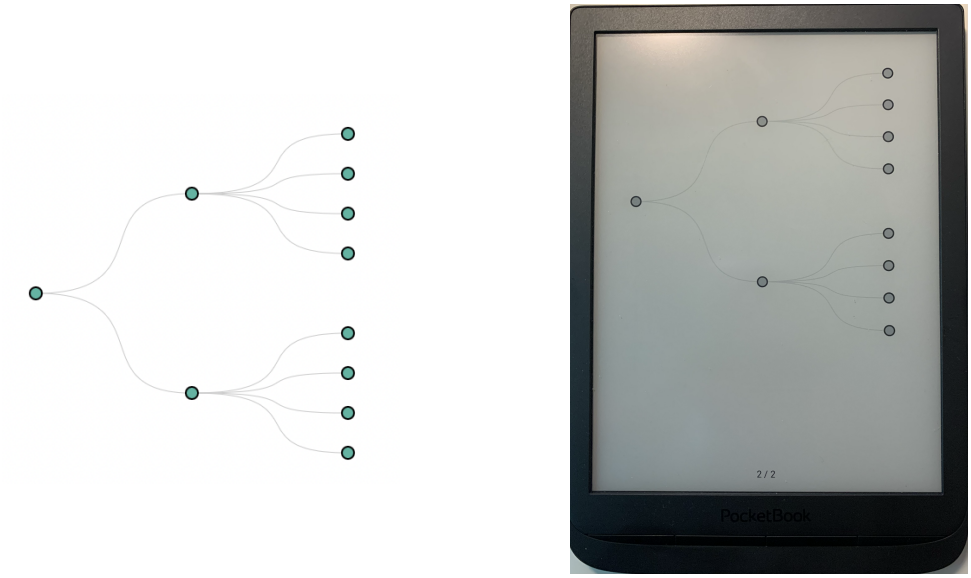


Figure 14: Dendogram in iBooks and PocketReader

6.6 Ridgeline

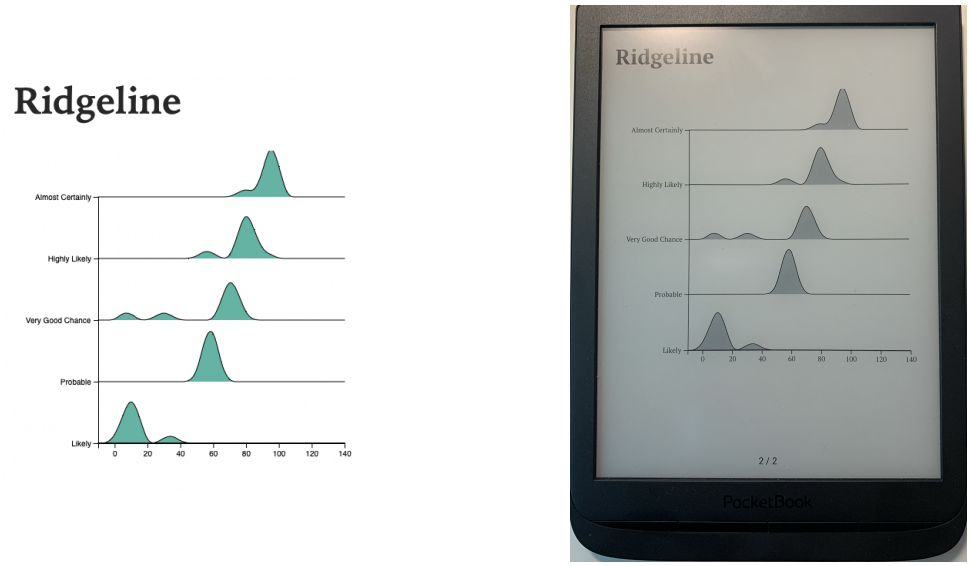


Figure 15: Ridgeline in iBooks and PocketReader

6.7 Streamgraph

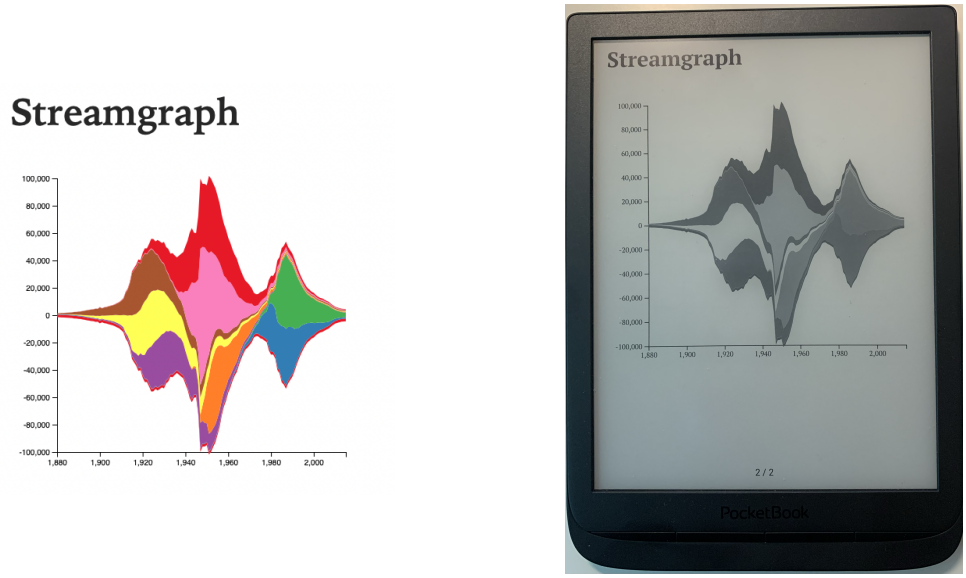


Figure 16: Streamgraph in iBooks and PocketReader

6.8 Arc Diagram

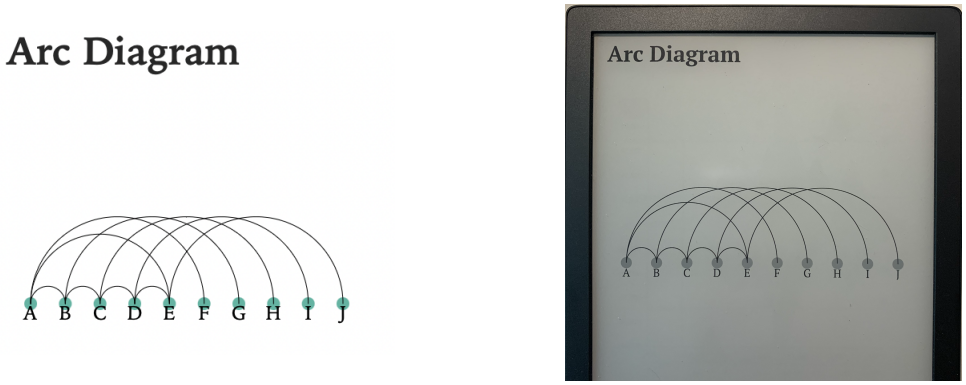


Figure 17: Arc diagram in iBooks and PocketReader

6.9 Treemap

Figure 18: Treemap in iBooks and PocketReader

7 Conclusions

In this student project, we have firstly made a comprehensive explanation of how EPUB publications work and how they can, theoretically, use JavaScript code to enhance the reading experience and use the resources available in computers, phones, tablets and e-readers. Afterwards, a methodology for embedding data visualizations using the D3 library is presented, showing how to import the library, how to build the publication, how to create a data pipeline and what are the differences between the different containers available for the visualization, providing different examples. Further research has been made for other different libraries. A gallery of data visualizations is provided, with examples for several different plots. A GitHub repository supports all the examples provided, with the files that create the different visualizations.

8 Future Work

This work is intended to serve as a foundation to push the capabilities of JavaScript and Data Visualisation in the EPUB format forward. To do so, a further study on compatibility has to be made. Usually, manufacturers of EPUB reading system do not disclose the different options that are available in terms of scripting, which makes the scripting process obscure and difficult. The EPUB standard should include options in terms of scripting and visualization that all readers must comply, to standardize the process.

Further research can also be made both on the practical consequences of the usage of the different HTML containers and the possibilities of interactivity. Even though most of the physical devices for reading EPUB files do not support touch input, iBooks and other platforms do support click input, and thus the possibility for interactivity is open. These two topics are very related, as interaction on an element that is directly embedded to the DOM could have a profound impact on the structure of the whole publication, and therefore constrain the possible reach of the interactivity by constraining this element in a container could be necessary.

In terms of the visualizations provided, further work can improve the results, creating more complex visualizations and involving other libraries and technologies in the process. Most of the resources of Data Visualisation for web applications can be used to start this process.

Acronyms

CORS Cross-Origin Resource Sharing. 17

CSS Cascading Style Sheets. 1, 5, 7, 9, 10, 13, *Glossary:* CSS

CSV Comma-Separated Values. 17, 18, *Glossary:* CSV

DOM Document Object Model. 13, 18, 19, 27, *Glossary:* DOM

EPUB Electronic PUBlication. 1, 4, 5, 7, 9–15, 17, 22, 27, *Glossary:* EPUB

HTML HyperText Markup Language. 1, 5, 7, 9–12, 15, 18, 27, *Glossary:* HTML

JS JavaScript. 4, 5, 7, 9, 10, 13, 15, 16, 18, *Glossary:* JavaScript

PDF Portable Document Format. 4, *Glossary:* PDF

SVG Scalable Vector Graphics. 4, 10, 18–20, *Glossary:* scalable vector graphics

XML eXtensible Markup Language. 10, *Glossary:* Extensible Markup Language

Glossary

canvas HTML element to draw graphics using a JS script. It is processed as a raster image.. 18–20

content.opf OPF file that combines both the metadata and the manifest files. 11

CSS It is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. 1

CSV Delimited text file that uses a comma to separate values. Each line of the file is a data record.. 17

D3 D3.js, short for Data-Driven Documents, is a JavaScript library for producing dynamic, interactive data visualizations. It makes use of SVG, HTML, CSS standards. 1, 5, 9, 13–15, 18–20, 27

Data Science Interdisciplinary field that uses methods, processes and algorithms to extract knowledge and insights from data and apply knowledge and actionable insights from data across a broad range of domains. 9

Data Visualisation Graphical representation of information and data by using visual elements like charts, graphs and maps. 1, 7, 9, 20, 27

DOM Document Object Model, it is a programming API for HTML and XML documents that defines their logical structure and the way they are accessed and manipulated. 13

EPUB EPUB is an e-book file format that is currently the standard, and it is supported by many devices and reading software. 1

Extensible Markup Language It is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It was designed to be both machine-readable and human-readable. 10

HTML It is standard markup language for documents designed to be displayed in a web browser. 1

JavaScript JavaScript, often abbreviated as JS, is a programming language that, alongisde HTML and CSS, compose the core of website, as over 97% of website use it on client side. All major web browsers have a dedicated JavaScript engine. 1, 4, 5, 9–12, 14–16, 18–21, 27

PDF Portable Document Format (PDF), standardized as ISO 32000, is a file format developed by Adobe in 1992 to present documents in a manner agnostic to software, hardware and operating systems. Each PDF files encapsulates a description of the layout, the text, the fonts, the graphics, and all the information needed to visualize it. 4

raster graphics Raster graphics are based on pixels. Each image can only contain a fixed number of pixels; the amount of pixels determines the quality of the image. 4, 10, 19

scalable vector graphics Images based on geometrical shapes that can be expressed as mathematical expression, and therefore can be stored in less size and can be scaled and transformed without losing quality. 4

Scripted Content Document An EPUB document that contains Javascript scripting. 9, 11

References

[1] H. Patel and P. Morreale, "Education and learning: Electronic books or traditional printed books?" *J. Comput. Sci. Coll.*, vol. 29, no. 3, p. 21–28, jan 2014.

[2] G. Bhatta, "E-ink (electronic ink)," *International Journal of Computer Science and Engineering (IJCSE)*, vol. 6, no. 1, pp. 51–66, 2017.

[3] Y. Zhong, T. Isenberg, and P. Isenberg, "Black-and-white textures for visualization on e-ink displays," in *Posters of IEEE Visualization*, 2020.

[4] J. Heikenfeld, P. Drzaic, J.-S. Yeo, and T. Koch, "A critical review of the present and future prospects for electronic paper," *Journal of the Society for Information Display*, vol. 19, no. 2, pp. 129–156, 2011.

[5] A. Klein, "A new printing technology sets off a high-stakes race," Jan 2000. [Online]. Available: https://www.wsj.com/articles/SB946939872703897050?reflink=desktopwebshare_permalink

[6] L. Owen, *Selling rights*. Routledge, 2014.

[7] M. Rollins, *Taking Your Kindle Fire to the Max*. Springer, 2012.

[8] M. Kozlowski, "Pocketbook inkpad x review," Apr 2021. [Online]. Available: https://goodereader.com/blog/electronic-readers/pocketbook-inkpad-x-review

[9] "Pebble teardown," Apr 2020. [Online]. Available: https://www.ifixit.com/Teardown/Pebble+Teardown/13319

[10] "Press release," Jan 2006. [Online]. Available: https://web.archive.org/web/20150505013110/http://www.eink.com/press_releases/lexar_usb_drive_using_e_ink_010306.html

[11] S. Claes, J. Coenen, and A. Vande Moere, "Empowering citizens with spatially distributed public visualization displays," 06 2017, pp. 213–217.

[12] J. Coenen, M. Houben, and A. Vande Moere, "Citizen dialogue kit: Public polling and data visualization displays for bottom-up citizen participation," 06 2019, pp. 9–12.

[13] K. Klamka and R. Dachselt, *Bendable Color EPaper Displays for Novel Wearable Applications and Mobile Visualization*. New York, NY, USA: Association for Computing Machinery, 2021, p. 6–10. [Online]. Available: https://doi.org/10.1145/3474349.3480213

[14] Y. Zhong, "Studying black and white textures for visualization on e-ink displays," Master's thesis, University of Twente, 2020.

[15] A. Jain, "Data visualization with the d3. js javascript library," *Journal of Computing Sciences in Colleges*, vol. 30, no. 2, pp. 139–141, 2014.

[16] K. Dale, *Data visualization with python and javascript: scrape, clean, explore & transform your data*. " O'Reilly Media, Inc.", 2016.

[17] M. Garrish and M. Gylling, *EPUB 3 Best Practices: Optimize Your Digital Books*. O'Reilly Media, 2013. [Online]. Available: https://books.google.dk/books?id=z0zpvR5QNvsC

[18] "Epub 3.3 scripting manifest." [Online]. Available: https://www.w3.org/TR/epub-33/#scripted-contexts-example

[19] "Epub 3 samples project." [Online]. Available: https://idpf.github.io/epub3-samples/30/feature-matrix.html#Scripted_Content_Documents

[20] "Older versions of epub." [Online]. Available: https://web.archive.org/web/20170831011752/http://idpf.org/epub-older-versions

[21] E. Cavanaugh, "Learning about epub: Structure and content," Mar 2016. [Online]. Available: https://www.altova.com/blog/learning-about-epub-structure-and-content/

[22] D. Price, "The different ebook formats explained: Epub, mobi, azw, iba, and more," Jul 2018. [Online]. Available: https://www.makeuseof.com/tag/ebook-formats-explained/

[23] "Anatomy of an epub." [Online]. Available: https://www.edrlab.org/open-standards/anatomy-of-an-epub-3-file/

[24] Cross-origin resource sharing (CORS) - HTTP | MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/CORS

[25] A visualization grammar. [Online]. Available: https://vega.github.io/vega/

[26] Plotly. [Online]. Available: https://plotly.com/javascript/

[27] Apache ECharts. [Online]. Available: https://echarts.apache.org/en/index.html