Degree project

UNIVERSITY
OF SKÖVDE

Comparing the total response times between SVG-based D3.js and WebGL-based Three.js and Pixi.js when loading and rendering PNG images

Bachelor Degree Project in Information Technology
First Cycle 30 credits
Spring 2026
Student: Edvin Heyman Tolestam
Supervisor: András Márki
Examiner: Yacine Atif

Abstract

PNG images are widely used across the internet today. Loading and rendering PNG images can take a long time depending on factors like the number of pixels and redundancy. It is therefore important to know what tool to use when loading and rendering PNG images, as it can otherwise lead to longer total response time and therefore dissatisfaction for the user. This research aimed to find out which JavaScript library out of SVG-based D3.js, WebGL-based Three.js, and WebGL-based Pixi.js has the shortest total response time when loading and rendering PNG images. This aim was achieved through a quasi-experiment that included selecting a dataset, loading and rendering the dataset using the three libraries, and performing a statistical analysis. The results pointed to D3.js having shorter total response times than Three.js and Pixi.js. These results imply that it is time-saving for the user to choose D3.js over Three.js and Pixi.js. Future work includes testing on more libraries, different image formats, and more iterations for each image.

Table of Contents

1 Background **1**
1.1 JavaScript 1
1.2 SVG 1
1.3 WebGL 2
1.4 Related Research 2
2 Problem Formulation **4**
2.1 Aim and Research Question 4
2.2 Objectives 5
2.3 Hypothesis 5
3 Method and Implementation **6**
3.1 Method Type 6
3.2 Alternative Methods 6
3.3 Procedure 6
3.4 Data Collection 7
3.5 Data Analysis 7
3.6 Dataset 7
3.7 Gathering of Map Images 7
3.8 Gathering the Results 9
4 Results **11**
4.1 Running the Experiment 11
4.2 Organizing in Excel 11
4.3 Analyzing the Results 11
5 Discussion **20**
5.1 Results 20
5.2 Threats to Validity 20
5.2.1 Internal Validity 20
5.2.2 External Validity 21
5.2.3 Conclusion Validity 22
5.2.4 Construct Validity 23
5.3 Ethical and Societal Considerations 23
5.3.1 Research Ethics 23
5.3.2 Effects on Society 24
5.3.3 Authorship 24
5.4 Generalization 24
5.5 Future Work 25
6 Conclusion **26**

1 Background

Today, images and videos are used by a wide range of users all across the internet (Guilbeault et al., 2024). They make webpages more appealing, attractive, and engaging (Dornauer & Felderer, 2023). In fact, as of 2023, 41 percent of transmitted data was made up of images on the web (Dornauer et al., 2023). Users tend to be drawn to images before they are drawn to normal text (Dornauer et al., 2023). These images must be loaded and rendered quickly. A delay of even a fraction of a second can cause user dissatisfaction (Arapakis, Park & Pielot, 2021). This can negatively impact business outcomes, with users leaving for alternative websites.

It is therefore important to be able to load and render the images as quickly as possible (Arapakis et al., 2021). One web image format that dominates the internet is the PNG format. It is an image format that is raster-based (Zhou et al., 2023).

1.1 JavaScript

Loading and rendering media such as images can be done with JavaScript. JavaScript is a language that combines object-oriented and functional programming models. JavaScript is one of the most popular programming languages today, and most dynamic websites are developed using JavaScript (Sharov et al., 2024), which holds a high position in international rankings (Sharov et al., 2024). Consequently, JavaScript is a highly relevant programming language. Additionally, JavaScript enjoys native support in web browsers (Wagner, 2016). It is a language that is very dynamic, meaning that programmers do not need to use type annotations (Andreasen et al., 2017).

JavaScript is used to create, for example, interactive websites, online games, and email clients (Andreasen et al., 2017). It is pivotal to have some background knowledge about JavaScript since it is highly relevant for this project.

Visualization tools that are based on JavaScript give interactive and meaningful visualizations, meaning it is easier to analyze and understand the data (Soylu & Das, 2025). This, together with the many libraries that focus on visualization offered by JavaScript, is the reason why JavaScript is used for this project.

Two common ways of visualization are raster graphics and vector graphics. The difference between raster graphics and vector graphics is that in raster graphics, all tiles are generated according to a standardized scheme, while vector graphics are made up of vector geometry (Netek et al., 2020). Within visualization in web development, SVG and WebGL are two visualization tools that both provide powerful visualization capabilities (Garret, 2025), and many JavaScript libraries are based on these two visualization tools. SVG uses vector graphics, and WebGL uses raster graphics.

1.2 SVG

Scalable vector graphics, SVG for short, is a markup language that is dominant within visualizations on the web. SVG offers graphics that are 2D, animated, and interactive (Quint, 2003). The main advantages of SVG are that it is relatively easy to learn, open source, and has a document object model, DOM for short, that is accessible and searchable (Schwab et al., 2021).

The reason why SVG was created was to give a generic solution to graphics that is document-oriented and that can be used for media today (Quint, 2003). The disadvantages of SVG, however, are that if the dataset becomes too large, it can lead to the visualizations becoming increasingly slow, and that only a couple of hundred nodes can lead to significantly slower performance (Schwab et al., 2021).

SVG is organized by an XML structure and defines shapes that are DOM elements, instead of individual pixels (Tapflare, 2025). SVG keeps every object in memory, which can be a problem if you have a large number of objects. This is why SVG works better for smaller visualizations (Tapflare, 2025).

SVG is used in several JavaScript libraries such as D3.js, which can be used to render PNG images. D3.js enables fast development since data is bound to visual mark objects such as “circle” in the SVG DOM (Schwab et al., 2021). Often, D3.js generates SVG charts with the help of a declarative interface (Tapflare, 2025). While SVG is primarily used to render vector-based images, D3.js can still be used to render raster-based PNG images (Afonso et al., 2022; Schwab et al., 2021).

1.3 WebGL

Web Graphics Library, WebGL for short, is a JavaScript API used for rendering both 3D and 2D graphics within a web browser (Han et al., 2025). WebGL uses a rendering pipeline, whose task is to color the elements and to turn the 3D coordinates into screen space coordinates (Liu et al., 2024).

The advantage of WebGL is that it is supported by many different browsers such as Google Chrome, Firefox, and Internet Explorer, to name a few. Another advantage of WebGL is that it is very powerful when rendering a large amount of data (Tapflare, 2025). One downside to WebGL, however, is that the right hardware needs to be used in order to use it.

The WebGL API gives hardware-accelerated access to the GPU (Tapflare, 2025). Its sheer strength, together with the GPU, means that it is more powerful than SVG when rendering a large number of shapes.

One JavaScript library where WebGL is used is Three.js. Three.js is a high-level 3D engine that is built on top of WebGL and enables the user, with the help of JavaScript, to build 3D scenes with lights, cameras, and more (Tapflare, 2025). While Three.js is mainly used for 3D visualizations, it can also be used to render 2D visualizations. Another library that uses WebGL is Pixi.js. Pixi.js is a 2D graphics library that is mostly made for animations with high performance, such as video games, where it is possible to handle sprites with the help of an API that is easy to use (Tapflare, 2025).

1.4 Related Research

This project was inspired by an earlier bachelor’s thesis project by Myrén (2025). Similarly to this project, the author renders images and measures the loading time of the formats JPEG, PNG, and WEBP using different JavaScript frameworks.

Earlier research has directly compared and benchmarked SVG and WebGL. This includes a study by Koren Ivančević, Ježić & Stanić Loknar (2026) that compares animation

performance between different visualization tools, including SVG and WebGL. The authors measure, for example, framerate, GPU usage, and CPU utilisation. They conclude that SVG and DOM-based technologies are easy to implement and powerful for light visualizations, but start to deteriorate as the visualization becomes heavier. WebGL, on the other hand, is better at rendering large-scale visualizations. While their research is focused on animation and not static PNG images, it compares SVG and WebGL in a related way. What has not been investigated, however, is a direct comparison between the total response times of D3.js, Three.js, and Pixi.js when loading and rendering PNG images.

Another related study is Lukács & Gartus (2023), where they try to find a better way of timing in browser-based experiments. It compares whether a nested RAF function or a contiguously running RAF loop has the best precision to give the users, in this case researchers, the best tool for timing. The authors conclude that a continuously running RAF loop has better precision than a nested RAF function. The article connects to this study since it is about measuring time in a web application, as was done in this experiment.

A third related article explores how response latency affects users when it comes to mobile web search (Arapakis et al., 2021). The authors found that when the total response time exceeds 7-10 seconds, the user reports feeling tense and frustrated, leading to a significantly worse user experience. The article is interesting to this project since it includes both loading time and rendering time as their total response time, and therefore captures the total user experience. It is interesting to this research to know how much time needs to elapse before the user experience becomes worse.

One final related work is an article made by Moura et al. (2025). It contributes a comparative analysis of the latency, frame rate, and memory usage in several JavaScript frameworks, including React.js, Vue.js, and Angular.js, when it comes to real-time image streaming in Robot Operating System 2. They found that React.js and Vue.js had less latency than Angular.js. The related study is similar to this study since both evaluate latency regarding visualization with different JavaScript libraries.

2 Problem Formulation

As stated in the background literature, the loading and rendering of PNG images can become problematic if processing latency degrades the user experience (Arapakis et al., 2021). PNG files scale in computational weight as pixel counts increase, specifically when encountering high color variation and low redundancy (Regoršek, Gorič & Trost, 2024).

Prior research, such as the study by Koren Ivančević et al. (2026), has focused on animation performance when comparing SVG and WebGL as raw visualization tools, yet it overlooks the specific library frameworks built upon them. Consequently, a distinct research gap exists concerning the visualization of PNG images using the specific libraries D3.js, Three.js, and Pixi.js.

Selecting the appropriate visualization tool depends heavily on the application's intended purpose. While one tool may excel at loading and rendering flat PNG images, another might be better suited for rendering 3D video game assets (Garret, 2025). Investigating SVG and WebGL-based libraries is highly relevant because both are powerful, widely adopted frameworks that operate on fundamentally different rendering models, which can lead to significant differences in total response times (Garret, 2025).

These underlying technologies possess distinct trade-offs regarding performance and ease of use (Schwab et al., 2021; Koren Ivančević et al., 2026). SVG-based approaches are typically suited for visualizations of lower complexity, whereas WebGL is built for complex tasks like interactive environments and real-time gameplay mechanics (Koren Ivančević et al., 2026). Although both technologies can become performance bottlenecks when processing heavy image data, SVG elements tend to experience performance degradation much sooner than WebGL (Koren Ivančević et al., 2026).

What remains missing from existing literature is a direct comparison of total response times when loading and rendering PNG images across D3.js, Three.js, and Pixi.js. To address this, the contribution of this project is a statistical performance comparison of total response times between the SVG-based D3.js library and the WebGL-based Three.js and Pixi.js libraries. While generic SVG and WebGL benchmarks exist, the novelty of this study lies in isolating the empirical performance of these specific, industry-standard library abstractions. Total response time is interesting because if it takes too long, it will lead to a bad user experience.

2.1 Aim and Research Question

This project aims to compare the total response times between different types of JavaScript libraries, namely the SVG-based library D3.js and the WebGL-based libraries Three.js and Pixi.js, when loading and rendering PNG images representing survey maps.

The research question for this study is "How does the total response time differ between SVG-based D3.js, WebGL-based Three.js, and WebGL-based Pixi.js when loading and rendering PNG images?"

2.2 Objectives

The objectives that need to be addressed to answer the research question are:

- To compare the total response time of the three different JavaScript libraries D3.js, Three.js, and Pixi.js when loading and rendering PNG images
- To iterate the experiment 400 times to increase the precision
- To analyze whether there is a statistical difference between the three different libraries

2.3 Hypothesis

The null hypothesis is that there is no statistically significant difference in total response time between SVG-based D3.js, WebGL-based Three.js, and WebGL-based Pixi.js.

The alternative hypothesis is that there is a statistically significant difference in total response time between SVG-based D3.js, WebGL-based Three.js, and WebGL-based Pixi.js.

3 Method and Implementation

3.1 Method Type

The chosen method for this study was a quasi-experiment. A quasi-experiment means that the independent variables are changing, which in this case are the chosen libraries, but the other elements are stable, including the images, hardware, and browser (Wohlin et al., 2012). By alternating the libraries in the experiment while keeping the other variables stable and fixed, one was able to make sure that any observed difference in total response times was caused only by the libraries themselves. The study was quantitative, meaning that the results were measured numerically rather than through subjective thoughts and opinions.

The independent variables in this experiment were the three different libraries, D3.js, Three.js, and Pixi.js, as well as the four different numbers of images that were loaded and rendered. The dependent variables were the results of the experiment, including the total response times for each image and library, measured in milliseconds. This can also be seen as a cause-effect relation where the libraries and numbers of images are the cause that affects the effect, which in this case was the total response times.

3.2 Alternative Methods

Regarding alternative methods, one method that could have been used is a controlled experiment. This method includes randomizing treatments, meaning that the image chosen to be loaded and rendered for a certain iteration would be random. All three libraries still load and render the same images in the same order, but this order becomes random. This is contrary to the current method, which was a quasi-experiment, where the images were loaded and rendered in order from the first image to the last image, without any randomization. During early stages of development, a controlled experiment was considered as an alternative to the quasi-experiment.

Another alternative method that could have been used to complement the quasi-experiment is to do a quantitative questionnaire, such as a user survey, given to people following certain selection criteria, such as developers or users. Doing a quantitative questionnaire would measure how smoothly users experience loading and rendering by using an already established index, such as a scale. However, a questionnaire only captures subjective human experiences rather than precise empirical data measured in milliseconds.

3.3 Procedure

When performing the experiment, 108 PNG images representing maps were gathered from the public land surveying map database (Lantmäteriet, n.d.). These map images were visualized using the libraries D3.js, which uses SVG (Schwab et al., 2021), Three.js, which uses WebGL, and Pixi.js, which also uses WebGL. (Tapflare, 2025). All map images were loaded and rendered with JavaScript inside Visual Studio Code.

For each of the three different libraries, the first 27 map images were loaded and rendered 100 times. After that, the first 54 map images were loaded and rendered 100 times. After that, the first 81 map images were loaded and rendered 100 times. Lastly, all 108 map images were loaded and rendered 100 times. Between each run, defined as the loading and

rendering of a full image group consisting of 27, 54, 81, or 108 images by a single library, the browser cache was cleared. This ensured that assets were not retrieved from memory during subsequent sessions.

3.4 Data Collection

Data was collected by carrying out the experiment using “performance.now”, including both the loading time and the rendering time of the map image. The rendering times, including the loading time for all map images with each library, were measured. A supplementary experiment was performed in which loading time was excluded and only rendering time was included. However, most of the results for D3.js became extremely low, below one millisecond or even zero milliseconds. This negligible duration did not accurately reflect the real-world total response time experienced by a user. It was therefore decided to measure the total response time, including loading time. Each value of total response time was imported into an Excel document.

3.5 Data Analysis

Lastly, the data from the experiment was analyzed statistically to compare which library had the shortest total response time. First, a descriptive analysis was done to summarize and graph the collected data. Mean value, standard deviation, maximum value, and minimum value were calculated. An inferential analysis was then performed using a one-way ANOVA, a Welch one-way ANOVA including a Tukey test, and lastly a two-way ANOVA. The analysis determined if the differences between the three libraries were statistically significant in order to generalize the results to the broader population beyond only the map images and libraries that were tested. After an analysis of each library’s total response times, additional statistical comparisons were completed for the total response time for each of the 27, 54, 81, and 108 map-image groups. All statistical analyses were performed in SPSS.

3.6 Dataset

The material used in this project included 108 PNG images representing maps. These 108 map images were gathered from the public land surveying map database (Lantmäteriet, n.d.). The map images covered two different scales as well as different file sizes. In the public land surveying map database, it was possible to choose a geographical area and generate 12 separate map images covering that area and choose between two different scales, 1:10 000 or 1:50 000. Out of the 108 map images that were going to be tested, 36 had the scale 1:50 000 and 72 had the scale 1:10 000. It was chosen to have slightly more map images at the scale 1:10 000 compared to 1:50 000.

3.7 Gathering of Map Images

The next step was to gather the map images from the public land surveying map database (Lantmäteriet, n.d.). On the website, 108 map images with the scales 1:50 000 and 1:10 000 were selected over the desired area and then saved as 9 PDF files with 12 map images in each. See Figures 1-3.

Figure 1. The total chosen area for the first 12 maps

Figure 2. Example of a map image with the scale 1:50 000

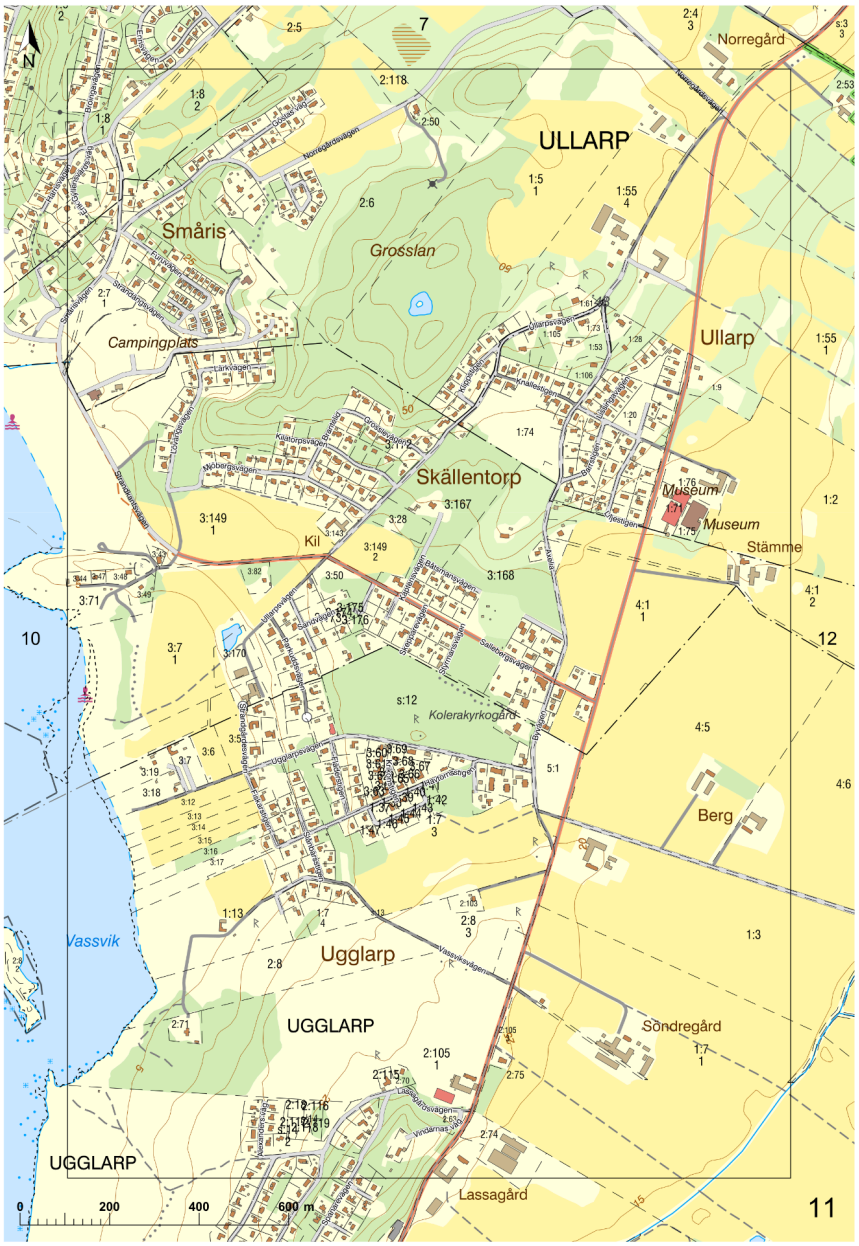

Figure 3. Example of a map image with the scale 1:10 000

After downloading, the map images in the PDF files were converted into PNG files. For this purpose, GIMP 3.2.2 was used. Each PDF file was imported one by one into GIMP and converted into 108 PNG images.

3.8 Gathering the Results

To prepare for the loading and rendering of the PNG images representing survey maps, Visual Studio Code was downloaded to the computer of choice, which was a PC with Windows 10 having an AMD Ryzen 7 1700X Eight-Core Processor, an NVIDIA GeForce GTX 1070 8GB graphics card, and 16 GB RAM. Live Server, an extension to Visual Studio Code, was also used in order to run the program. Two files were created inside Visual Studio Code,

an HTML file and a JavaScript file. The map images were then imported into the project. Inside the body of the HTML code, a dropdown menu and a start button were set up. A canvas element was created where the map images were supposed to be loaded and rendered. The three different libraries (Bostock & Observable, 2024; Cabello, 2026; Groves et al., 2024) and the JavaScript file were then imported into the HTML file. D3.js version 7.9.0, Three.js version 0.183.2, and Pixi.js version 7.4.2 were used.

In the JavaScript file, the code goes through and loads and renders the images with the selected library within a selected interval, whether it is the first 27, 54, 81, or 108 images. Two functions were created, used for starting and stopping the timing. The function used for starting the timing includes “performance.timeOrigin” plus “performance.now()”, which measures the elapsed time from the program starting to the function being called when the timer starts, measured in milliseconds. It was placed so that it included both loading time and rendering time, before the image had started to load. The function used for stopping the time also includes “performance.timeOrigin” plus “performance.now()” and therefore measures the elapsed time from the program starting to the function being called when the timer ends.

The function that starts the timing was placed right before the image is requested. The function that ends the timing was placed right after the image was rendered. The end time minus the start time was then calculated and saved together with the start time, stop time, and image number, as a CSV file. When writing the code that loads and renders the images using the three different libraries, the code looks slightly different, as the three different libraries work in three different ways.

4 Results

4.1 Running the Experiment

Applications and background processes that were not relevant for the experiment were closed so that they would not interfere with the experiment. Wifi was enabled to retrieve the libraries from the internet. The browser that was used for loading and rendering the images was Google Chrome.

For each library, the first 27 map images were loaded and rendered 100 times. Then the first 54 map images were loaded and rendered 100 times. After that, the first 81 map images were loaded and rendered 100 times. Lastly, all 108 map images were loaded and rendered 100 times. Between each run of the program, it was necessary to clear the cache so that the map images would not be saved or remembered for the next run. When all three libraries had been loaded and rendered, the results were saved as 1200 CSV files, one for each run and library.

4.2 Organizing in Excel

The CSV files were imported into Excel. For each library, there was one column for the name of the CSV file, one for the image number, one for the total response time, and one for the library that it was loaded and rendered by. There were also two extra columns containing all the total response times for all three libraries, as well as which library it was loaded and rendered by. Each row in the Excel document was assigned to each total response time for all three libraries.

4.3 Analyzing the Results

When analyzing the results, it was decided to use SPSS. A descriptive analysis was carried out for the total response times of all three libraries.

When looking at the mean values of the total response times for the three libraries, D3.js has a mean value of 29.50 milliseconds, Three.js has a mean value of 149.18 milliseconds, and Pixi.js has a mean value of 128.20 milliseconds. When looking at the standard deviation of the three libraries, D3.js has 13.44 milliseconds, Three.js has 30.62 milliseconds, and Pixi.js has a standard deviation of 36.09 milliseconds.

When looking at the min and max values, D3.js has a minimum value of 4.30 milliseconds and a maximum value of 115.10 milliseconds. Three.js has a minimum value of 87.00 milliseconds and a maximum value of 289.10 milliseconds. Pixi.js has a minimum value of 61.80 milliseconds and a maximum value of 436.10 milliseconds.

Table 1 outlines the mean values, standard deviations, and minimum and maximum ranges for each library. Additionally, Figure 4 presents a box plot illustrating the distribution and outliers of the iterations across all three libraries.

	D3.js	Three.js	Pixi.js
Mean value	29.50 ms	149.18 ms	128.20 ms
Standard deviation	13.44 ms	30.62 ms	36.09 ms
Minimum value	4.30 ms	87.00 ms	61.80 ms
Maximum value	115.10 ms	289.10 ms	436.10 ms

Table 1. A table showing the mean value, standard deviation, minimum value, and maximum value for the total response times of the three different libraries, measured in milliseconds (ms).

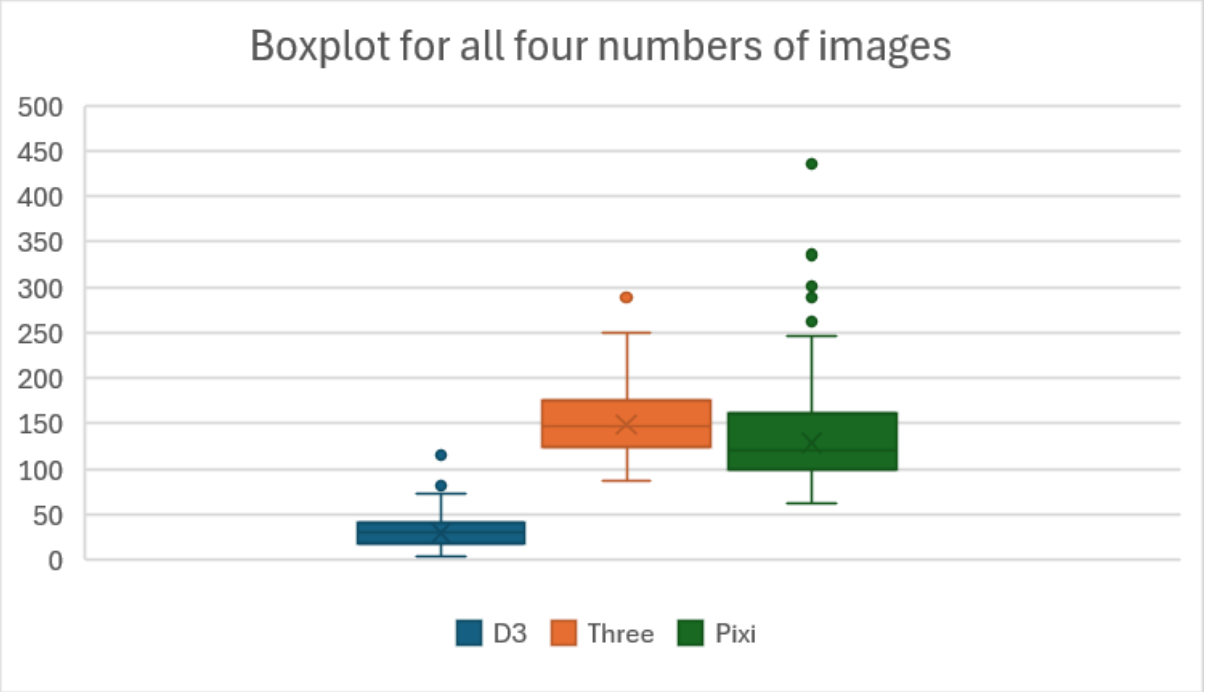

Figure 4. A box plot of all the total response times in milliseconds for the three different libraries, including outliers

Another descriptive analysis was done by separating the results into the four different numbers of images being loaded and rendered, that being 27, 54, 81, and 108 images.

When looking at the mean values of the total response times when loading and rendering 27 images, D3.js has a mean value of 34.95 milliseconds, Three.js has a mean value of 168.83 milliseconds, and Pixi.js has a mean value of 150.94 milliseconds. When looking at the standard deviation of the total response times when loading and rendering 27 images, D3.js has a standard deviation of 11.26 milliseconds, Three.js has a standard deviation of 27.84 milliseconds, and Pixi.js has a standard deviation of 32.86 milliseconds.

When looking at the minimum and maximum values of the total response times when loading and rendering 27 images, D3.js has a minimum value of 4.60 milliseconds and a maximum value of 67.40 milliseconds, Three.js has a minimum value of 89.80 milliseconds and a maximum value of 289.10 milliseconds, and Pixi.js has a minimum value of 64.30 milliseconds and a maximum value of 335.50 milliseconds.

Below is a boxplot showing how all iterations when loading and rendering with 27 images were distributed for all three libraries, including outliers. See Figure 5.

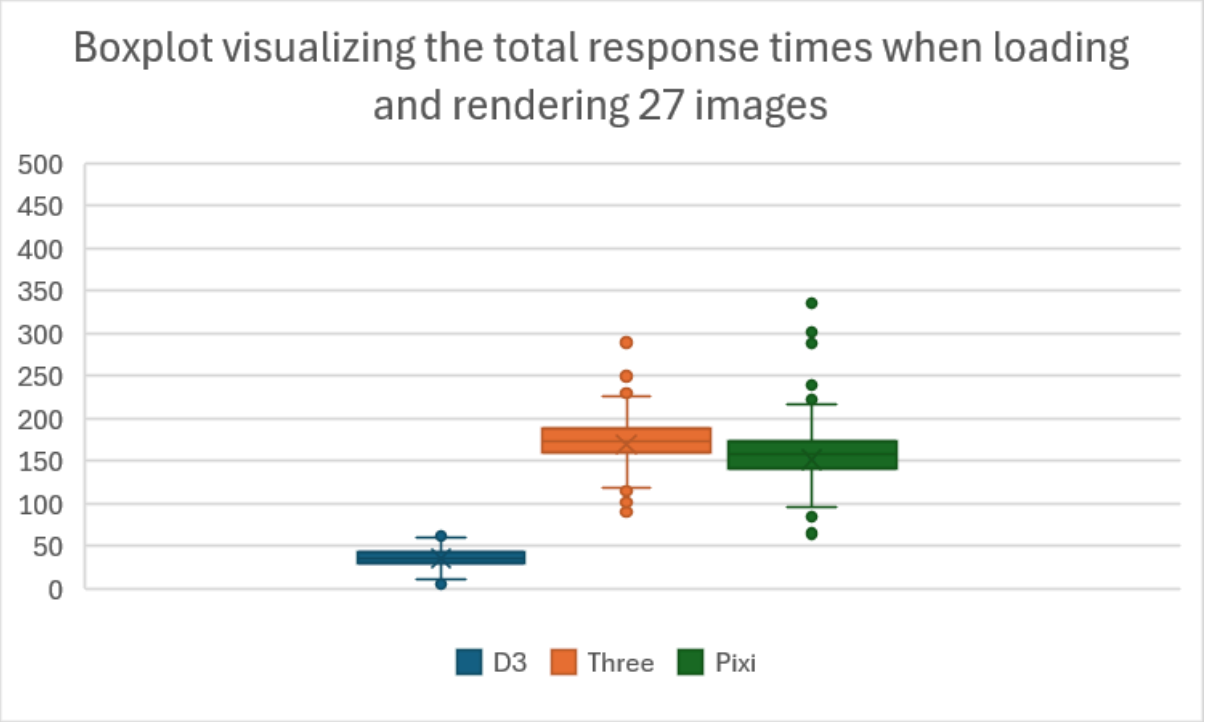

Figure 5. A box plot of the total response times in milliseconds when loading and rendering 27 images for the three different libraries, including outliers.

When looking at the mean values of the total response times when loading and rendering 54 images, D3.js has a mean value of 32.06 milliseconds, Three.js has a mean value of 155.95 milliseconds, and Pixi.js has a mean value of 135.75 milliseconds. When looking at the standard deviations of the total response times when loading and rendering 54 images, D3.js has a standard deviation of 13.90 milliseconds, Three.js has a standard deviation of 13.67 milliseconds, and Pixi.js has a standard deviation of 37.70 milliseconds.

When considering the minimum and maximum values of the total response times when loading and rendering 54 images, D3.js has a minimum value of 4.50 milliseconds and a maximum value of 70.90 milliseconds, Three.js has a minimum value of 89.40 milliseconds and a maximum value of 230.70 milliseconds, and Pixi.js has a minimum value of 64.40 milliseconds and a maximum value of 265.70 milliseconds.

Below is a boxplot showing the results for loading and rendering 54 images using the three libraries. See Figure 6.

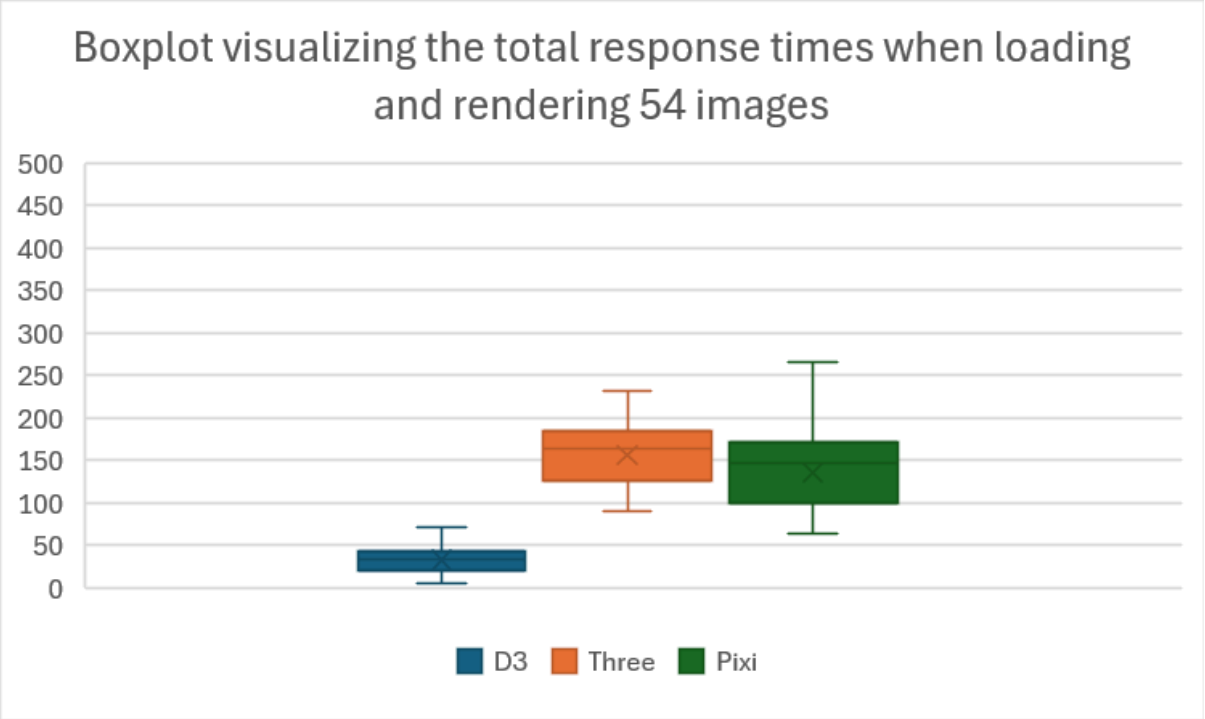

Figure 6. A box plot of the total response times in milliseconds when loading and rendering 54 images for the three different libraries.

Regarding the mean values of the total response times when loading and rendering 81 images, D3.js has a mean value of 28.66 milliseconds, Three.js has a mean value of 146.02 milliseconds, and Pixi.js has a mean value of 124.56 milliseconds. When looking at the standard deviations of the total response times when loading and rendering 81 images, D3.js has a standard deviation of 13.76 milliseconds, Three.js has a standard deviation of 30.44 milliseconds, and Pixi.js has a standard deviation of 35.92 milliseconds.

When looking at the minimum and maximum values of the total response times when loading and rendering 81 images, D3.js has a minimum value of 4.30 milliseconds and a maximum value of 72.50 milliseconds, Three.js has a minimum value of 87.00 milliseconds and a maximum value of 235.50 milliseconds, and Pixi.js has a minimum value of 61.80 milliseconds and a maximum value of 234.60 milliseconds.

See the boxplot below in Figure 7, showing the total response times for all iterations when loading and rendering with 81 images.

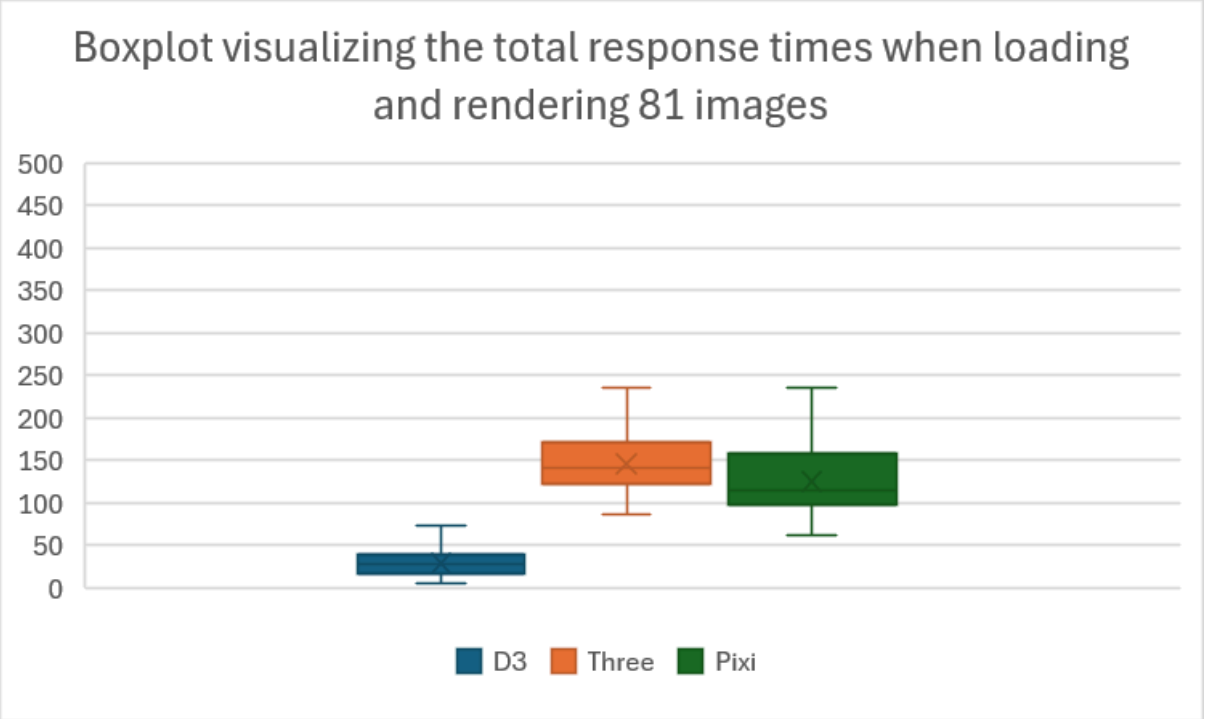

Figure 7. A box plot of the total response times in milliseconds when loading and rendering 81 images for the three different libraries.

When looking at the mean values of the total response times when loading and rendering all 108 images, D3.js has a mean value of 27.48 milliseconds, Three.js has a mean value of 143.25 milliseconds, and Pixi.js has a mean value of 121.47 milliseconds. When looking at the standard deviations of the total response times when loading and rendering all 108 images, D3.js has a standard deviation of 12.88 milliseconds, Three.js has a standard deviation of 27.56 milliseconds, and Pixi.js has a standard deviation of 33.09 milliseconds.

When looking at the min and max values of the total response times when loading and rendering all 108 images, D3.js has a minimum value of 4.30 milliseconds and a maximum value of 115.10 milliseconds, Three.js has a minimum value of 87.20 milliseconds and a maximum value of 232.20 milliseconds, and Pixi.js has a minimum value of 62.20 milliseconds and a maximum value of 436.10 milliseconds. See the boxplot below in Figure 8.

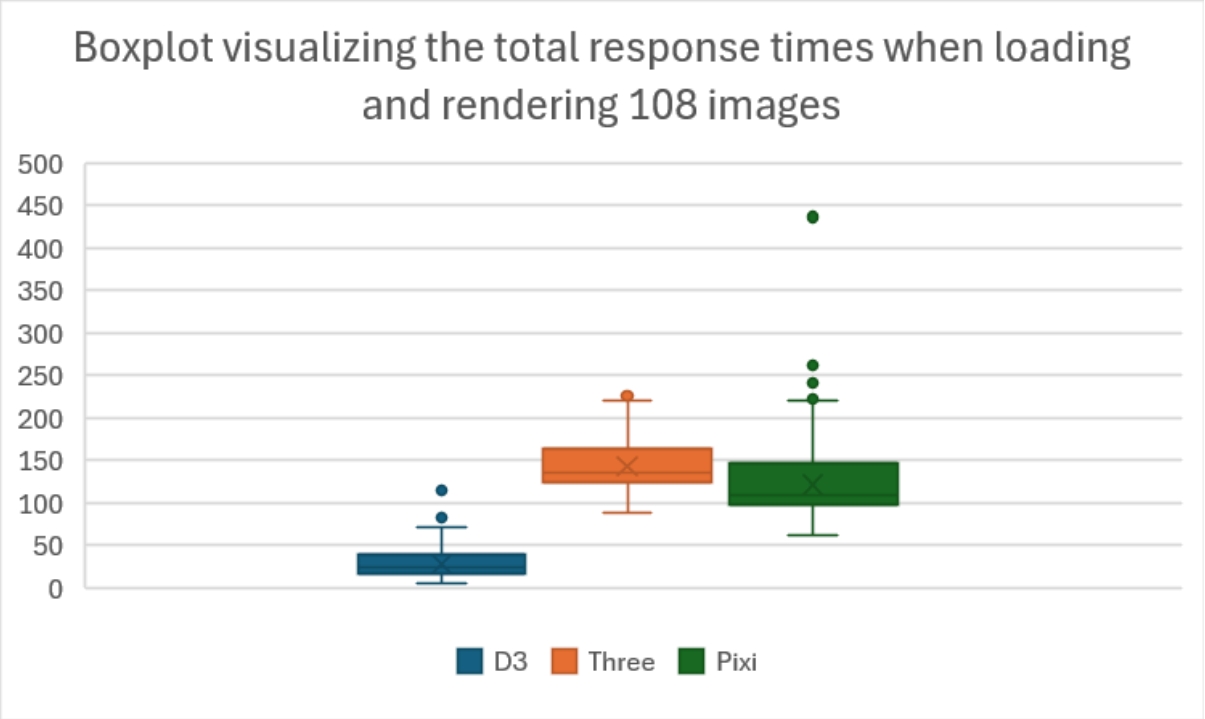

Figure 8. A box plot of the total response times in milliseconds when loading and rendering 108 images for the three different libraries, including outliers.

When comparing the four different numbers of images being loaded and rendered, combining all the libraries, loading and rendering with 27 images had a mean value of 118.24 milliseconds, loading and rendering with 54 images had a mean value of 107.92 milliseconds, loading and rendering with 81 images had a mean value of 99.75 milliseconds, and loading and rendering with 108 images had a mean value of 97.40 milliseconds. Loading and rendering with 27 images had a standard deviation of 64.67, loading and rendering with 54 images had a standard deviation of 61.97, loading and rendering with 81 images had a standard deviation of 58.36, and loading and rendering with 108 images had a standard deviation of 56.54.

When looking at the minimum and maximum values of the total response times when divided into the four different numbers of images, loading and rendering with 27 images had a minimum value 4.60 milliseconds and a maximum value of 335.50, loading and rendering with 54 images had a minimum value of 4.50 milliseconds and a maximum value of 265.70, loading and rendering with 81 images had a minimum value of 4.30 milliseconds and a maximum value of 235.50, and loading and rendering with 108 images had a minimum value of 4.30 milliseconds and a maximum value of 436.10. See the boxplot below in Figure 9.

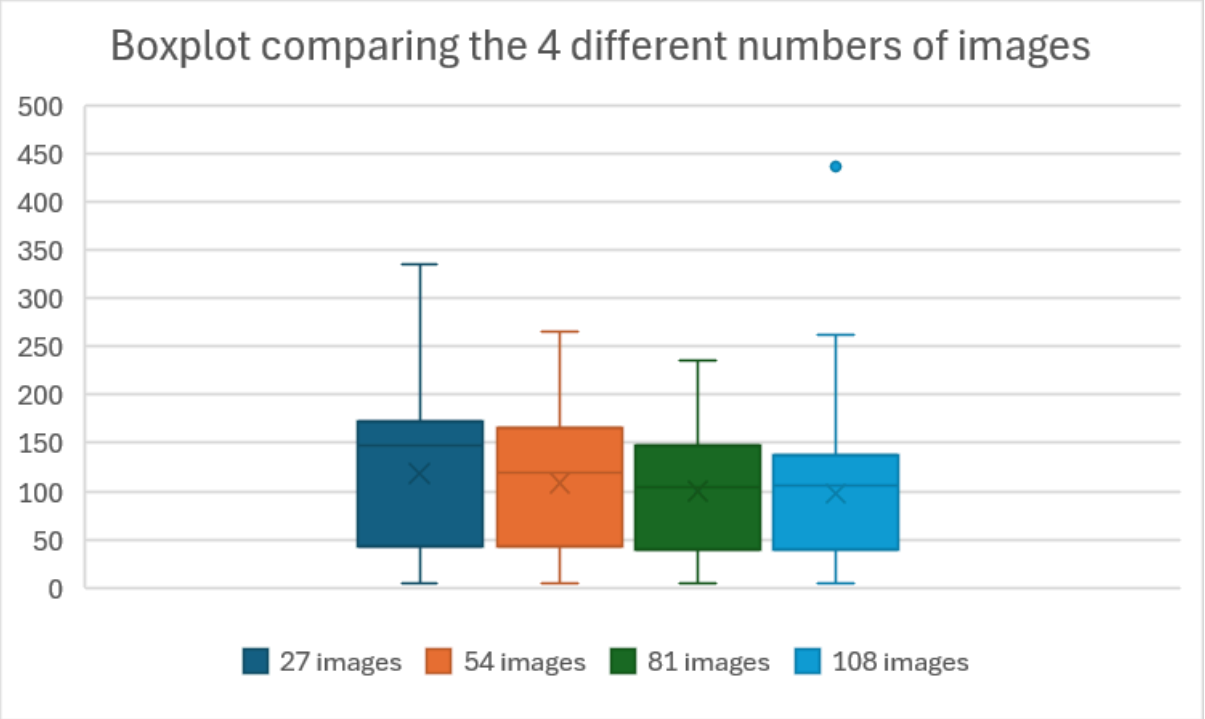

Figure 9. A boxplot comparing the total response times of the four different numbers of images being loaded and rendered while mixing the libraries.

As illustrated in Figure 10, D3.js exhibited a noticeably lower standard deviation than both Three.js and Pixi.js. When visualizing the standard deviation, it becomes apparent that the standard deviation is lower for D3.js than for Three.js and Pixi.js. Below is a graph visualizing the standard deviations. See Figure 10.

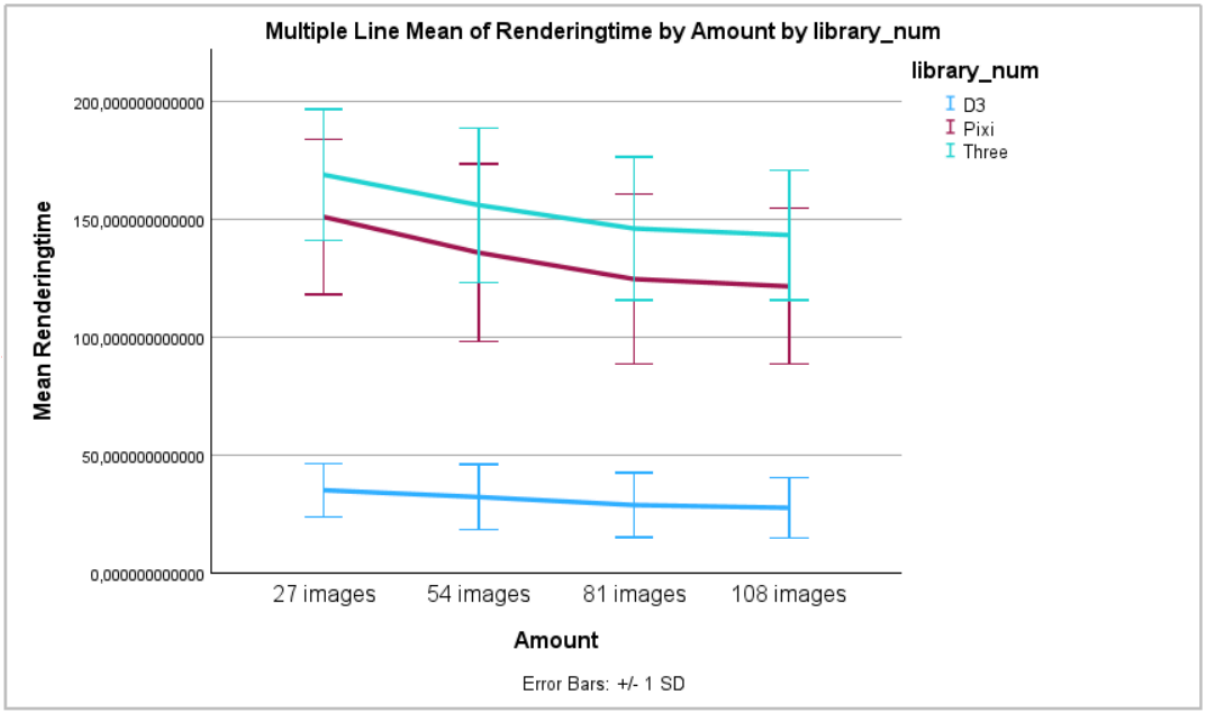

Figure 10. Showing the Standard deviation of the three libraries

For the inferential analysis, a one-way ANOVA and a Welch one-way ANOVA were used, including a Tukey test in order to see how the libraries compare with each other when paired two and two. Lastly, a two-way ANOVA was used in order to see if there are any differences between the numbers of images. First, a test of homogeneity of variance was performed. As shown in Figure 11, the p-values in the significance, or "sig.", column were all less than 0.001.

Tests of Homogeneity of Variances

		Levene Statistic	df1	df2	Sig.
Renderingtime_all	Based on Mean	16007,685	2	80997	<,001
	Based on Median	12953,869	2	80997	<,001
	Based on Median and with adjusted df	12953,869	2	58725,564	<,001
	Based on trimmed mean	15946,447	2	80997	<,001

Figure 11. Showing the "Tests of Homogeneity of Variances" table that was gathered from the one-way ANOVA

Because of that, a Welch one-way ANOVA test, which does not assume equal variances, was done instead in the same way as an ordinary one-way ANOVA, but this time including a Welch test as well. In the "sig." column of the "Robust Tests of Equality of Means" table, the p-value was still under 0.001.

When looking at the "Multiple comparisons" Tukey table from the same test, all the p-values under the "sig." column were also less than 0.001. See Figure 12.

Multiple Comparisons

Dependent Variable: Renderingtime_all

Tukey HSD

(I) Library_all_recoded	(J) Library_all_recoded	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
D3	Three	-119,6805629*	,24447000270	<,001	-120,2535379	-119,1075879
	Pixi	-98,70200367*	,24447000270	<,001	-99,27497871	-98,12902863
Three	D3	119,6805629*	,24447000270	<,001	119,10758785	120,25353794
	Pixi	20,97855923*	,24447000270	<,001	20,405584183	21,551534271
Pixi	D3	98,70200367*	,24447000270	<,001	98,129028627	99,274978715
	Three	-20,97855923*	,24447000270	<,001	-21,55153427	-20,40558418

*. The mean difference is significant at the 0.05 level.

Figure 12. Showing the "Multiple Comparisons" Tukey table, which was gathered from the one-way ANOVA test.

A two-way ANOVA was subsequently conducted to determine whether there was any correlation in the total response times between the number of images and the library. When looking at the table "Tests of Between-Subjects Effects", it becomes apparent that the "sig" values for both the library, the number of images, and the library combined with the number of images were under 0.001. The F-values are also significantly high for the library, the number of images, and the library combined with the number of images. See Figure 13.

Tests of Between-Subjects Effects

Dependent Variable: Renderingtime

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	224767984,1[a]	11	20433453,100	27069,523	<,001
Intercept	696692495,91	1	696692495,91	922953,811	<,001
library_num	179427203,09	2	89713601,545	118849,436	<,001
Amount	3506654,431	3	1168884,810	1548,498	<,001
library_num * Amount	710726,166	6	118454,361	156,924	<,001
Error	61133863,057	80988	754,851		
Total	1133463040,5	81000			
Corrected Total	285901847,16	80999			

a. R Squared = ,786 (Adjusted R Squared = ,786)

Figure 13. Showing the "Tests of Between-Subjects Effects" table

5 Discussion

5.1 Results

When answering the research question “How does the total response time differ between SVG-based D3.js, WebGL-based Three.js, and WebGL-based Pixi.js when loading and rendering PNG images?” it became apparent that there was a statistically significant difference between all three libraries, as well as there being a statistical difference between the four different numbers of images being loaded and rendered, where it became apparent that rendering fewer images resulted in slightly longer total response time. In this study, the observation represents the total time to process all images divided by the number of images. This shows that the average total response time per image decreases as the number of images increases. One possible explanation for this is that the total response times in this study include both a fixed setup delay and the time spent actually rendering the images. This downward trend most likely happens because every run requires a setup delay, and as the image count increases, the average total response time decreases as this constant setup delay gets divided by a higher number. This phenomenon is known as “Amortization of fixed initialization overhead” (Kar et al., 2021; Liu et al., 2023). Another possible explanation for this observation is that modern web browsers start running code at a slow pace but start running faster and optimize the code when they have a larger task to do. This is known as “Browser JIT warm-up” (Barret et al., 2017). Furthermore, the library with the shortest total response time depends on the number of images loaded and rendered.

It also becomes apparent that D3.js had a shorter average total response time than Three.js and Pixi.js when loading and rendering all four numbers of images. Therefore, the null hypothesis was rejected in favour of the alternative hypothesis, which is that there is a statistically significant difference in total response time between SVG-based D3.js and WebGL-based Three.js and Pixi.js.

5.2 Threats to Validity

5.2.1 Internal Validity

Internal validity is the extent to which the thing that is to be measured is actually what is being measured, or if there is anything else that affects the outcome of the results without the awareness of the researcher. In other words, there is something other than the independent variable or variables that affects the dependent variable or variables without the researcher knowing it. This is a threat to internal validity since it might lead to the result being inaccurate because there are more factors affecting the results than the one we want to measure (Wohlin et al., 2012). Wohlin et al. (2012) categorize internal validity threats into three types. These are single group threats, multiple group threats, and social threats (Wohlin et al., 2012).

One example of a threat to internal validity that Wohlin et al. (2012) mention is history, which is when different treatments are applied to an object at different times, which can be a threat considering that the environment might have changed between those two moments. For this project, it could be that some background processes were more active at one moment compared to the other, as it was not possible to close all background processes. This was avoided by doing the loadings and renderings for all three libraries right after each

other. However, this could still be a threat to internal validity since circumstances might still have changed, considering the experiment was completed over the course of five days, meaning that the environment might have changed slightly during the course of the experiment.

Another type of threat to internal validity that (Wohlin et al., 2012) mentions is timing. This means that if the test is iterated multiple times, the subjects could respond differently for different iterations, which becomes an issue since something else affects the dependent variables other than the known independent variables. This was prevented by rerunning the program each time the same map images would be loaded and rendered again, as well as clearing the cache after each run, since the cache stores the map images for future iterations. However, this threat to internal validity can still be an issue since it is possible to forget to reload the page or clear the cache.

The last example of a threat to internal validity that Wohlin et al. (2012) mention is instrumentation. When designing the artifacts for the experiment poorly, it might lead to threats to internal validity. This is an issue because it could lead to certain libraries being rendered in different ways. In this experiment, a threat to internal validity regarding instrumentation would be that the three different libraries were rendered in different ways due to differences in the code, which is expected since they are different libraries with inherent differences. More specifically, D3.js adds image blocks directly to the DOM tree of the webpage (Bostock, Ogievetsky & Heer, 2011), while Three.js and Pixi.js require code that builds things like camera matrices and virtual scenes (Mwalongo et al., 2016). However, the three libraries were still rendered as similarly as possible by minimizing differences in the code. The timers were also placed similarly. However, there is still a risk that small details in the code lead to this type of threat to internal validity.

5.2.2 External Validity

External validity is the extent to which the results are applicable to the outside world or only to the context of the experiment. If the experimental subjects or conditions are rarely utilized in real-world scenarios, it poses a threat to external validity, making the results impractical (Wohlin et al., 2012). Threats to external validity can be divided into three different types of interactions with the treatment. These three different types are interaction of selection and treatment, interaction of setting and treatment, and interaction of history and treatment (Wohlin et al., 2012).

Interaction of selection and treatment, also called mono-operation bias, concerns whether the specific subjects that are tested are representative of the outside world, or if instead only one certain type of subject has been chosen, which would mean that the experiment would not be representative (Wohlin et al., 2012). In this project, one restriction that might lead to a threat to external validity by interaction of selection and treatment is that the use of only raster-based PNG images will lead to the results not applying to people using vector data. PNG images are widely used on the internet today, but there is still a valid threat to validity, as some people may use vector data.

Interaction of setting and treatment, which is another threat to external validity, concerns whether the setting of the experiment, such as the use of certain tools or hardware, can be generalized to the outside world and is representative within the industry (Wohlin et al., 2012). Threats to external validity concerning the interaction of setting and treatment could

therefore include using old-fashioned tools or studying toy examples. In this project, the experiment was done on a 7-year-old PC with Windows 10. Since it is not very new hardware and operating system, it can most certainly be a threat to external validity. However, it may still reflect the condition for many users today.

Interaction of history and treatment concerns itself with whether the specific time that the experiment was conducted will affect the results (Wohlin et al., 2012). In the context of this experiment, if it had been made at the same time as an update in the system was being made, this would not be representative of how it normally works. This was avoided by making sure that no updates were made during the course of the experiment. However, there could still be factors threatening the external validity regarding the interaction of history and treatment, such as updates running unknown in the background.

5.2.3 Conclusion Validity

Conclusion validity is the extent to which the conclusion about the relation between the treatment and the outcome is correct. If the outcome is unrelated to the treatment, there is a threat to conclusion validity. There are several types of threats to conclusion validity.

One threat to conclusion validity is low statistical power. It concerns whether the test actually detects a pattern in the data. In this experiment, one example of a threat to conclusion validity regarding low statistical power is that the sample size would be too small or, in other words, the number of images that were included and the number of iterations that they were loaded and rendered should have been larger. This threat was mitigated by loading and rendering the images as many as 27000 times for each library. However, since all the libraries turned out to have a statistically significant difference in total response times, this threat to conclusion validity seems irrelevant in this experiment, as the number of images that were loaded and rendered was enough to suggest a statistical difference in total response times between the three different libraries.

Another threat to conclusion validity is the reliability of measures, which concerns whether identical measurements yield consistent results. In this project, it involves whether the tool measuring the time is accurate enough or not. In this experiment, the same images had different total response times during different iterations, which could be a threat to conclusion validity regarding the reliability of measures. However, it could also have to do with other factors, such as the different iterations actually taking different amounts of time during different iterations. This was most likely the case since the tool that was used to measure, “performance.now()”, is widely used and considered a reliable function offered by JavaScript. If the total response times had been measured manually using, for example, a timer, it would have been a serious threat to conclusion validity regarding the reliability of measures, as humans tend to not be able to capture the precise time, especially considering the times are extremely short, below one second.

A third threat to conclusion validity is random irrelevancies in experimental setting. This threat to conclusion validity concerns whether there are any elements outside of the experiment disturbing the experiment. In the context of this experiment, there may have been certain background processes running in the background. This was mitigated to the best of the researcher’s ability. However, some background processes were not able to be closed down. There were also some background processes that were part of the operating

system that may still disturb the experiment, but in this case they should have distributed all libraries equally.

5.2.4 Construct Validity

Construct validity is the extent to which the results of the experiment can be generalized to the theory behind the experiment. It concerns the design of the experiment and how well it reflects the general concept that it studies. There are multiple different types of threats to construct validity.

One threat to construct validity is mono-operation bias. It concerns whether the thing that is measured represents the entire theory of the concept that is being studied or only a small part or variable. In the context of this experiment, the general concept that is studied is the total user experience, and therefore both the loading and rendering times were included. Had only the rendering time been measured while the total user experience is still the thing that is studied, it would have been a threat to construct validity regarding mono-operation bias, as the rendering time is only a part of the total time that the user has to wait. However, there are still other parts of the overall user experience beyond just the total response time that are not captured in this research question, such as the user experience if the program easily crashes. This could potentially be a threat to construct validity regarding mono-operation bias.

Another threat to construct validity is experimenter expectancies. This concerns whether the expectations of the person carrying out the experiment may affect the results. In the context of this experiment, it concerns whether the code that carries out the experiment is written similarly for all three libraries. As discussed in 5.2.1, the code was written slightly differently when loading and rendering the three different libraries since they are fundamentally different libraries and therefore load and render in different ways. Even though it was avoided as much as possible, it is still a threat to construct validity regarding experimenter expectancies, as the experimenter has purposely written the code differently.

5.3 Ethical and Societal Considerations

5.3.1 Research Ethics

When researching and performing an experiment, it is essential to think about certain ethical aspects. Within the field of information technology, there are certain ethical aspects that need to be considered. A lot of these ethical aspects involve human participants, which, according to Wohlin et al. (2012), includes guidelines such as subjects needing to give consent and have access to all relevant information about the study, as well as maintaining confidentiality of data and sensitive information. However, a study that does not involve human participants, such as this one, has to focus on other ethical aspects. One ethical aspect is copyright and the use of public data. This includes only using data sets and tools that are open to the public and respecting copyright laws (Longpre et al., 2024; Pater, Fiesler & Zimmer, 2022). Another ethical aspect, including no human participants, is reproducibility. This means that the way the experiment was conducted is documented to the extent that it can be replicated by another researcher (Pineau et al., 2020).

In this study, no human participants or personal data were used, so an ethics board review was not needed. Instead, the challenge was to make sure that no copyrighted material was

used. This was fixed by ensuring that all test data, code, libraries, and images used for the experiment were used with integrity and were free to use as long as there was no monetization involved. All open-source libraries as well as the website used for downloading the dataset were properly credited and cited. Regarding reproducibility, Pinaeau et al. (2020) outline several systematic reasons why digital experiments often fail to be replicated, such as an incorrect application of statistics or selectively reporting favorable results. This framework was taken into consideration during this study by maintaining transparency in the data analysis and documenting every type of software used, including the versions of the three libraries.

5.3.2 Effects on Society

Regarding effects on society, they can be divided into three different aspects. The first one is the environmental aspect. It concerns itself about how the research affects the environment and the carbon footprint. The results of this study have shown that the CPU-driven SVG-based library D3.js is faster than the GPU-driven WebGL-based libraries Three.js and Pixi.js when rendering PNG images. One possible implication is that these results can help developers build applications that are based on libraries that consume less energy, lowering the overall footprint of digital infrastructure (Moura et al., 2025).

The second sustainability aspect is the economic aspect. This aspect concerns itself with how the economy will be affected. From an economic perspective, reducing the total response times keeps users happy, and in turn it doesn't drive the users away, increasing the revenue for the platform (Arapakis et al., 2021). The findings of this study can possibly help developers choose the appropriate library with the lowest total response time, which in this case was D3.js, which can actively prevent user abandonment and, in turn, protect business outcomes.

The third and last aspect of sustainability is the social aspect. This aspect deals with how the results of the study affect people. The results of this study can possibly lead to digital accessibility, allowing people with cheap old smartphones or slow mobile connections to access images online by being informed that D3.js has a shorter total response time. Therefore, the findings of this study can help developers build websites that are more inclusive for people who can not afford expensive technology or high-speed internet.

These theories regarding the three different sustainability aspects are all possible implications that are not yet proven. However, those theories could be explored in the future.

5.3.3 Authorship

The author of this research paper is responsible for the content and guarantees that everything written was written by the author and that there were no intentional mistakes made. The author also guarantees that no collected metric data or test loops have been altered, hidden, or cherry-picked to skew the results.

5.4 Generalization

In related work, it is clear that the results from the study by Koren Ivančević et al. (2026) are similar to the ones in this experiment since the map images used in this experiment are not very complex, something that Koren Ivančević et al. (2026) state is an advantage of the SVG-based library of D3.js. This correlates with the results of this project, where D3.js

outperformed Three.js and Pixi.js. If this work and the work of Koren Ivančević et al. (2026) were to be put together, it would address the points of Koren Ivančević et al. (2026) while also gaining knowledge about the total response times of the three specific libraries, something that was a research gap.

Regarding the time-measuring and webbrowser, Lukács & Gartus (2023) state that "performance.now()" measures the time elapsed since the page was loaded. The authors also state that the browser and operating system being used might affect how "performance.now()" is used. They claim that if "performance.now()" is used on Google Chrome on Windows, the RAF timestamp indicates the time right before the RAF command executes, while on Safari on macOS the RAF timestamp indicates the time right after the RAF command executes. This difference means that if Safari and macOS were used instead, the total recorded response times would appear shorter due to the command execution duration being excluded. This was not a problem as Google Chrome and Windows were used in the experiment.

When generalizing the results gathered in this experiment with Arapakis et al. (2021), it becomes apparent that the total response times of all three libraries are far below the time it takes for a user to feel frustrated and get a negative user experience. This becomes apparent when looking at the mean values of the total response times for the three different libraries, as the mean value for D3.js is 29.5 milliseconds, which is 0.0295 seconds. The mean value for Three.js is 149.18 milliseconds, which is 0.14918 seconds, and the mean value of Pixi.js is 128.2 milliseconds, which is equal to 0.1282 seconds. Comparing this to the time it takes for a user to get a negative user experience, which is 7 to 10 seconds, it becomes apparent that none of the libraries lead to a negative user experience.

When generalizing the results of this study alongside the study by Moura et al. (2025), the authors show that different code structures change the latency. Given the fact that D3.js uses the CPU and the DOM tree, while Three.js and Pixi.js use the GPU, the results of this study show that D3.js is more beneficial for users using, for example, a cheap mobile phone.

5.5 Future Work

If the project were to span over a significantly longer time, future work would include testing on more libraries. This could potentially include a canvas 2D API framework like Fabric.js. This would determine whether the browser's native engine handles flat images better than WebGL (Sengupta et al., 2025).

Future work might also include having a larger dataset including other image file formats such as JPG. Hence, it would be possible to compare the total response times when the libraries load and render PNG images compared with JPG images. Another way that this project could be expanded is by iterating the images more times, which would help the results to become more trustworthy and help detect outliers that do not correlate with the rest of the results.

If the project had been expanded by measuring more than just total response time, like measuring memory or bandwidth, it would expose more aspects of loading and rendering the map images other than only the total response time, which is relevant since tracking RAM allocation shows how the libraries affect low-end hardware like mobile phones where the mobile web browser could crash (Qazi et al., 2020).

6 Conclusion

This study has provided a comparison between the total response times of the three different JavaScript libraries D3.js, Three.js, and Pixi.js when loading and rendering PNG images. The findings demonstrate a statistically significant difference in total response times among the JavaScript libraries D3.js, Three.js, and Pixi.js when loading and rendering PNG images representing survey maps with all four numbers of images.

Furthermore, the results indicate that the number of images being loaded and rendered affects the total response time and that which library has the shortest total response time is also dependent on the number of images being loaded and rendered when loading and rendering PNG images.

Ultimately, D3.js, which uses SVG, outperformed both WebGL libraries Three.js and Pixi.js when loading and rendering the PNG images with all four numbers of images.

A primary limitation of this study is that the experiment was restricted to only PNG images and not other image formats. There could also have been external factors that might have changed the outcome of the experiment. For future research, it would therefore be appropriate to include image formats other than PNG.

References

Afonso, M. Q., Fonseca, N., Miranda, T. G. & Bleicher, L. (2022). Naview: A d3.js Based JavaScript Library for Drawing and Annotating Voltage-Gated Sodium Channels Membrane Diagrams. *Frontiers in Bioinformatics, 2,* 1-8.
https://doi.org/10.3389/fbinf.2022.774417

Andreasen, E., Gong, L., Møller, A., Pradel, M., Selakovic, M., Sen, K. & Staicu, C. (2017). A Survey of Dynamic Analysis and Test Generation for JavaScript. *ACM Computing Surveys, 50(5),* 1-36.
https://doi.org/10.1145/3106739

Arapakis, I., Park, S. & Pielot, M. (2021). Impact of Response Latency on User Behaviour in Mobile Web Search. *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval,* 279-283.
https://doi.org/10.1145/3406522.3446038

Barret, E., Bolz-Tereick, C. F., Killick, R., Mount, S. & Tratt, L. (2017). Virtual machine warmup blows hot and cold. *Proceedings of the ACM on Programming Languages, 1(52),* 1-27.
https://doi.org/10.1145/3133876

Bostock, M. and Observable. (2024). *D3.js* (Version 7.9.0).
Available on the Internet:
https://d3js.org/d3.v7.min.js
[Accessed 26.04.16]

Bostock, M., Ogievetsky, V. & Heer, J. (2011). D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics, 17(12),* 2301-2309.
10.1109/TVCG.2011.185

Cabello, R. (2026). *Three.js* (Version 0.183.2).
Available on the Internet:
https://cdn.jsdelivr.net/npm/three@0.183.2/build/three.module.js
[Accessed 26.04.16]

Dornauer, B. & Felderer, M. (2023). Web Image Formats: Assessment of Their Real-World-Usage and Performance across Popular Web Browsers. *Product-Focused Software Process Improvement 24th International Conference,* 1-16.
https://doi.org/10.48550/arXiv.2310.00788

Garret, T. (2025) *SVG vs. Canvas vs. WebGL: Rendering Choice for Data Visualization.*
Available on the internet:
https://dev3lop.com/svg-vs-canvas-vs-webgl-rendering-choice-for-data-visualization/
[Accessed 26.03.04].

Groves, M., Karl, M., Burns, S. & Soldatov, D. (2024). *Pixi.js* (Version 7.4.2).
Available on the internet:
https://pixijs.download/v7.4.2/pixi.js
[Accessed 26.04.16]

Guilbeault, D., Delecourt, S., Hull, T., Desikan, B. S., Chu, M. & Nadler, E. (2024). Online images amplify gender bias. *Nature, 626,* 1049-1055. https://doi.org/10.1038/s41586-024-07068-x

Han, Y., Bi, W., An, R., Tian, D., Yang, Q. & Ma, Y. (2025). Accelerating WebGL Applications via Dynamic API Translation to WebGPU. *Proceedings of the ACM on Web Conference 2025,* 751-762. https://doi.org/10.1145/3696410.3714785

Kar, S., Rehrmann, R., Mukhopadhyay, A., Alt, B., Ciucu, F., Koeppl, H., Binnig, C. & Rizk, A. (2021). On the Throughput Optimization in Large-Scale Batch-Processing Systems. *ACM SIGMETRICS Performance Evaluation Review, 48(3),* 128-129. https://doi.org/10.1145/3453953.3453982

Koren Ivančević, T., Ježić, T.J., Stanić Loknar, N. (2026). A Cross-Device and Cross-OS Benchmark of Modern Web Animation Systems. *Journal of Imaging, 12(1),* 1-19. https://doi.org/10.3390/jimaging12010045

Lantmäteriet (n.d.) *Kartutskrift* Available on the Internet: https://kartutskrift.lantmateriet.se/ [Accessed 26.04.02]

Liu, X., Wen, J., Chen, Z., Li, D., Liu, Y., Wang, H. & Jin, X. (2023). FaaSLight: General Application-level Cold-start Latency Optimization for Function-as-a-Service in Serverless Computing. *ACM Transactions on Software Engineering and Methodology, 32(5),* 1-29. https://doi.org/10.1145/3585007

Liu, X., Wu, T., Xu, W. & Liu, Q. (2024). Design and Implementation of WebGL Model Rendering Engine. *Scientific Journal of Intelligent Systems Research, 6(7),* 19-23. https://doi.org/10.54691/778tvh90

Longpre, S., Mahari, R., Chen, A., Obeng-Marnu, N., Sileo, D., Brannon, W., Muennighoff, N., Khazam, N., Kabbara, J., Perisetla, K., Wu, X., Shippole, E., Bollacker, K., Wu, T., Villa, L., Pentland, S. & Hooker, S. (2024). A large-scale audit of dataset licensing and attribution in AI. *Nature Machine Intelligence, 6,* 975-987. https://doi.org/10.1038/s42256-024-00878-8

Lukács, G. & Gartus, A. (2023). Precise display time measurement in JavaScript for web-based experiments. *Behavior Research Methods, 55,* 1079-1093. https://doi.org/10.3758/s13428-022-01835-2

Moura, M., Andrade, F., Lima, L. & Djenouri, Y. (2025). A Comparative Analysis of Web Frameworks for Online Image Visualization With Robot Operating System 2. *25th International Conference on Control, Automation and Systems (ICCAS),* 1858-1862. https://doi.org/10.23919/iccas66577.2025.11301174

Mwalongo, F., Krone, M., Reina, G. & Ertl, T. (2016). State-of-the-Art Report in Web-based Visualization. *Computer Graphics Forum, 35(3),* 553-575. https://doi.org/10.1111/cgf.12929

Myrén, M. (2025). Rendering av bilder för en SPA-artefakt hos olika JavaScript-ramverk: Jämförelse av laddningstider hos JavaScript-ramverk för rendering av JPEG, PNG och WEBP-filer i en SPA. *Unpublished bachelor's thesis. University of Skövde.* https://his.diva-portal.org/smash/record.jsf?pid=diva2%3A1973350&dswid=4301

Netek, R., Masopust, J., Pavlicek, F., Pechanec, V. (2020). Performance Testing on Vector vs. Raster Map Tiles—Comparative Study on Load Metrics. *International Journal of Geo-Information, 9(2),* 1-24. https://doi.org/10.3390/ijgi9020101

Pater, J. A., Fiesler, C. & Zimmer, M. (2022). No Humans Here: Ethical Speculation on Public Data, Unintended Consequences, and the Limits of Institutional Review. *Proceedings of the ACM on Human-Computer Interaction, 6(38),* 1-13. https://doi.org/10.1145/3492857

Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché-Buc, F., Fox, E. & Larochelle, H. (2020). Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program). *ArXiv.* https://doi.org/10.48550/arXiv.2003.12206

Qazi, I. A., Qazi, Z. A., Benson, T. A., & Murtaza, G. (2020). Mobile web browsing under memory pressure. *ACM SIGCOMM Computer Communication Review, 50(4),* 35-48. https://doi.org/10.1145/3431832.3431837

Quint, A. (2003) Scalable vector graphics. *IEEE MultiMedia, 10(3),* 99-102. 10.1109/MMUL.2003.1218261

Regoršek, Ž., Gorkič, A. & Trost, A. (2024). Parallel Lossless Compression of Raw Bayer Images on FPGA-Based High-Speed Camera. *Sensors, 24(20),* 1-19. https://doi.org/10.3390/s24206632

Schwab, M., Saffo, D., Bond, N., Sinha, S., Dunne, C., Huang, J., Tompkin, J. & Borkin, M. (2021) Scalable Scalable Vector Graphics: Automatic Translation of Interactive SVGs to a Multithread VDOM for Fast Rendering. *IEEE Transactions on Visualization and Computer Graphics, 28(9),* 3219-3234. 10.1109/TVCG.2021.3059294

Sengupta, S., Wu, N., Varvello, M., Jana, K., Chen, S. & Han, B. (2025). From WebGL to WebGPU: A reality check of browser-based GPU acceleration. *Proceedings of the 2025 ACM Internet Measurement Conference,* 1018-1024. https://doi.org/10.1145/3730567.3764504

Sharov, S., Chakraborty, S., Zemlianskyi, A., Kolmakova, V. & Prosiankina-Zharova, T. (2024) Quantitative Analysis of Online Courses for Learning the JavaScript Programming Language. *International Journal on Advanced Science, Engineering and Information Technology, 14(4),* 1216-1225. https://doi.org/10.18517/ijaseit.14.4.19976

Soylu, M. & Das, R. (2025) Prediction and graph visualization of cyber attacks using graph attention networks. *Computers & Security, 157,* 1-13.

Tapflare (2025) *Comparing Web Graphics: Canvas, SVG, WebGL, and CSS*
Available on the internet:
https://tapflwdaare.com/articles/web-graphics-comparison-canvas-svg-webgl
[Accessed 26.03.04]

Wagner, G. (2016) Introduction to simulation using JavaScript. *2016 Winter Simulation Conference (WSC),* 148-162.
10.1109/WSC.2016.7822086

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. *Springer*.
https://doi.org/10.1007/978-3-642-29044-2