Politechnika Opolska

Faculty of Electrical Engineering, Automatic Control and Informatics

Department of Computer Science

MASTER OF SCIENCE
DIPLOMA THESIS

Second-cycle studies

Full-time studies

Informatics

TOPIC OF THE THESIS

Analysis and measurement of CSS animation performance for selected web browsers

Supervisor:

dr hab. inż. Michał Podpora, prof. uczelni

Thesis author:

Bilal Hassona

Student ID: 97630

Opole, September 2023

CSS animation performance in web browsers

The thesis contains a detailed comparison of the performance of various modern GUI-based (Graphical User Interface) web browsers as pertains to CSS-based animations. Moreover, the thesis includes a graphical demonstration of the measurements along with a discussion and analysis of the results. Additionally, a theoretical introduction to the topic and a detailed description of the methodology of animation measuring were included.

Table of contents

1. Purpose and scope of the thesis **5**
2. Introduction **6**
3. Theory **7**
3.1 Browser components 7
3.2 Rendering engines 9
3.3 Global desktop browser market share 12
4. Methodology **13**
4.1 Tested browsers 13
4.1.1 Brave 13
4.1.2 Chromium 14
4.1.3 Google Chrome 14
4.1.4 Konqueror 15
4.1.5 Microsoft Edge 16
4.1.6 Mozilla Firefox 16
4.1.7 Opera 17
4.1.8 Vivaldi 17
4.1.9 Yandex Browser 18
4.2 Browsers that were omitted 19
4.3 Technical setup 20
4.3.1 Hardware and the operating system 20
4.3.2 Supporting software 20
4.4 Animation’s performance measuring 24
4.5 Other unreliable methods of measuring the performance of animations 25
4.6 Problems with the chosen method of measuring performance 26
5. Animations **27**
5.1 General setup 27
5.2 Animation 1 33
5.3 Animation 2 34
5.4 Animation 3 35
5.5 Animation 4 36
5.6 Animation 5 37
5.7 Animation 6 39
5.8 Animation 7 40
5.9 Animation 8 41
5.10 Animation 9 42
5.11 Animation 10 44
6. Results **48**
6.1 Animation 1 48
6.2 Animation 2 49

6.3 Animation 3 50
6.4 Animation 4 51
6.5 Animation 5 52
6.6 Animation 6 53
6.7 Animation 7 54
6.8 Animation 8 55
6.9 Animation 9 56
6.10 Animation 10 58
7. Discussion **59**
8. Summary **62**
8.1 Further research possibilities 62
9. Table of figures **63**
10. Table of listings **65**
11. List of tables **66**
12. References **67**

1. Purpose and scope of the thesis

The purpose of the thesis is to describe the process of measuring the performance of Cascading Style Sheets (CSS) [1-4] animations [5,6] for various web browsers [7] and to present the results in a graphically friendly and readable manner. The outcome of the measurements is then analysed and examined in order to draw constructive conclusions that will contribute to the improvement of the understanding of CSS animations and their adequate usage. Moreover, it will facilitate the optimisation of websites for front-end developers which in turn will improve the end user's experience.

Due to the differences in rendering engines and the way they handle animations, CSS animations may perform differently across various browsers. This can lead to inconsistencies in User Experience (UX) [8], where animations may appear smooth and fluid in one browser but choppy or even broken in another. If animations are too slow, they can make a website appear laggy or unresponsive, which may frustrate users and lead to a negative perception of the site or brand thereby leading to significant financial losses. In order to preserve the website's desired functionality and design it is essential to correctly optimise the CSS animations through making informed decisions about which CSS properties to animate. Some properties, such as transform and opacity, are typically more performant and lead to smoother animations than others. By focusing on these properties, developers can create animations that are not only visually appealing but also perform well across a wide range of browsers.

2. Introduction

The purpose of Cascading Style Sheets (CSS) animations in websites extends beyond simple aesthetic appeal. They serve a critical function in enhancing user engagement, improving navigation, and reinforcing brand identity. CSS animations cater to the human predisposition for visual storytelling, enabling developers to create dynamic, interactive websites that effectively communicate with users.

CSS animations significantly improve user experience (UX) by helping to orient the user. Animated transitions can visually communicate a change in context or draw attention to important actions or elements, thereby reducing cognitive load. When a site uses animation to indicate the result of an action, it helps users understand how to interact with it. For example, a menu might slide out from the side when a user clicks on a hamburger icon, visually showing that they have triggered a new section of the site to appear.

For these reasons, it is vital to provide a smooth experience for the users – the animations should not be lagging, undesirably slowing down or impeding the user's processor.

3. Theory

The web browser emerges as a crucial thread in the complex web of digital technology, interacting directly with end users and playing a crucial role in the representation of online-based information. This section attempts to dissect the operation, structure, and components of web browsers, focusing specifically on the rendering engine [9] and the role it plays in delivering content on a website. Additionally, the section extends its inquiry to examine the current landscape of browser market share [10].

3.1 Browser components

Web browsers, while serving as daily tools for internet users, operate through intricate machinery that remains concealed under an interface of simplicity. This subsection will include a brief examination of the user interface (UI) [11] and its backend [12], the browser and rendering engines [13-15], the JavaScript (JS) engine [16], networking components [16], and data persistence tools [16] (Fig. 3.1). Each segment of this study will unpack the roles these components play and how they collectively contribute to the user's browsing experience.

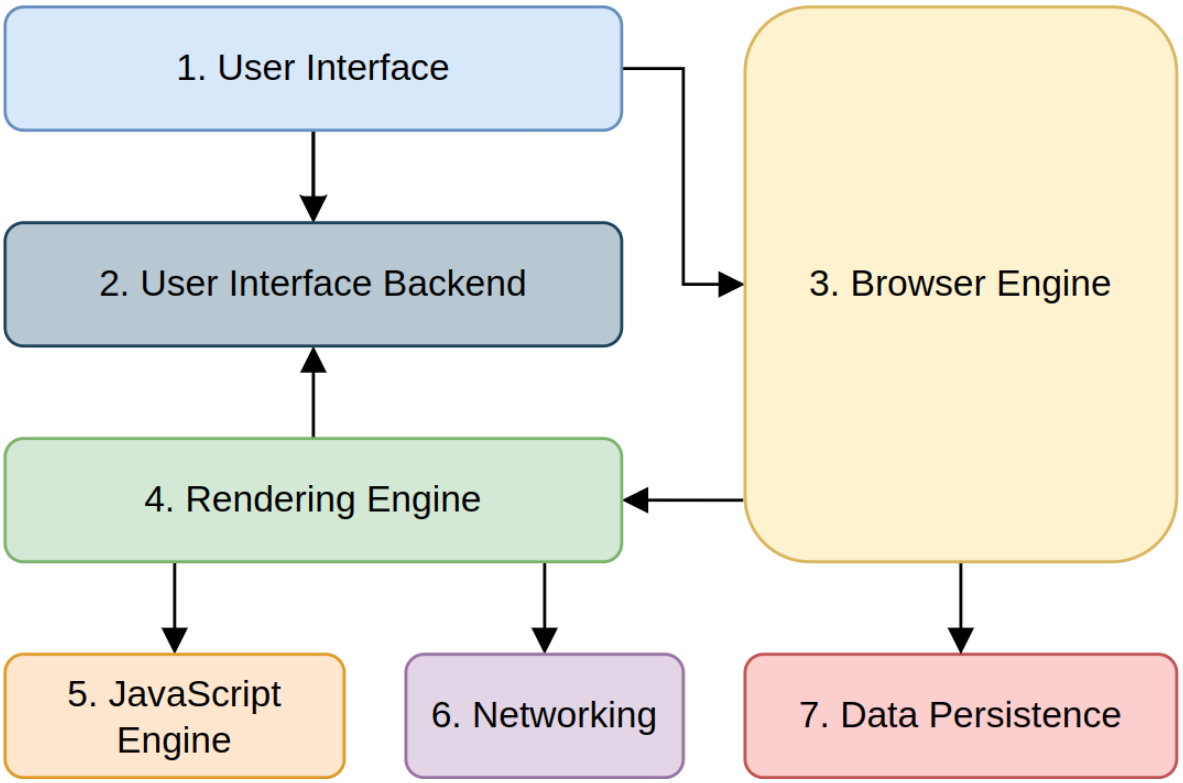

Figure 3.1 Browser components [12,16]

1. The User Interface serves as the tangible bridge between the user and the browser's complex internal mechanisms. It consists of all the elements that users interact with directly, providing a user-centric view and control over the

functionalities of the browser. Key elements of the UI typically include the address bar for URL input, navigation buttons such as back and forward, bookmarks, reload page button and others.

2. The User Interface Backend refers to the underlying system that translates generic graphical control elements into the specific visual elements of the operating system's graphical user interface (GUI) [17]. This abstraction layer allows developers to design and implement user interface elements such as buttons, checkboxes or list menus without concerning themselves with the peculiarities of different operating systems.

3. The Browser Engine is the intermediary between the UI and the rendering engine. It receives commands from the UI, processes these requests, and instructs the rendering engine accordingly. The browser engine is responsible for managing the high-level architecture of the browser, handling interactions and coordination between the user interface, rendering engine, and other browser components. For instance, when a user requests a web page, the browser engine is the component that interfaces with the network, retrieves the requested document and triggers the rendering engine to display the page.

   Different web browsers employ different browser engines, contributing to variances in how they interpret and display web content. Examples include the Gecko [18] engine for Mozilla Firefox [19], the Blink [20] engine for Google Chrome [21], and the WebKit [22] engine for Safari [23].

4. The rendering engine holds a central role in translating web content into the visual output that users interact with. Also known as the layout engine, the rendering engine interprets mainly HTML (HyperText Markup Language) [24-26] and CSS files and visually represents them on the screen (canvas) as a formatted web page. It is tightly coupled with the browser engine and they are often treated as one. More about the rendering engine is written in Subsection 3.2 as it is the most relevant part of a browser to this thesis.

5. Within the architectural framework of web browsers, the JavaScript engine plays an integral role as it is tasked with interpreting and executing JavaScript code embedded within web pages, which is fundamental to creating interactive and dynamic web content. The JavaScript engine first parses JavaScript code and then generates a more computer-friendly format, known as bytecode. The bytecode is then compiled into machine code using just-in-time (JIT) compilation [27]. The JavaScript engine carries out this process each time a web page is loaded, providing the functionality necessary for interactive elements.

   Different web browsers utilise different JavaScript engines, each with unique features and performance characteristics. For instance, Blink-based browsers (such as Google Chrome) use the V8 engine [28], Mozilla Firefox uses SpiderMonkey [29], and Safari uses JavaScriptCore [30]. These three JavaScript engines are the most notable, but there are other engines. The performance of a JavaScript engine significantly influences the speed and efficiency of web browsing, making it a critical component in the overall browser architecture.

6. The networking component is primarily tasked with network calls, such as HTTP requests [31], using different implementations for different operating

systems. It is responsible for fetching resources (like HTML documents, images, and scripts) over the internet, allowing the browser to display web content requested by the user. The networking component handles various types of network protocols and security features, such as the aforementioned HTTP, HTTPS [32], FTP [33] and others. It communicates with servers to request the necessary files and resources needed to render a web page, manage cookies [34], and deal with various other aspects of Internet communication.

7. The data persistence component manages the storage and retrieval of data that needs to persist across browsing sessions, contributing significantly to the user's browsing experience by enabling form auto-fill or saving various user preferences and settings. The data persistence component manages several types of storage, including cookies, local storage [35] and IndexedDB [36]. These storage mechanisms allow websites to store data on the user's device, which can be retrieved even after the browser is closed and reopened.

3.2 Rendering engines

The rendering engines are a complicated and highly-optimised component of web browsers. They are responsible for multiple calculations relating to a swift display of HTML and CSS as well as tracking changes in those. They optionally support other files such as PDF [37] or XML [38] files. Below are depictions of the main flow of Mozilla Firefox's Gecko and Safari's WebKit rendering engines. Google Chrome's Blink is a fork of WebKit and the two discussed (Gecko and WebKit) are sufficient to understand how the rendering engines work in general (Fig. 3.2-3).

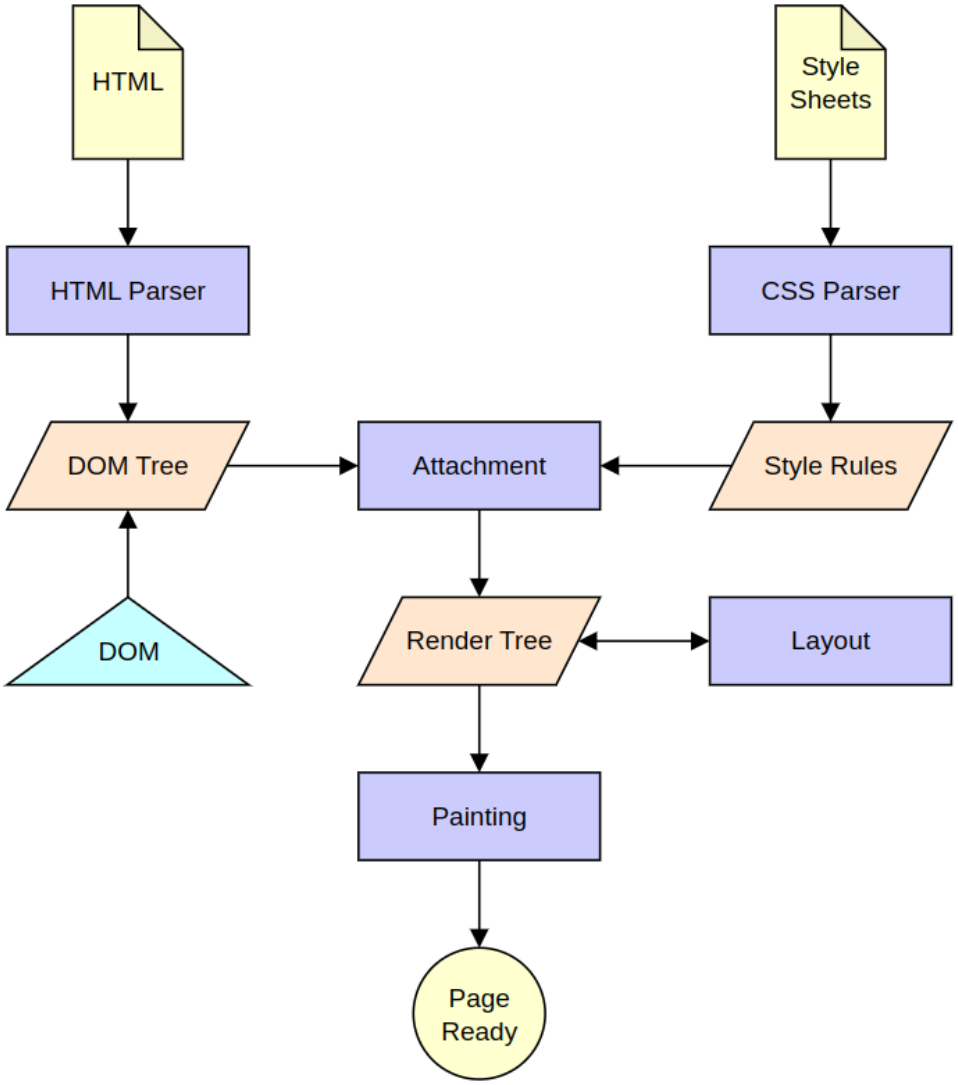

Figure 3.2 WebKit engine's general flow [39]

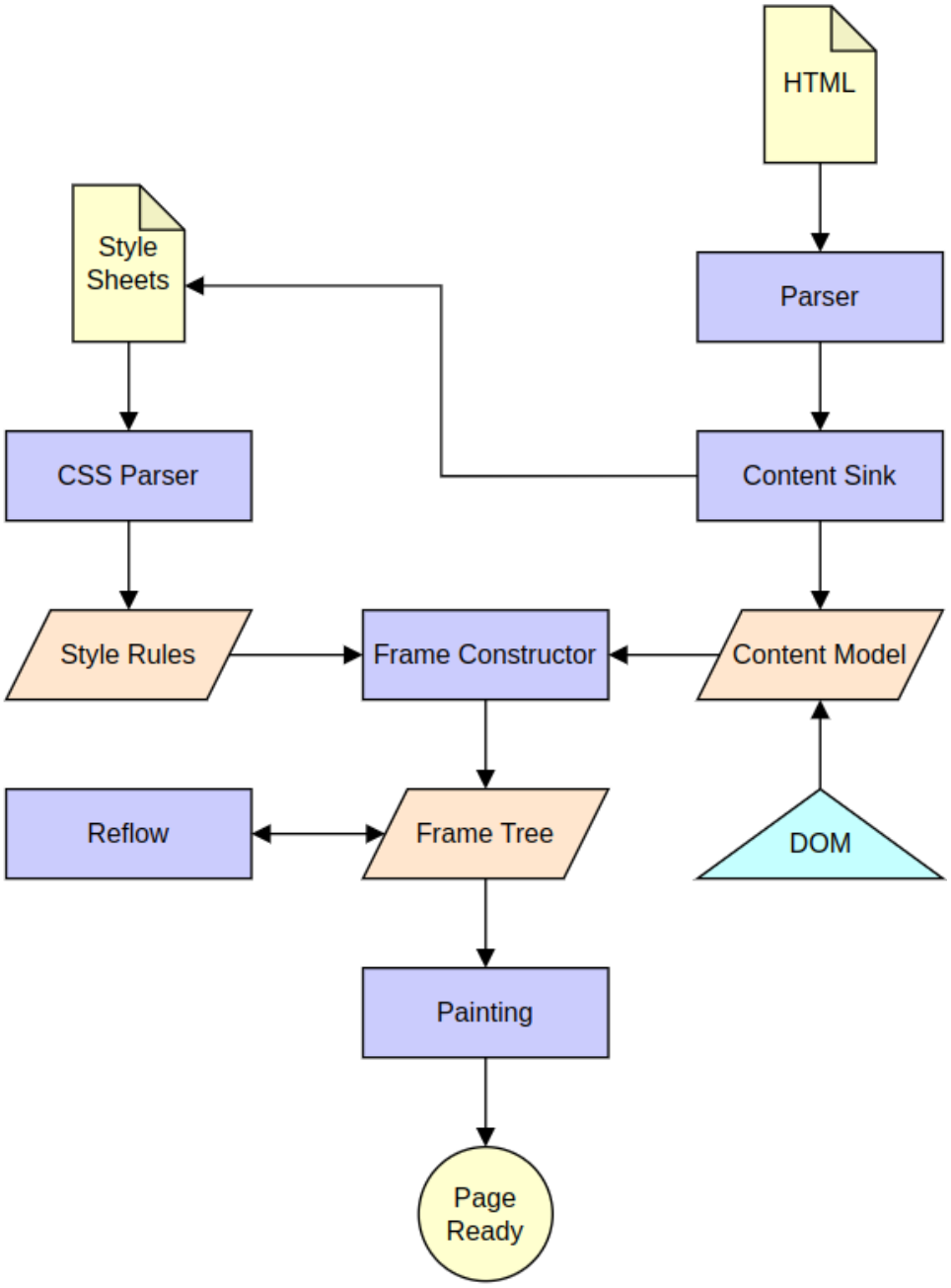

Figure 3.3 Gecko engine's general flow [39]

The rendering engine processes and displays the HTML and CSS files immediately, to the best of its ability, even if not all HTML/CSS files are yet delivered through the network. The rendering process is gradual and is partially or fully repeated multiple times if needed.

The rendering engine parses an HTML file and creates a tree called "content tree" filled with the DOM (Document Object Model) [40] nodes (Fig. 3.2-3). The same process happens for CSS files and style definitions in the HTML file. Gecko has one additional step called "content sink" which is a factory for creating DOM elements.

Next, these two trees are merged into a new tree called "render tree" or "frame tree". The order of elements is preserved in the same order they will be rendered. The render tree includes "objects" or "frames" containing information about styles such as dimensions or background colour.

Then, the render tree undergoes a process called “layout” or “reflow” wherein each DOM node is assigned the exact coordinates on the screen.

Lastly, the render tree is “painted”, that is the User Interface Backend layer of a browser is employed to show a user the web page.

3.3 Global desktop browser market share

The market share of desktop web browsers remains dynamic, influenced by a multitude of factors ranging from technological innovations to user preferences. Key players such as Google Chrome, Mozilla Firefox, Microsoft Edge [41], and Safari compete for dominance, with market share oscillating in response to shifts in user behaviour and technological advancements. An examination of the latest statistics and trends provides a crucial understanding of the current state of the browser market.

However, the task of accurately measuring market share is complicated by certain challenges, notably the issue of user agent [42] string uniformity. The user agent string, a line of text that browsers send to websites to identify themselves, has historically been used to distinguish between different browsers and their versions. However, many modern browsers, for compatibility reasons, use similar or identical user agents. This makes it difficult to accurately identify and differentiate between certain browsers, thus creating challenges in determining their true market share. For example, many browsers based on the Chromium project [43], including Google Chrome, Microsoft Edge, Opera [44], Brave [45] and Vivaldi [46] share a very similar user agent. As a result, certain analytical tools might inaccurately attribute visits from these browsers solely to Google Chrome, skewing the perceived market share. Also, certain anti-virus products fake their user agent to appear to be popular browsers. This is done to trick malicious sites that might display healthy content to the scanner, but not to the browser. Nonetheless, it is possible to estimate the browser market share and such data is available from multiple sources [10] (Fig. 3.4).

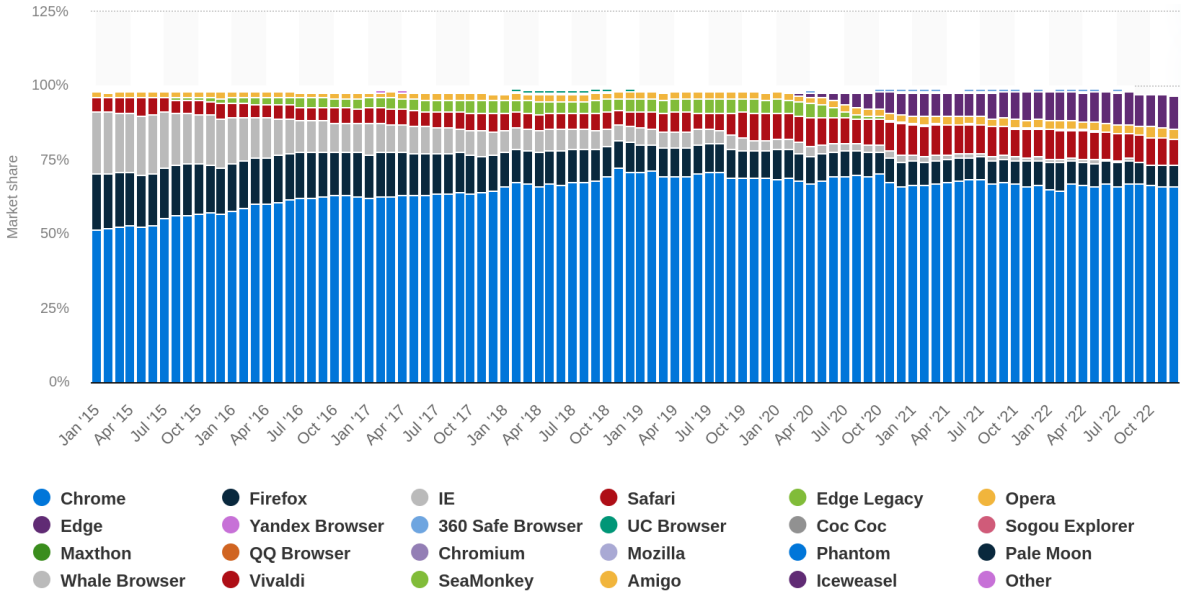

Figure 3.4 2015 to 2022 global desktop browser market share. Image source: [47].

4. Methodology

This section lays out a methodical approach to measuring the performance of CSS animations across a range of browsers and describes the specific hardware and software used in the process. The section also includes information about other unreliable methods of measuring the performance and imperfections of the chosen method. All of the browsers selected for this study are described and categorised in order to understand their inner workings and expected performance.

4.1 Tested browsers

The following browsers have been selected to be tested based on the following criteria:

1. The browser has to be available in most countries and regions of the world
2. The browser and its browser engine have to be actively developed
3. The browser must not be restricted to only one operating system, especially if it is not open-source
4. The browser must not be text-only – it has to support modern CSS (CSS3)
5. The browser should possess distinctive characteristics relevant to the topic of the thesis such as a unique browser engine
6. The browser should have a significant or visible browser market share

Based on the aforementioned criteria, the following 9 browsers were selected:

4.1.1 Brave

Brave [45] is a free and open-source browser based on Chromium and therefore based on the Blink browser engine and V8 JavaScript engine (Fig. 4.1). It reportedly has almost 60 million monthly active users and over 22 million daily active users [48]. It supports a variety of features that extend the out-of-the-box Chromium functionalities. Apart from that, it has great scores in protecting user privacy and preventing fingerprinting making it a growingly popular browser.

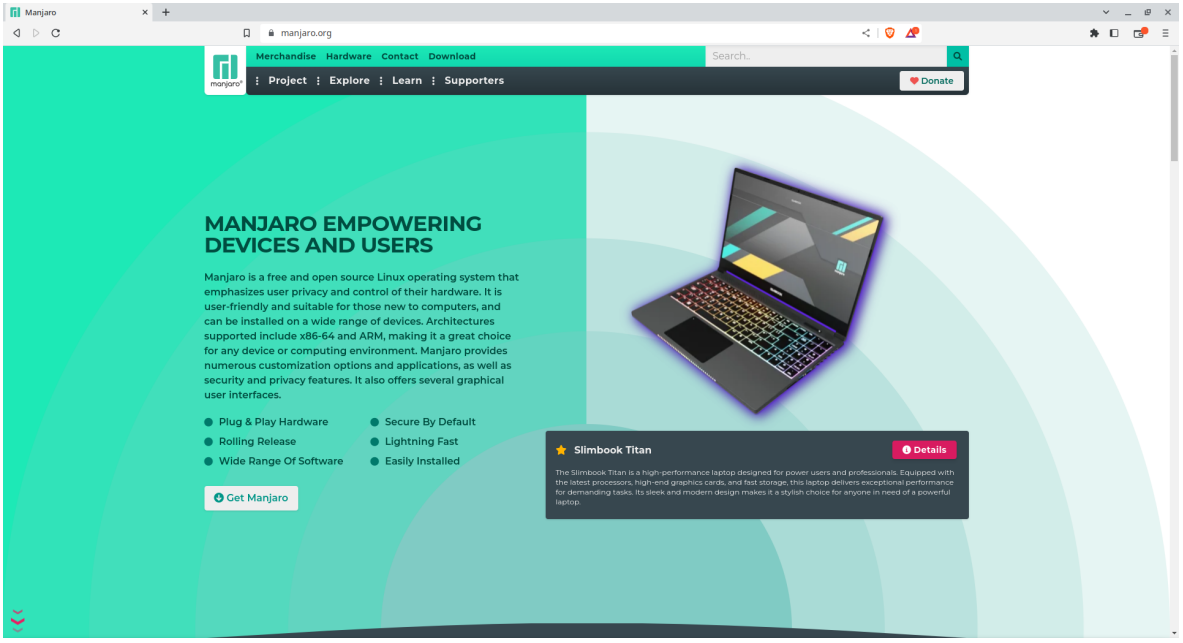

Figure 4.1 The main interface of Brave

4.1.2 Chromium

Chromium [43] is a free and open-source browser created and maintained mostly by Google [49] (Fig. 4.2). It is the basis of many browsers including Google Chrome. Its codebase comprises 37 million lines of code [50]. It uses the Blink and V8 engines.

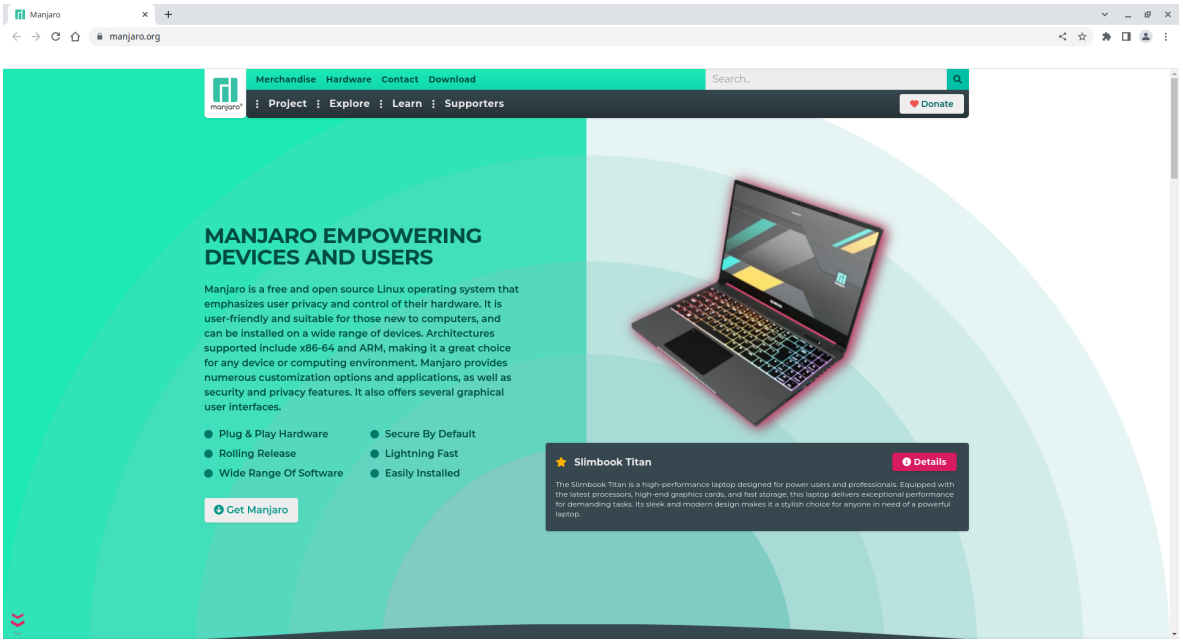

Figure 4.2 The main interface of Chromium

4.1.3 Google Chrome

Google Chrome [21] is the most popular browser in the world dominating the browser market share (Fig. 4.3). This proprietary browser was developed by Google. It is based on Chromium and therefore uses Blink and V8 under the hood.

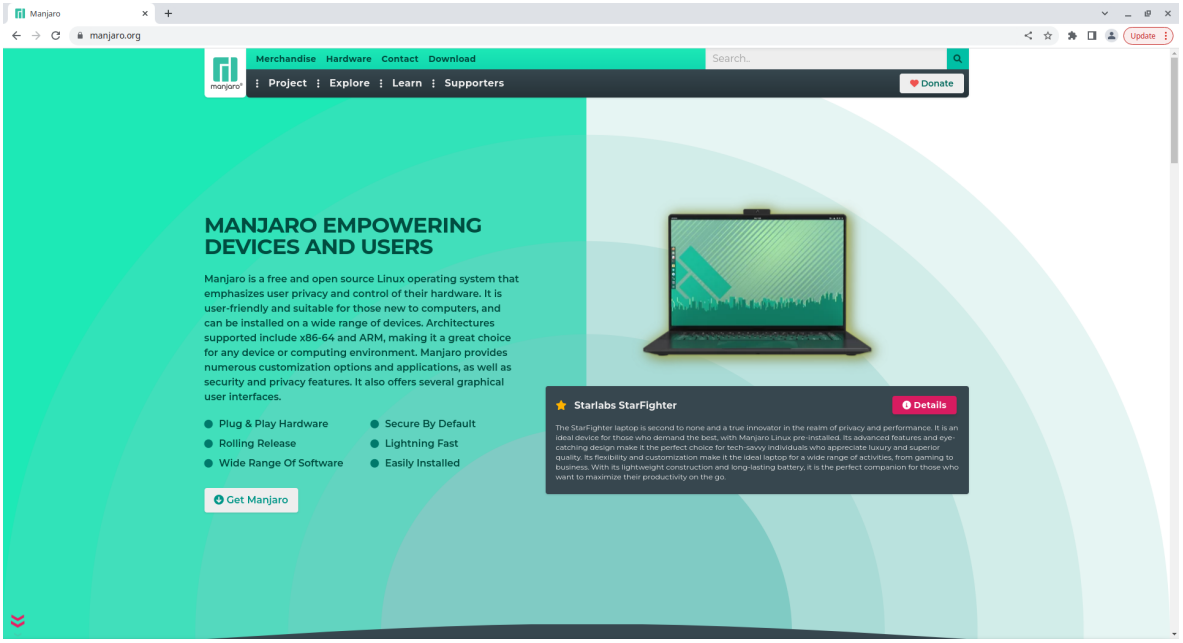

Figure 4.3 The main interface of Google Chrome

4.1.4 Konqueror

Konqueror [51] is a free and open-source browser and file manager created by the KDE project [52] (Fig. 4.4). It was replaced in KDE by Dolphin [53] as a file manager but continues to be a default browser. It initially used KHTML [54] as a browser engine, then due to its modular nature, it incorporated part of the WebKit browser engine (called WebKitPart [55]). For this reason, it was possible to test this browser as KHTML does not support CSS Animations. Moreover, this year (2023) KHTML was discontinued and will likely be substituted by WebKit in Konqueror in the future. It uses KJS [56] as a JavaScript engine. Nonetheless, it was decided to test this browser as KHTML was the basis of two modern browser engines that is WebKit and Blink.

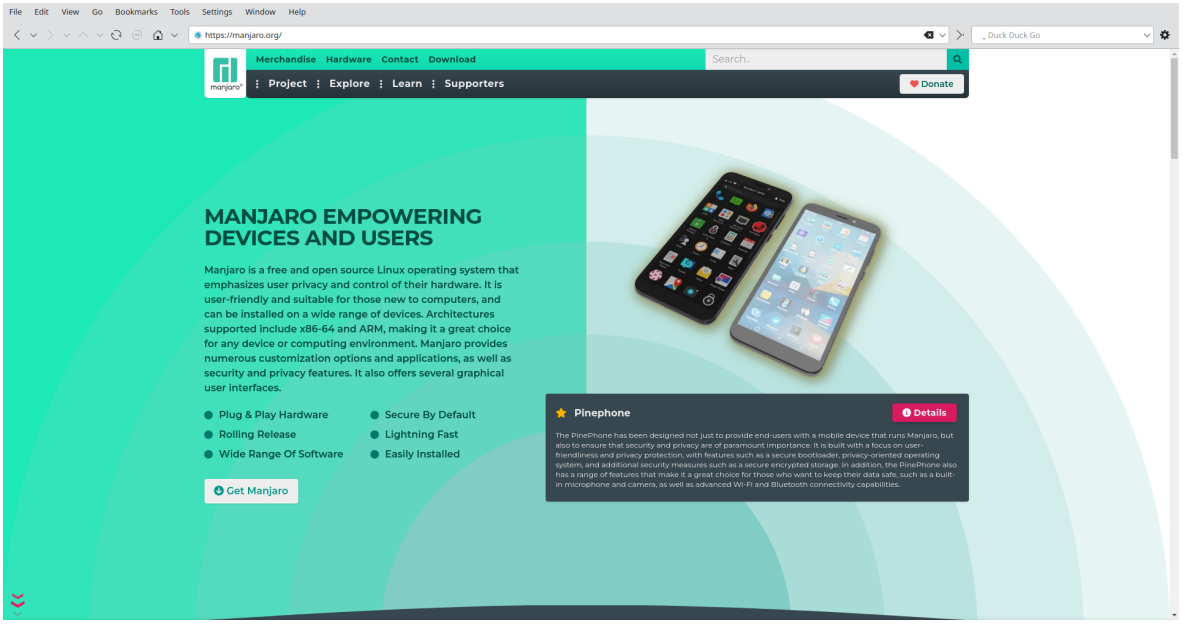

Figure 4.4 The main interface of Konqueror

4.1.5 Microsoft Edge

Microsoft Edge [41] is a proprietary browser developed by Microsoft [57] (Fig. 4.5). It is a successor of legacy Internet Explorer [58] and the default browser on the latest Windows 11 [59] operating system. It is based on Chromium and therefore uses Blink and V8.

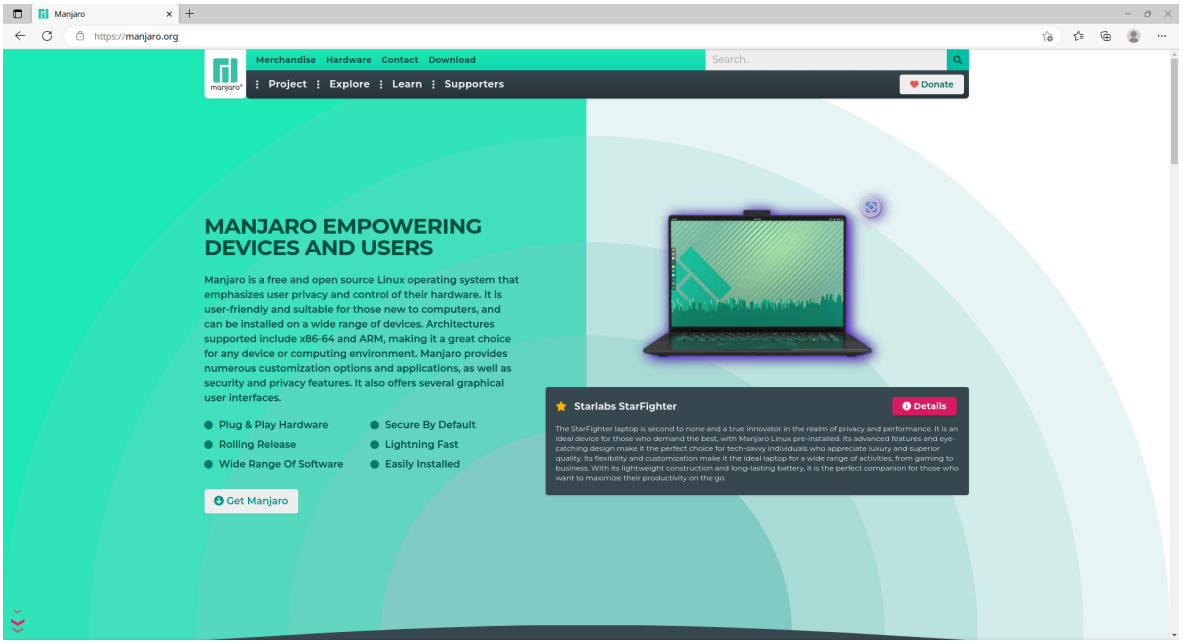

Figure 4.5 The main interface of Microsoft Edge

4.1.6 Mozilla Firefox

Mozilla Firefox [19] is a free and open-source browser developed and maintained by the Mozilla Foundation [60] (Fig. 4.6). It has a significant market share in the desktop browser space and uses Gecko as a browser engine and SpiderMonkey as a JavaScript engine. It has a number of third-party forks and is often preinstalled in various GNU/Linux [61] distributions.

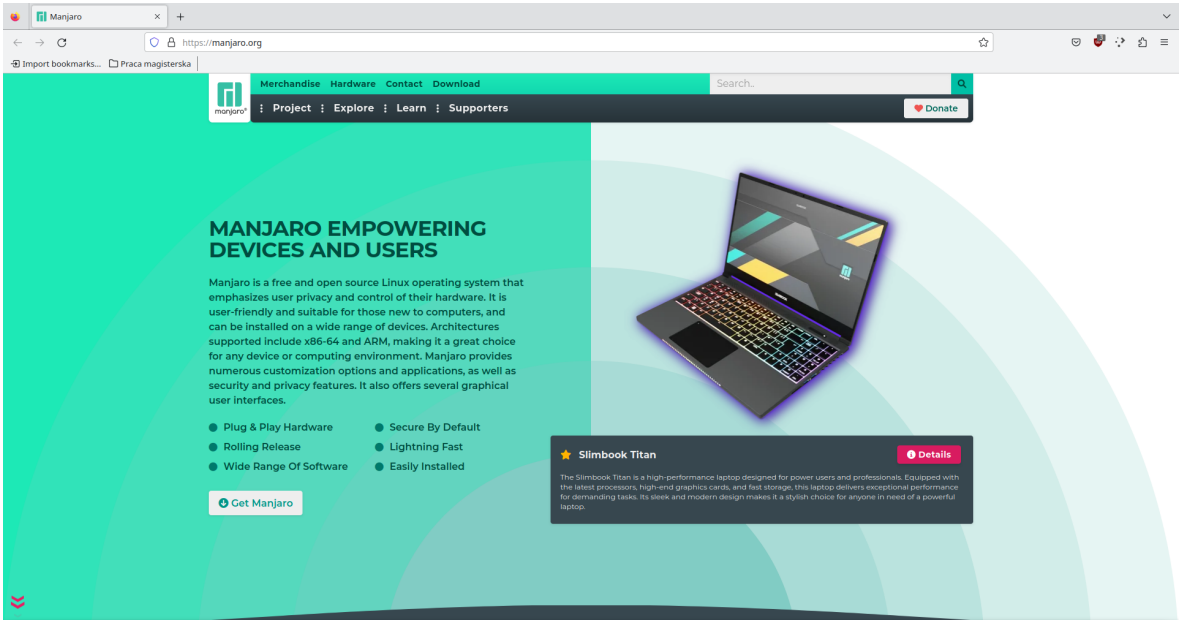

Figure 4.6 The main interface of Mozilla Firefox

4.1.7 Opera

Opera [44] is a proprietary browser developed by the company Opera [62] with a significant user base (Fig. 4.7). It is based on Chromium but provides additional features. In the past Opera had its own proprietary browser engine Presto [63] but in 2013 the company switched to Blink and V8 and discontinued working on Presto.

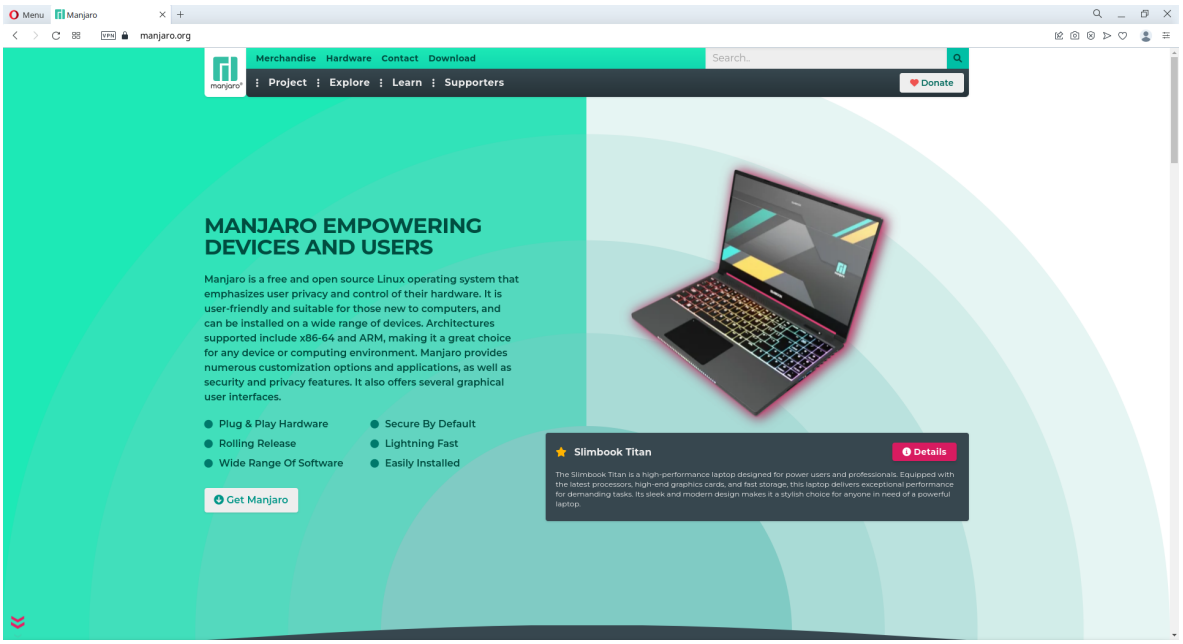

Figure 4.7 The main interface of Opera

4.1.8 Vivaldi

Vivaldi [46] is a proprietary browser created by Vivaldi Technologies [64] (Fig. 4.8). It aims to revive some of Opera's features lost when switching from Presto to Chromium

and Blink, despite itself being based on Chromium. It is directed towards more computer-literate users and gathers around 2.5 million active users.

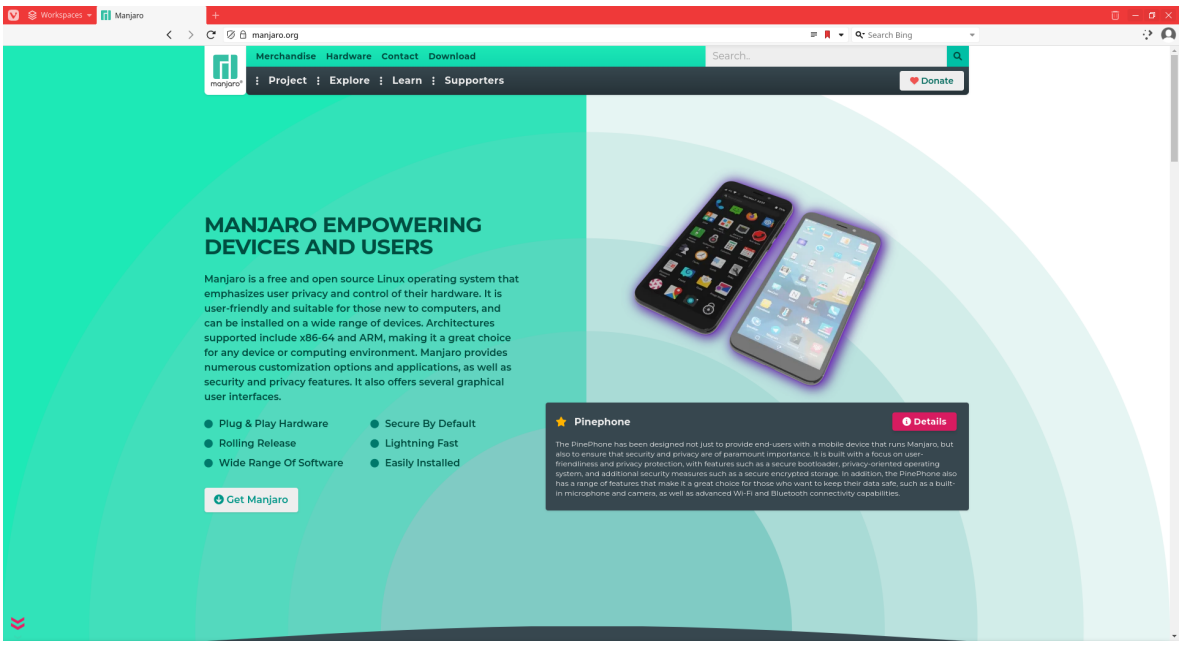

Figure 4.8 The main interface of Vivaldi

4.1.9 Yandex Browser

Yandex Browser [65] is a proprietary browser developed by the corporation Yandex [66] (Fig. 4.9). It is based on Chromium and by default uses Yandex Search [67] (search engine) instead of Google Search in order to increase its popularity. It also provides additional features over the default Chromium experience.

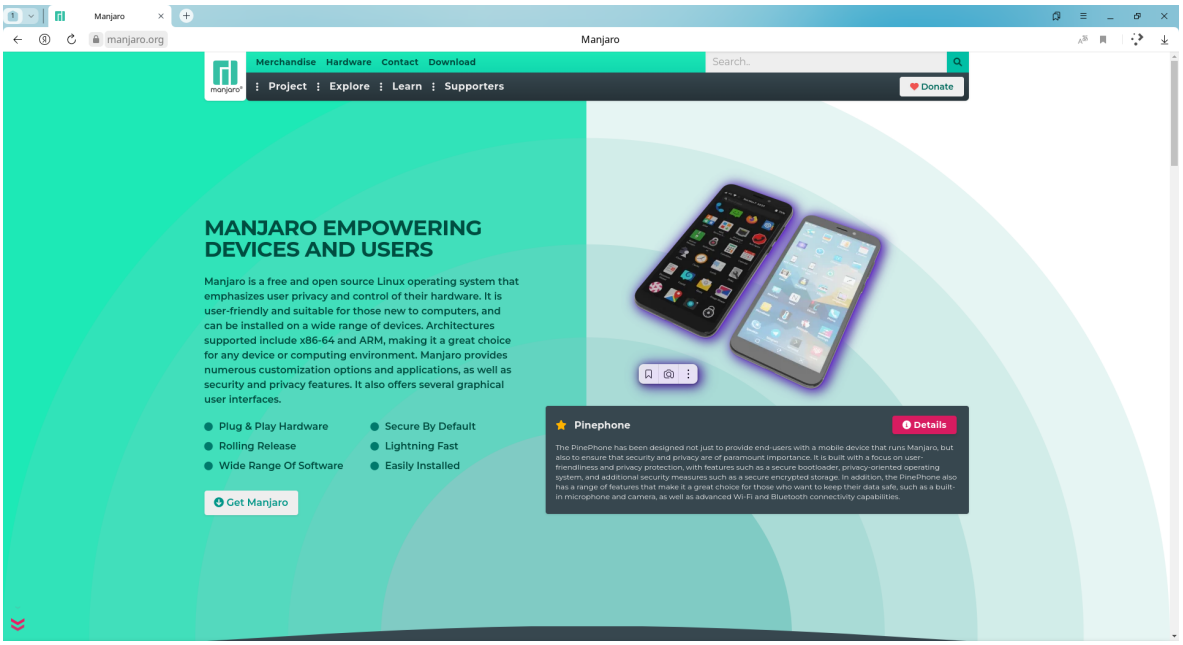

Figure 4.9 The main interface of Yandex Browser

Fig. 4.10 presents the categorisation of the browsers mentioned above based on the JavaScript and browser engines they use:

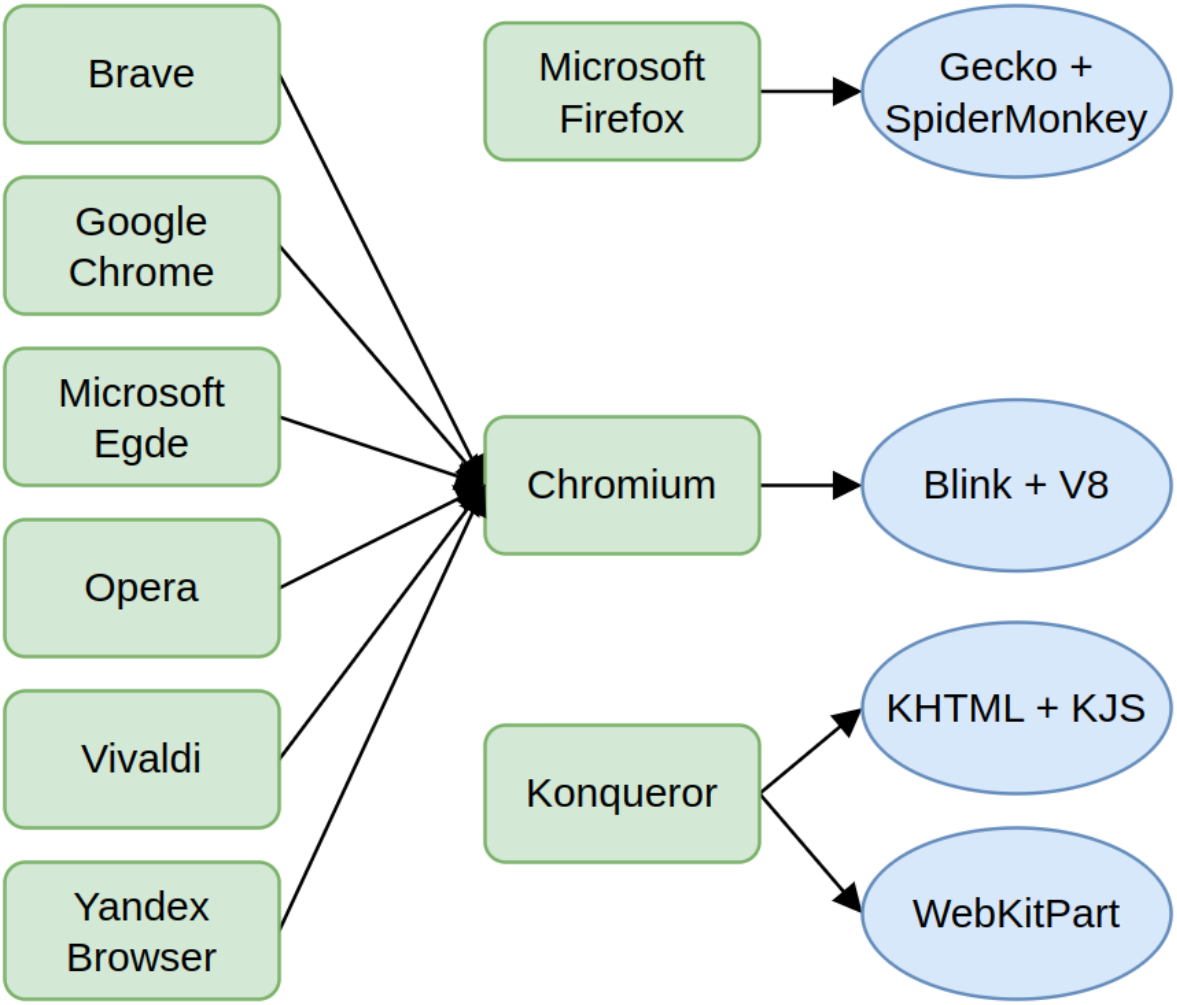

Figure 4.10 Classification of browsers based on the underlying browser and JavaScript engines

4.2 Browsers that were omitted

There are a few browsers that have not been tested despite having a significant or visible desktop browser market share and these are Safari [23], Internet Explorer, 360 Secure Browser [68] and UC Browser [69].

Despite the fact that Safari holds a large portion of the desktop browser market share it was not tested due to the fact that it is available only on macOS [70] (counting the desktop operating systems). In order to perform measurements for Safari the whole underlying hardware would have to be changed which would make them scientifically incomparable.

360 Secure Browser, Sogou Explorer [71], Maxthon [72] and QQ Browser [73] are not available for Linux and are mainly or completely directed towards the Chinese market.

UC Browser is a mobile-oriented browser and therefore it was not tested.

Coc Coc Browser [74] is a browser directed towards the Vietnamese market and is not available for GNU/Linux and for these reasons, it was not tested.

Internet Explorer – though a popular browser many years ago as it was preinstalled on the Windows operating system [75], nowadays is not supported by modern front-end frameworks (except by using polyfills [76]). Moreover, it is deprecated and in maintenance mode (as well as its rendering engine – Trident [77]).

This thesis encompasses only desktop browsers – it does not include measurements for tablet and mobile users as the number of tablet and mobile devices is huge in number with varying screen resolutions, hardware, operating systems and different additional software preinstalled on these devices by the manufacturers. Also, many older smartphone models have fluidity issues with just the operating systems let alone with browser and CSS animations running on the screen. Moreover, it would be difficult to gather enough devices to perform viable measurements, especially since some devices are available only in certain countries or the same device is distributed with different specifications depending on the country it is released. Therefore, the scale of this study would be greatly multiplied and it was decided to do so.

4.3 Technical setup

A variety of tools was employed to measure the performance of CSS animations. This subsection presents and explains wherever it is needed why these tools were used.

4.3.1 Hardware and the operating system

The following hardware was used to conduct this study – a standalone computer (PC) with:

1. Processor: Intel Core i5-6600K
2. Monitor with screen resolution 1920x1080, 60Hz
3. 16 GB of DDR4 RAM

As for the operating system, Manjaro [78] release 22.1.2 “Talos” [79] one of the GNU/Linux distributions was selected. It is free and open-source, and by default has a preinstalled KDE Plasma [80] desktop environment. It is based on Arch Linux [81] and follows the curated rolling release model [82]. This makes it a quite standard and approachable GNU/Linux distribution for an average user.

The reason why a macOS hasn’t been chosen is that only a limited number of browsers can be installed on it. Moreover, most GNU/Linux distributions have fewer processes and analytics running in the background, therefore, creating a more stable and reliable setup for the animations to be tested. Moreover, most GNU/Linux distributions are free and open-source thereby creating a consumer-friendly environment, respecting user privacy and stimulating market competition and general improvement of products.

4.3.2 Supporting software

A handful of tools have been used in order to perform various activities necessary in carrying out the comparison of the performance of CSS animations. These tools include:

1. OBS Studio

OBS Studio [83] is a widely popular free and open-source software that was used to record the display during the measuring of animations (Fig. 4.11). It was initially created by Lain Bailey and is now developed by various contributors around the globe.

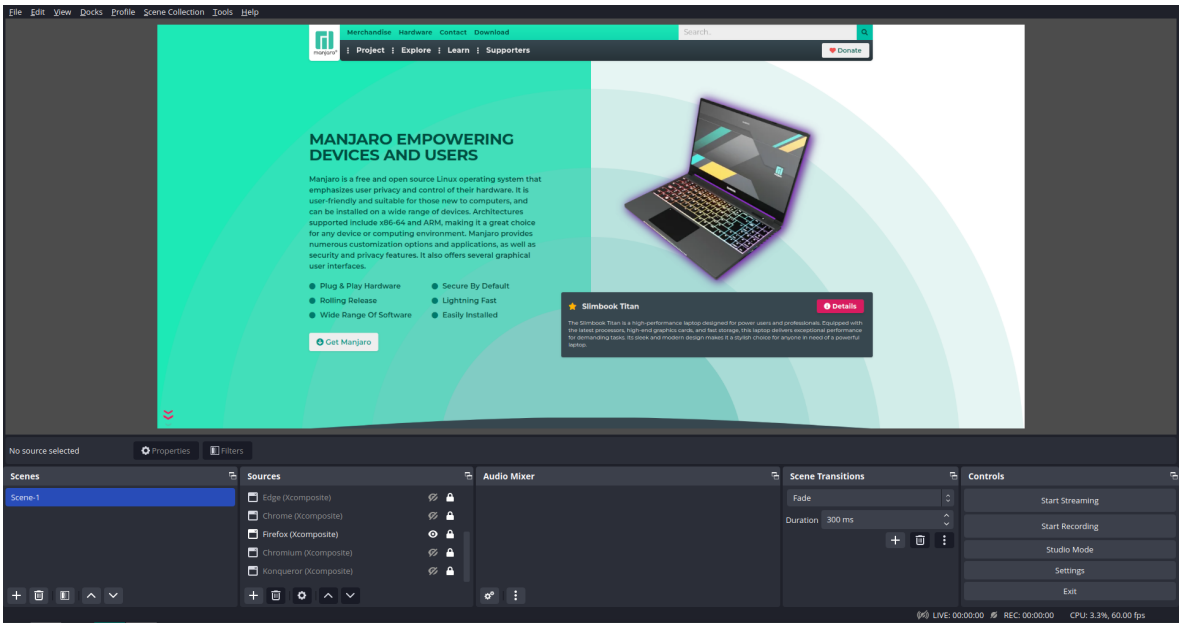

Figure 4.11 The main interface of OBS Studio

2. WebStorm

WebStorm [84] is a sophisticated proprietary Integrated Development Environment (in short, IDE) [85] produced by company JetBrains [86] (Fig. 4.12). It belongs to a bundle of related IDEs for various programming languages and technologies providing a common interface to all of them. It was used to prepare animations for the thesis, graphs and other helper scripts.

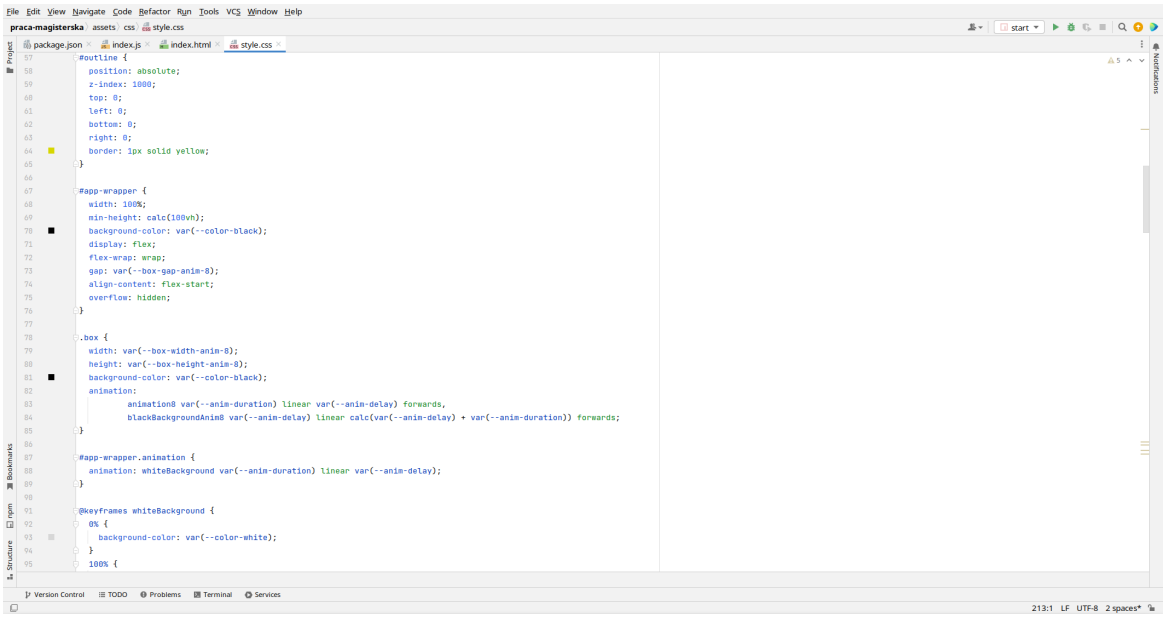

Figure 4.12 The main interface of WebStorm

3. Kate

Kate [87] is a free and open-source source code editor developed under the umbrella of the KDE community of free software developers, bundled by default with the KDE Plasma desktop environment (Fig. 4.13). It was used to create and modify helper scripts for measuring the CSS animations for the thesis.

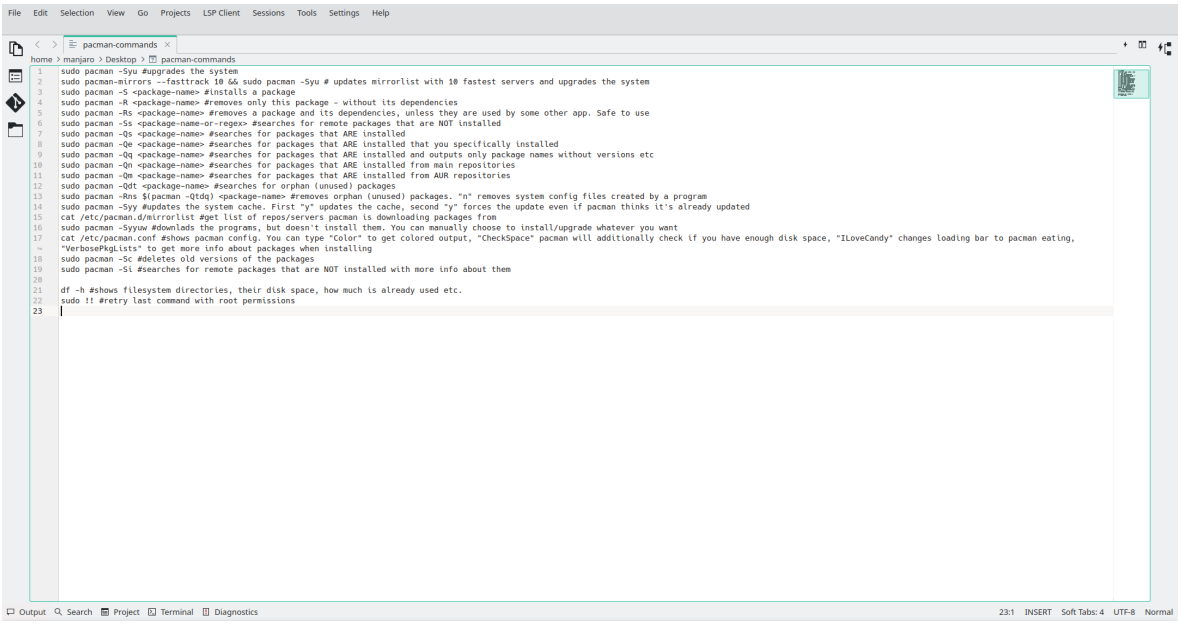

Figure 4.13 The main interface of Kate

4. mpv

mpv [88] is a free and open-source media player (Fig. 4.14). It was used to watch and analyze recorded animations.

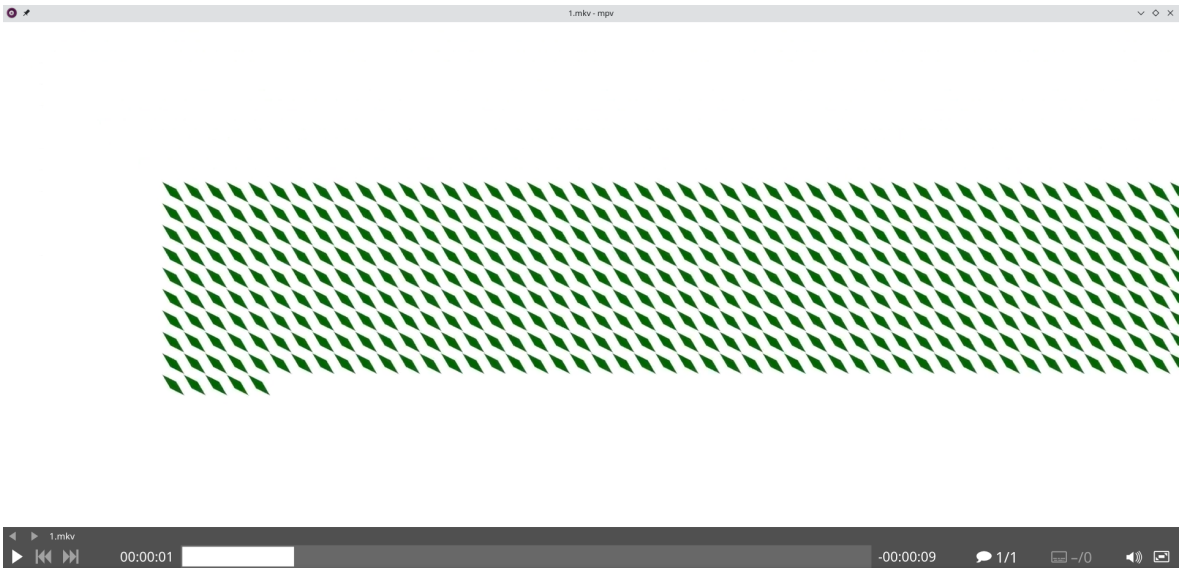

Figure 4.14 The main interface of mpv

5. Melt

Melt [89] is a free and open-source media player and video editor controlled through the command-line interface (CLI) [90] (Fig. 4.15). It was used for its functionality of playing videos frame by frame for the recorded CSS animations analysis.

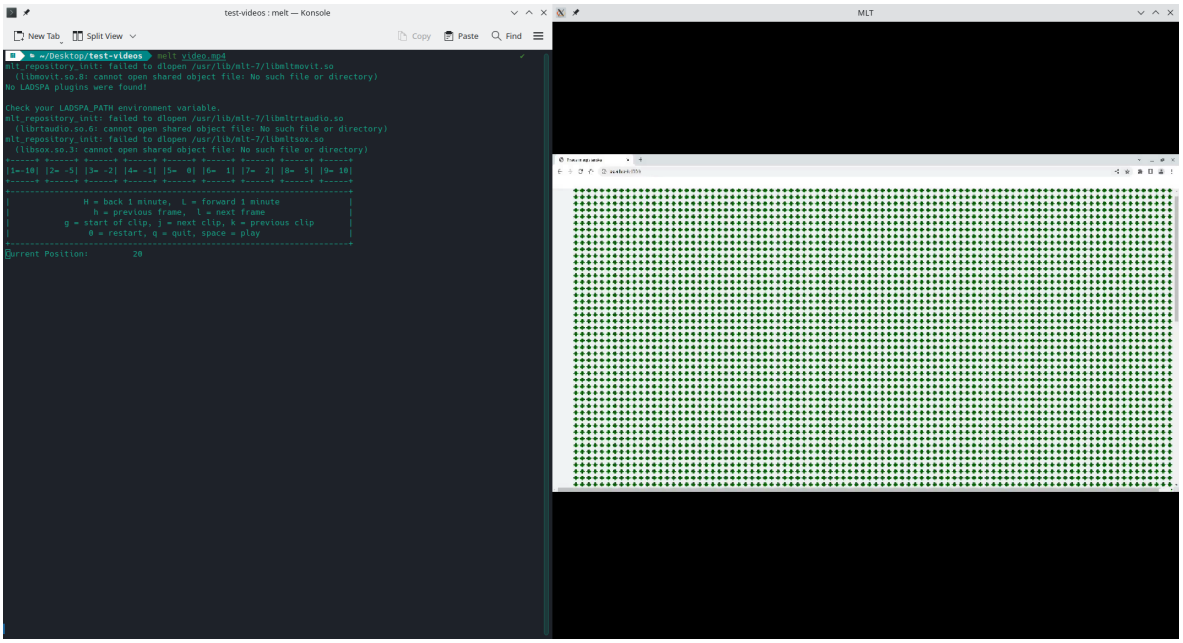

Figure 4.15 The main interface of melt

6. xdotool

Xdotool [91] is a free and open-source Xorg [92] automation tool. It played a key role in the automation process of recording animations for the thesis. Its basic functionality includes simulating the computer's keyboard and mouse but it can do other things, such as minimizing or resizing a window etc.

7. ffmpeg

Ffmpeg [93] is a free and open-source command-line program capable of advanced video and audio processing, converting, encoding, decoding and many more. It was for the purposes of this thesis to remove duplicate ("dead") frames from the recorded animations

8. MediaInfo

MediaInfo [94] is a free and open-source command-line tool that allows for viewing the tag data of video and audio files. This software was used to obtain the number of frames of the recorded animations before and after the removal of duplicate frames.

4.4 Animation's performance measuring

All animations were measured with a minimal number of programs running in the background (such as the recording software and the application itself) and all animations for a browser were run with a restarted (fresh) operating system.

A threshold is a variable containing the number of animated elements that appear on the screen. Each animation has its own thresholds defined. Each animation has five thresholds except animation 6 which has six thresholds for increased accuracy. Each threshold value is specified in Subsection 5.1.

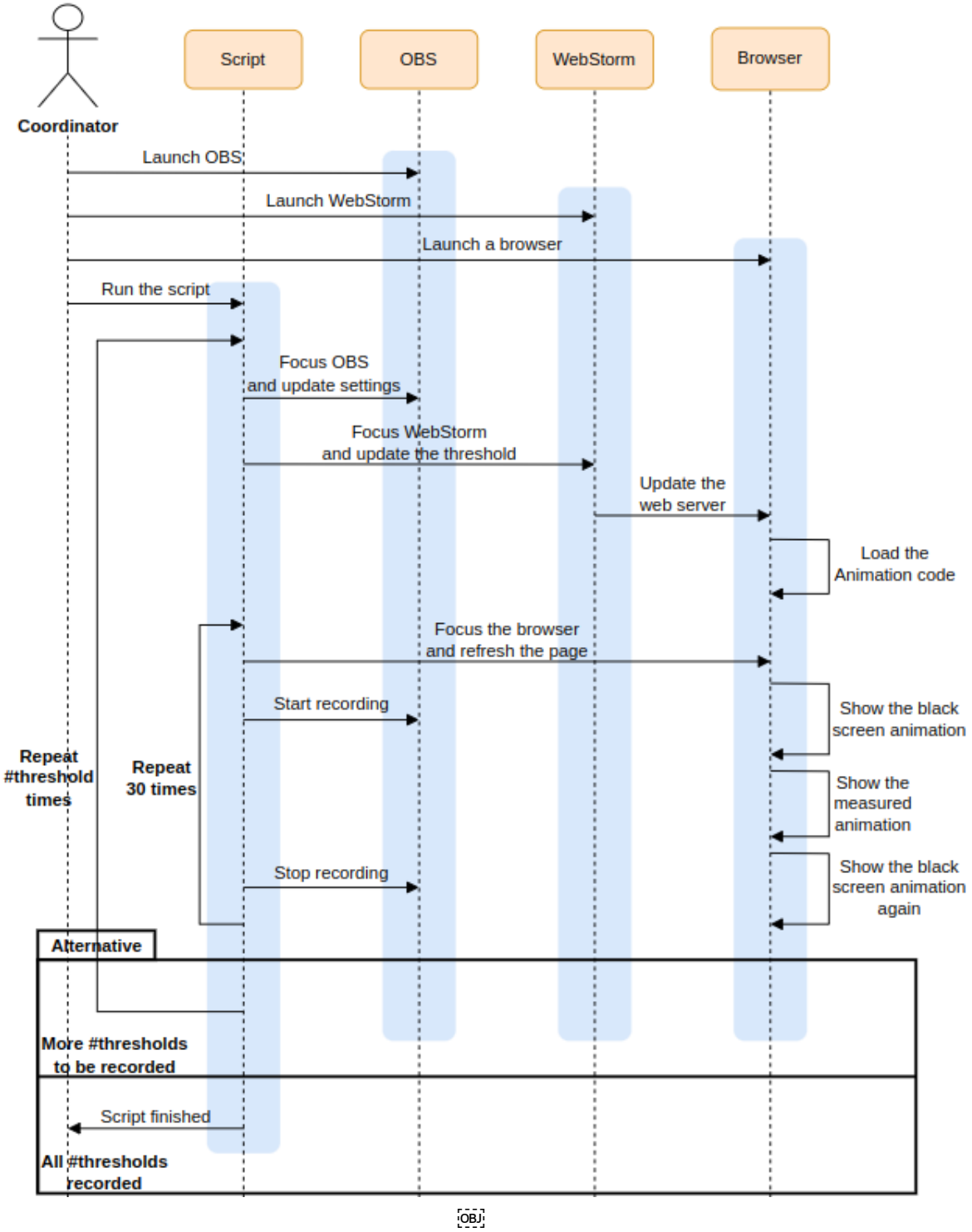

Figure 4.16 The process of performing measurements

The process presented in Fig. 4.16 consists of the following steps:

1. With just the necessary software running on the operating system, the automated script (written in Python [95]) recording the animations is launched and asks to input a browser name and animation number
2. The script opens the OBS Studio's window
3. The script updates the OBS Studio's settings that is the path recordings are saved to
4. The script opens the WebStorm's window
5. The script updates WebStorm's settings
6. The target browser is opened. The animation starts with showing a black screen
7. The OBS Studio starts recording when the black screen is still visible
8. The proper animation starts, runs and ends. Then the black screen is shown again.
9. The OBS Studio's recording is stopped
10. The page is refreshed. The black screen before the animation is visible again

Steps 2 to 6 are repeated 5 or 6 times that is the chosen number of thresholds to obtain data necessary to plot a graph. For each threshold steps 7 to 10 are repeated n, that is 30 times. This number was selected for statistical accuracy.

11. The script finishes
12. Animation and the OBS Studio settings (when switching browsers) are manually updated. This step could have been automated but it was decided not to as the OBS Studio would crash occasionally and had to be supervised anyway.

Between each of the steps mentioned above the script waits (sleeps) for a necessary number of milliseconds.

The whole process was repeated for each animation for each browser.

4.5 Other unreliable methods of measuring the performance of animations

There are other methods of measuring the performance of CSS animations that after consideration were rejected as unreliable.

The first method is performing measurements through JavaScript. This method involves using a library such as stats.js [96] to get current frames per second (fps) [97] or measuring them directly through the JavaScript method "requestAnimationFrame()" [98] which is called just before screen repaint. The "requestAnimationFrame()" therefore seems to be a good candidate as it is usually called 60 times per second or the browser matches the display frame rate. Unfortunately, these methods are not reliable when measuring CSS animations as they are sometimes run on a different thread thus lowering the fps measured by JavaScript.

The second method involves using the Developer Tools [99] provided by a browser to obtain a detailed analysis of the animation performance. This method however does not work for cross-browser testing and had to be rejected. Not all browsers provide such tools and even if they do, the measurements are not always reliable, especially for the CSS animations.

To sum up, it was decided that the best method to measure the performance of the CSS animations is to measure the fluidity as shown by the browser on the display and the same way the user sees them.

4.6 Problems with the chosen method of measuring performance

The selected method of measuring the performance of the CSS animations by recording the display and removing duplicate frames does bear some disadvantages. Firstly, the algorithm that removes duplicate frames has to be properly calibrated – it cannot delete frames that are similar to each other as this would lower the real frame rate. Secondly, the chosen method cannot detect certain issues such as observed problems with the Mozilla Firefox browser as it had a problem switching animations. After the animation was finished, the black screen indicating the end of the animation would not appear for a significant (visible) duration of time. This problem starts appearing and grows as the browser starts struggling with the increasing number of elements – the switching takes longer and longer to execute. This notably decreases the user experience and therefore should be improved.

5. Animations

This section presents an explanation of the animations that were tested and measured in terms of performance. It aims to depict them visually and through code in order to understand the results presented in Section 6.

5.1 General setup

The general setup of the animations is relatively simple and aims not to impede the performance of the measured animations. This requires a few complications but is nonetheless as straightforward as possible.

Table 5.1 presents the thresholds specified for each animation:

	Threshold number					
Animation number	1	2	3	4	5	6
1	50	500	1000	3000	4000	-
2	100	1000	2000	3000	5000	-
3	100	500	1000	2000	3000	-
4	100	500	1000	2000	3000	-
5	500	1000	1500	2000	3000	-
6	50	1000	4000	5000	10000	20000
7	100	500	1000	2000	5000	-
8	100	500	1000	2000	5000	-
9	500	1000	1500	2000	3000	-
10	10	50	100	120	200	-

Table 5.1 Threshold definitions for each animation

The project contains three essential for the thesis files – one containing HTML code, one with JavaScript code and one with CSS code. The HTML code is the simplest and presented in Listing 5.1:

Listing 5.1 HTML file of the project containing animations

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Master's thesis</title>
  </head>
  <body>
    <div id="app-wrapper"></div>
  </body>
</html>
```

Apart from the basic HTML structure, it contains a div inside a <body> tag with an id of “app-wrapper”. This div is a wrapper for all n elements, where n is the number of elements for a given animation for a given threshold. Those elements are injected through the JavaScript code (Listing 5.2). Firstly, the CSS style sheet is imported:

Listing 5.2 The JavaScript file of the project containing animations

```
import '../assets/css/style.css';

const appWrapper = document.getElementById( elementId: 'app-wrapper');

for (let i = 0; i < anim10Thresh1; i++) {
    const div = document.createElement( tagName: "div");
    div.classList.add("box");
    appWrapper.appendChild(div);

    /* ======= code for animation 10 ======= */
    const front = document.createElement( tagName: "div");
    const back = document.createElement( tagName: "div");
    const left = document.createElement( tagName: "div");
    const right = document.createElement( tagName: "div");
    const top = document.createElement( tagName: "div");
    const bottom = document.createElement( tagName: "div");
    front.classList.add("front");
    back.classList.add("back");
    left.classList.add("left");
    right.classList.add("right");
    top.classList.add("top");
    bottom.classList.add("bottom");

    div.appendChild(front);
    div.appendChild(back);
    div.appendChild(left);
    div.appendChild(right);
    div.appendChild(top);
    div.appendChild(bottom);
}

appWrapper.classList.add('animation');
```

The JavaScript code presented in Listing 5.2 first gets a handle to the "app-wrapper" div in the HTML file. Then, it contains a "for" loop which adds as many elements to "app-wrapper" as it is tested for the given animation for the given threshold. The values for the animations and subsequent thresholds are stored in variables such as "anim10Thresh1".

In the “for” loop a div of class “box” is added to the “appWrapper” handle. For animation 10 (and only for this animation) there is an additional code required to create a setup for this animation, namely to create a 3D cube. For each side of a cube (in total 6 sides), there are div elements being added inside of the “box” div with the following classes: “front”, “back”, “left”, “right”, “top” and “bottom”.

Lastly, after the generation of all elements “appWrapper” is ready to play the animation. Therefore, an “animation” class is added to it. A black background will appear and after a specified delay the proper animation will be displayed and afterwards, the black background will be brought back. This and more happen in the CSS file which looks as follows (Listing 5.3):

Listing 5.3 Basic CSS styles that apply to html and body tags

```
html,
body {
  padding: 0;
  margin: 0;
  overflow: hidden;

  --color-black: #000000;
  --color-white: #ffffff;
  --color-box: darkgreen;
  --color-box-anim-10: rgba(0, 100, 0, 0.5);
  --anim-1-duration: 5s;
  --anim-duration: 3s;
  --anim-delay: 1s;
}
```

These styles apply to html and body tags. They set the padding and margin of the website to zero and overflow to hidden so as to not show the scrollbar. The scrollbar is being resized as the elements are moving and this adds artificial frames to the recording thereby making the measurements unreliable. Apart from this, some basic self-explainable CSS variables have been prepared. These variables can be reused throughout the CSS file.

Listing 5.4 CSS styles that apply to div with the id of “app-wrapper”

```
#app-wrapper {
  width: 100%;
  min-height: calc(100vh);
  background-color: var(--color-black);
  display: flex;
  flex-wrap: wrap;
  gap: var(--box-gap-anim-10);
  align-content: flex-start;
  overflow: hidden;
}
```

The “app-wrapper” element in Listing 5.4 contains styles that make it occupy the full width and height of a browser window and put the elements next to each other with a specified gap between them. It also sets overflow to “hidden” for the same reasons as in the “html” and “body” tags. The background is set to black colour.

Listing 5.5 CSS styles that apply to “app-wrapper” and “box” elements. Definition of animations for white and black backgrounds.

```
.box {
  width: var(--box-width-anim-10);
  height: var(--box-height-anim-10);
  background-color: var(--color-black);
  animation:
          animation10 var(--anim-duration) linear
          var(--anim-delay) forwards,
          blackBackground var(--anim-delay) linear
          calc(var(--anim-delay) + var(--anim-duration)) forwards;
}

#app-wrapper.animation {
  animation: whiteBackground var(--anim-duration) linear var(--anim-delay);
}

@keyframes whiteBackground {
  0% {
    background-color: var(--color-white);
  }
  100% {
    background-color: var(--color-white);
  }
}

@keyframes blackBackground {
  0% {
    background-color: var(--color-black);
  }
  100% {
    background-color: var(--color-black);
  }
}
```

The “box” divs get their width and height set along with the background colour of black to match the black background at the beginning of the animation (Listing 5.5). They also have an animation that is delayed till the end of the proper animation (the one that is measured) to bring back the black colour of a box.

When the “app-wrapper” parent containing boxes is appended to the “animation” class it changes the black background to white to display the proper animation.

Lastly the “whiteBackground” and “blackBackground” are animations marked by the “@keyframes” keyword. The 0% and 100% denote the beginning and the end of these animations. During the time that these animations are played, respectively the white and black colour background is applied to the elements that these animations are attached to.

5.2 Animation 1

The first animation is a simple translation (Fig. 5.1). The elements start from the top left side, move to the right side on the X-axis for 700 pixels and then move back to the starting position.

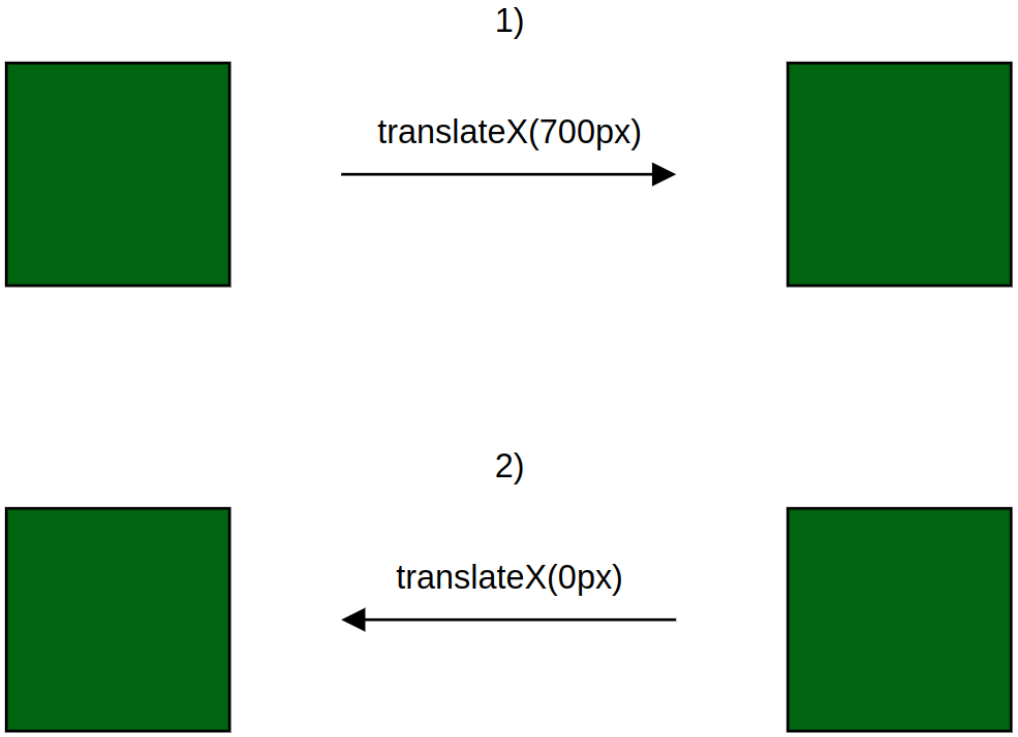

Figure 5.1 Depiction of animation 1
1) Phase of translating the element to the right
2) Phase of translating the element to the starting position

The CSS code specific to animation 1 depicted in Listings 5.6-7 firstly defines CSS variables for the width and height of the elements with class “box” and the gap between the boxes. Then, the animation specified in “@keyframes” starts at 0% with a box translation on the X-axis with a value of 0 (no translation) and sets the background colour of a box to “darkgreen”. At 50% which is in the middle of the animation, the box is moved on the X-axis for 700 pixels. Lastly, at 100% representing the end of the animation a box moves to the starting position and the background colour of “darkgreen” is maintained. It is enough to set the background colour at 0% and 100% and to omit the duplicate definition at 50% – the background will stay the same colour throughout the animation. At the end of the animation, the background colour will fall back to its default, which is black in order to match the black background of the “app-wrapper” parent.

Listing 5.6 The first part of the code specific to animation 1

```
--box-width-anim-1: 15px;
--box-height-anim-1: 15px;
--box-gap-anim-1: 5px;
```

Listing 5.7 The second part of the code specific to animation 1

```
@keyframes animation1 {
  0% {
    transform: translateX(0);
    background-color: var(--color-box);
  }
  50% {
    transform: translateX(700px);
  }
  100% {
    transform: translateX(0);
    background-color: var(--color-box);
  }
}
```

5.3 Animation 2

The second animation is a basic rotation (Fig. 5.2). The elements stay in place while being rotated over 360 degrees, that is two revolutions of a rectangle.

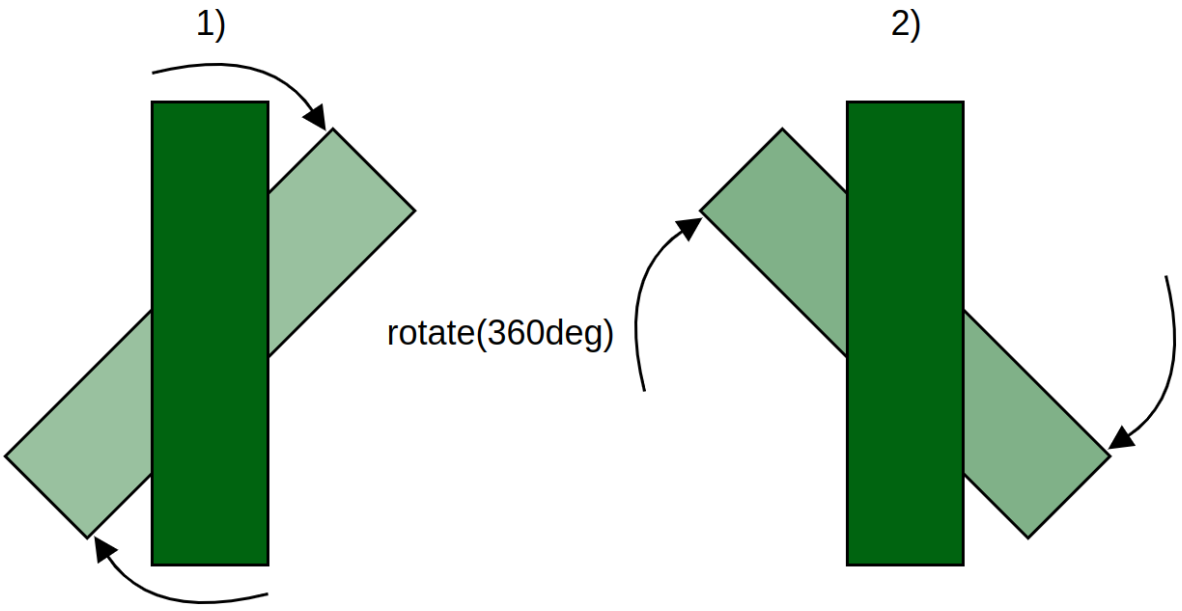

Figure 5.2 Depiction of animation 2
1) and 2) Phases of rotating the element over 360 degrees

The Listings 5.8-9 present the exact code executed for animation 2. Firstly, the variables defining the size of rectangles and the gap between them are initialised. Then, at 0% the rectangles are given the "darkgreen" colour contained in the "--color-box" CSS Variable and the rotation is set to start at 0 degrees. Finally, the elements are transformed from 0% to 100% where the rectangles will be rotated over 360 degrees.

Listing 5.8 The first part of the code specific to animation 2

```
--box-width-anim-2: 5px;
--box-height-anim-2: 20px;
--box-gap-anim-2: 7px;
```

Listing 5.9 The second part of the code specific to animation 2

```
@keyframes animation2 {
  0% {
    transform: rotate(0deg);
    background-color: var(--color-box);
  }
  100% {
    transform: rotate(360deg);
    background-color: var(--color-box);
  }
}
```

5.4 Animation 3

The third animation is scaling the element down and up (Fig. 5.3). The elements are not under any other transformations except the scaling.

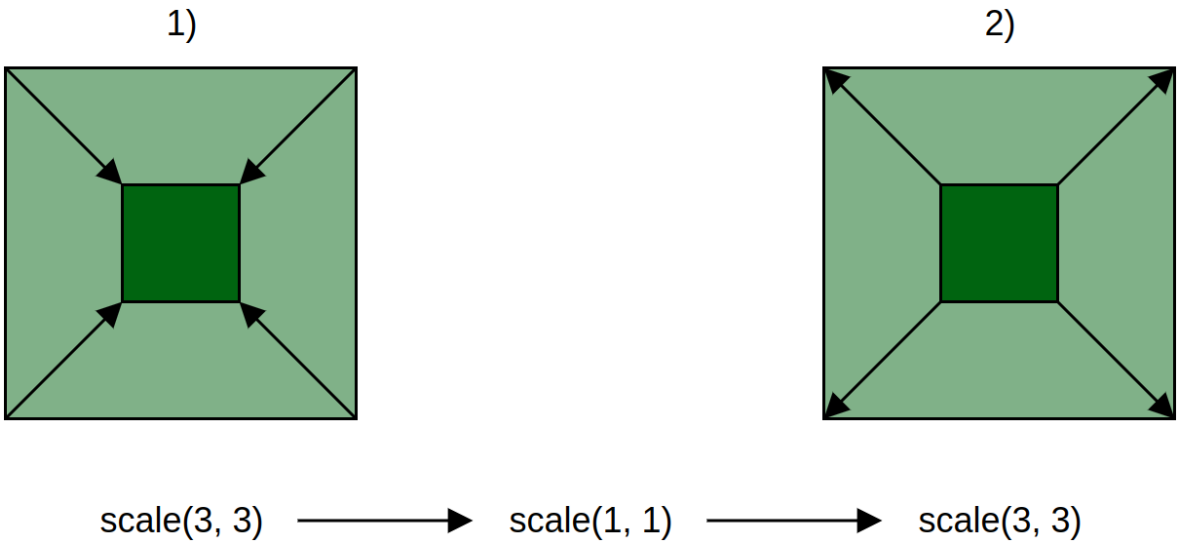

Figure 5.3 Depiction of animation 3
1) The elements are shrunk (scaled down)
2) The elements are scaled back up

The Listings 5.10-11 present the code executed for animation 3. Firstly, the variables that define the size of the elements and the gap between them are initialised. The animation defined in "@keyframes" is divided into three parts. The elements begin being scaled up threefold. Then, in the middle of animation at 50% they are scaled down to a size defined by the CSS properties. Lastly, they are scaled back up to the initial size. The "darkgreen" colour of elements is maintained throughout the duration of the animation.

Listing 5.10 The first part of the code specific to animation 3

```
--box-width-anim-3: 10px;
--box-height-anim-3: 10px;
--box-gap-anim-3: 24px;
```

Listing 5.11 The second part of the code specific to animation 3

```
@keyframes animation3 {
  0% {
    transform: scale(3, 3);
    background-color: var(--color-box);
  }
  50% {
    transform: scale(1, 1);
  }
  100% {
    transform: scale(3, 3);
    background-color: var(--color-box);
  }
}
```

5.5 Animation 4

The fourth animation is element skewing (Fig. 5.4). The elements are not under any other transformations except the skewing.

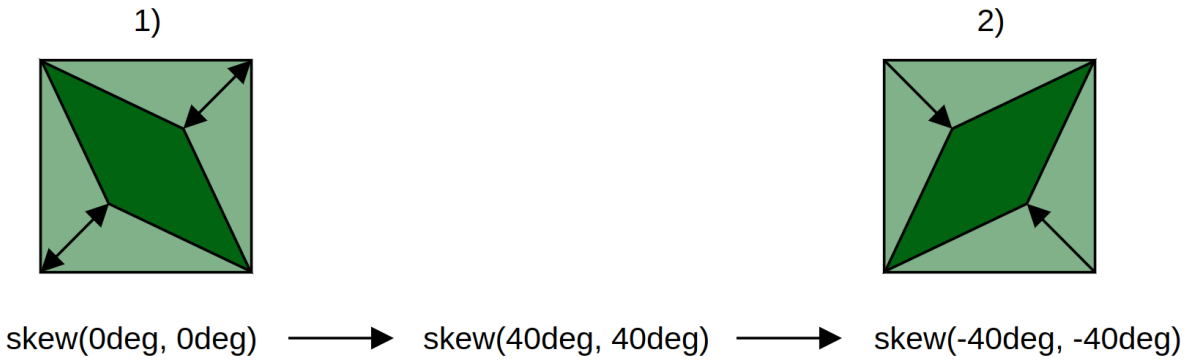

Figure 5.4 Depiction of animation 4
1) and 2) The elements are skewed on both diagonals

The Listings 5.12-13 present the code executed for animation 4. The elements are assigned size and the gap between them. The elements start untransformed. Then, at 50% they are skewed on the X and Y-axis by 40 degrees. Lastly, at 100% the elements are transformed to negative 40 degrees on both axes.

Listing 5.12 The first part of the code specific to animation 4

```
--box-width-anim-4: 18px;
--box-height-anim-4: 18px;
--box-gap-anim-4: 10px;
```

Listing 5.13 The second part of the code specific to animation 4

```
@keyframes animation4 {
  0% {
    transform: skew(0deg, 0deg);
    background-color: var(--color-box);
  }
  50% {
    transform: skew(40deg, 40deg);
  }
  100% {
    transform: skew(-40deg, -40deg);
    background-color: var(--color-box);
  }
}
```

5.6 Animation 5

The fifth animation is the element's simultaneous translation and skewing (Fig. 5.5-6).

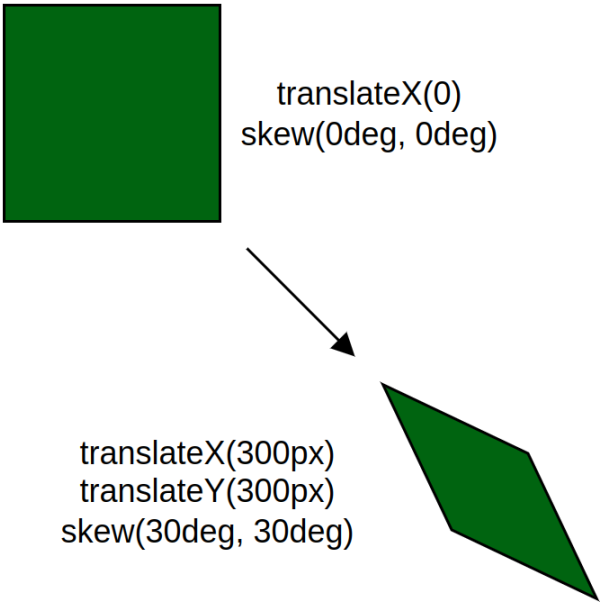

Figure 5.5 The first stage of animation 5

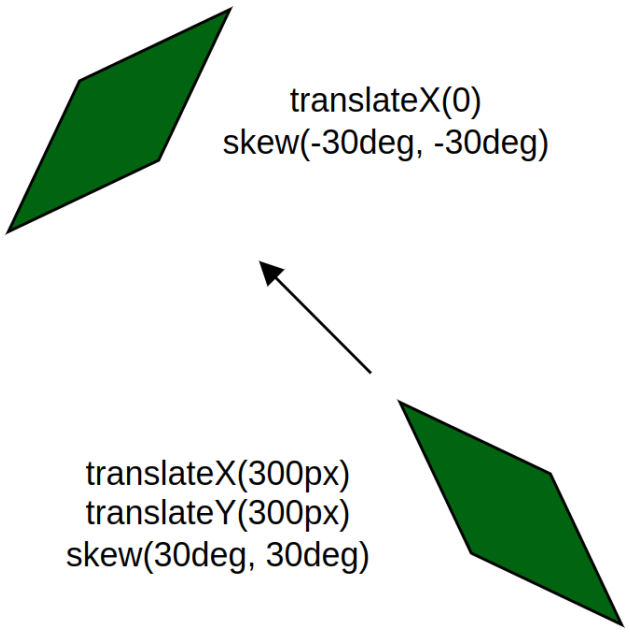

Figure 5.6 The second stage of animation 5

The Listings 5.14-15 present the code executed for animation 5. The elements are assigned size and the gap between them. The elements start with no translation and skew. Then, they are moved on both axes by 300 pixels and concurrently skewed by -30 degrees Then, the elements return to the previous translation and are skewed to 30 degrees.

Listing 5.14 The first part of the code specific to animation 5

```
--box-width-anim-5: 22px;
--box-height-anim-5: 22px;
--box-gap-anim-5: 13px;
```

Listing 5.15 The second part of the code specific to animation 5

```
@keyframes animation5 {
  0% {
    transform: translateX(0) skew(0deg, 0deg);
    background-color: var(--color-box);
  }
  50% {
    transform: translateX(300px) translateY(300px) skew(30deg, 30deg);
  }
  100% {
    transform: translateX(0) skew(-30deg, -30deg);
    background-color: var(--color-box);
  }
}
```

5.7 Animation 6

The sixth animation is a combination of translation, rotation and scaling (Fig. 5.7).

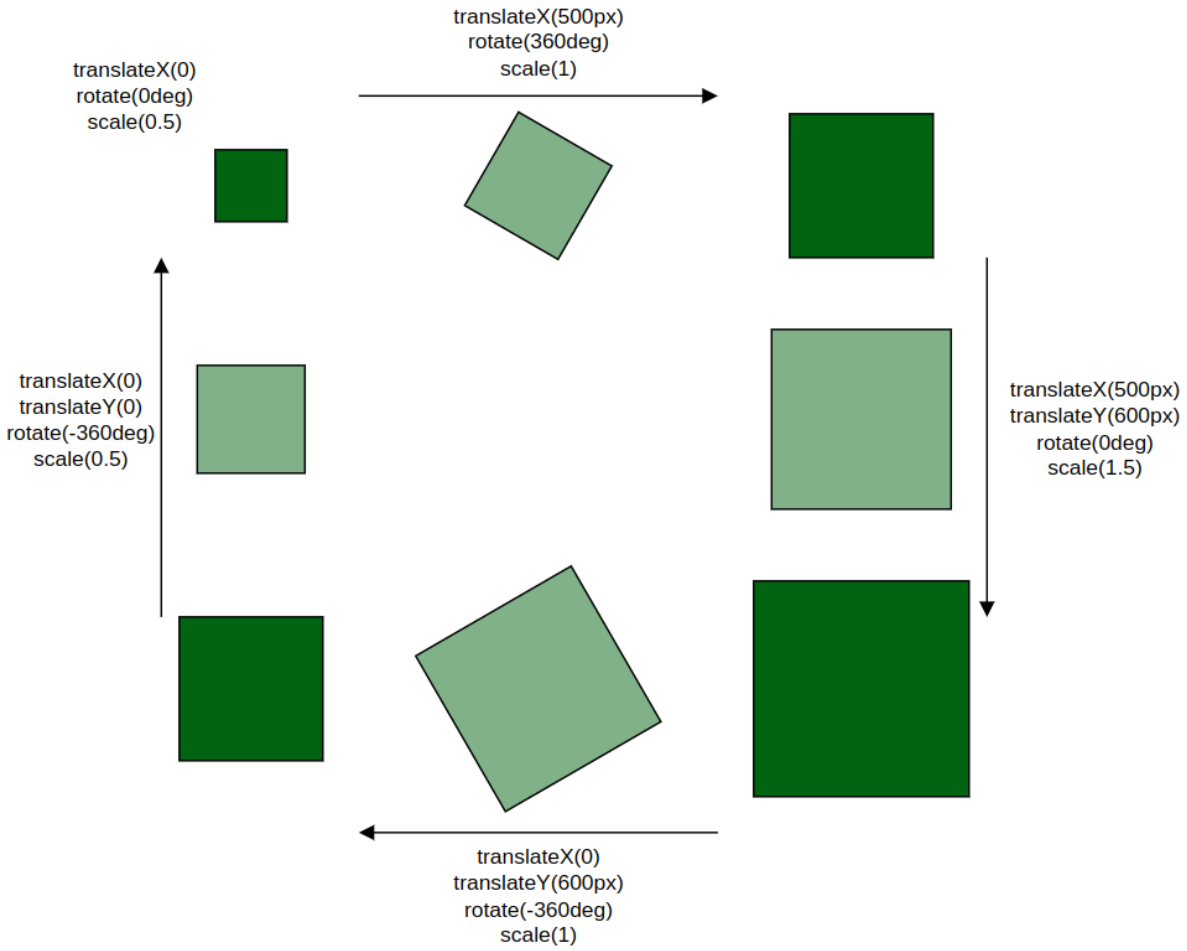

Figure 5.7 Depiction of animation 6

The Listings 5.16-17 present the code executed for animation 6. The elements are assigned size and the gap between them. The elements start with no translation, no rotation and a scale of 0.5. Then, they are concurrently moved on the X-axis by 500 pixels, rotated by 360 degrees and scaled up to 1. Next, the elements are simultaneously translated on the Y-axis by 600 pixels and scaled up to 1.5. The third phase contains concurrent translation on the X-axis to 0, rotated by -360 degrees (in the opposite direction) and scaled down to 1. Lastly, the elements are moved on the Y-axis to 0 (the start position) and scaled down to 0.5.

Listing 5.16 The first part of the code specific to animation 6

```
--box-width-anim-6: 15px;
--box-height-anim-6: 15px;
--box-gap-anim-6: 5px;
```

Listing 5.17 The second part of the code specific to animation 6

```
@keyframes animation6 {
  0% {
    transform: translateX(0) rotate(0deg) scale(0.5);
    background-color: var(--color-box);
  }
  20% {
    transform: translateX(500px) rotate(360deg) scale(1);
  }
  50% {
    transform: translateX(500px) translateY(600px) rotate(0deg) scale(1.5);
  }
  75% {
    transform: translateX(0) translateY(600px) rotate(-360deg) scale(1);
  }
  100% {
    transform: translateX(0) translateY(0) rotate(-360deg) scale(0.5);
    background-color: var(--color-box);
  }
}
```

5.8 Animation 7

The seventh and eighth animations aim to test the performance of animating the "width" CSS property when setting it by pixels versus percentage (Fig. 5.8). This animation sets the width by pixels.

Figure 5.8 Depiction of animation 7

The Listings 5.18-19 present the code executed for animation 7. The elements are assigned size and the gap between them. During the animation, the width of an element is increased from 9.6 pixels to 57.6 pixels. These numbers correspond to the width of 1 percent and 3 percent respectively. These numbers are specific to the resolution of the monitor as specified in 4.3.1.

Listing 5.18 The first part of the code specific to animation 7

```
/*equal to 1% or 3% of width for animation 8*/
--box-width-anim-7: 9.6px;
--box-height-anim-7: 10px;
--box-gap-anim-7: 15px;
```

Listing 5.19 The second part of the code specific to animation 7

```
@keyframes animation7 {
  0% {
    background-color: var(--color-box);
  }
  100% {
    /*equal to 1% or 3% of width for animation 8*/
    width: 57.6px;
    background-color: var(--color-box);
  }
}
```

5.9 Animation 8

The seventh and eighth animations aim to test the performance of animating the "width" CSS property when setting it by pixels versus percentage (Fig. 5.9). This animation sets the width by percentage.

Figure 5.9 Depiction of animation 8

The Listings 5.20-21 present the code executed for animation 8. The elements are assigned size and the gap between them. During the animation, the width of an element is increased from 1% to 3%.

Listing 5.20 The first part of the code specific to animation 8

```
--box-width-anim-8: 1%;
--box-height-anim-8: 10px;
--box-gap-anim-8: 15px;
```

Listing 5.21 The second part of the code specific to animation 8

```
@keyframes animation8 {
  0% {
    background-color: var(--color-box);
  }
  100% {
    width: 3%;
    background-color: var(--color-box);
  }
}
```

5.10 Animation 9

The ninth animation is an animated gradient (Fig. 5.10).

linear-gradient(130deg,
#650000, #ffffff, #053d3b, [...])

Figure 5.10 Depiction of animation 9

The Listings 5.22 – 5.24 present the code executed for animation 9. The elements are assigned size and the gap between them. During the animation, the gradient set on the background changes. Additionally, the background-size and position are set in order to allow the gradient to be animated.

Listing 5.22 The first part of the code specific to animation 9

```
--box-width-anim-9: 50px;
--box-height-anim-9: 50px;
--box-gap-anim-9: 5px;
```

Listing 5.23 The second part of the code specific to animation 9

```
@keyframes blackBackgroundAnim8 {
  0% {
    background-size: auto auto;
    background: var(--color-black);
  }
  100% {
    background-size: auto auto;
    background: var(--color-black);
  }
}
```

Listing 5.24 The third part of the code specific to animation 9

```
@keyframes animation9 {
  0% {
    background: linear-gradient(130deg, #650000, #ffffff,
                        #053d3b, #c7b8ee, #700769, #ffffff,
                        #053d3b, #c7b8ee, #700769, #ffffff,
                        #053d3b, #c7b8ee, #700769);
    background-size: 200% 200%;
    background-position: 0 0
  }
  100% {
    background: linear-gradient(130deg, #650000, #ffffff,
                        #053d3b, #c7b8ee, #700769, #ffffff,
                        #053d3b, #c7b8ee, #700769, #ffffff,
                        #053d3b, #c7b8ee, #700769);
    background-size: 200% 200%;
    background-position: 100% 100%
  }
}
```

5.11 Animation 10

The tenth animation is a 3D cube translated and rotated (Fig. 5.11).

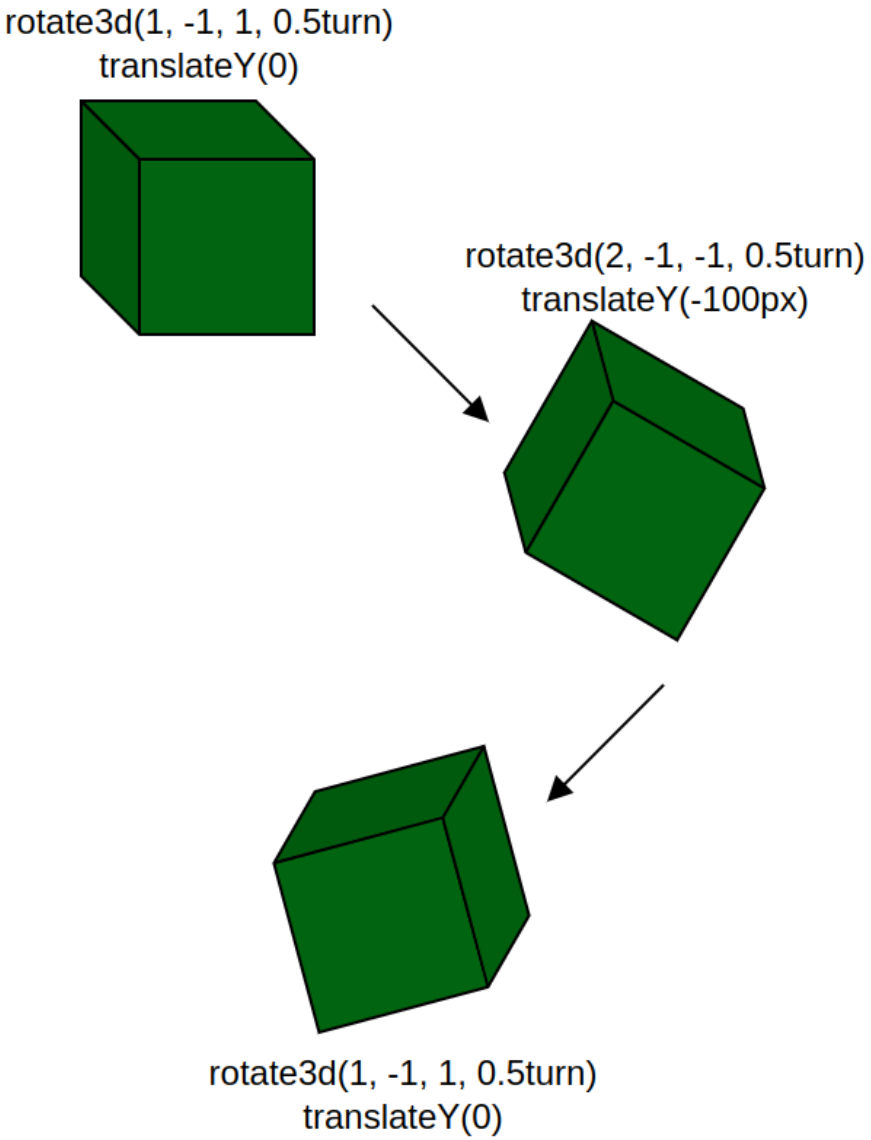

Figure 5.11 Depiction of animation 10

The Listings 5.25 – 5.27 present the code executed for animation 10. The elements are assigned size and the gap between them.

Listing 5.25 The first part of the code specific to animation 10

```
--box-width-anim-10: 22px;
--box-height-anim-10: 22px;
--box-gap-anim-10: 16px;
```

The following listing is a setup of a 3D cube. The HTML code for a cube is generated by JavaScript and presented in Subsection 5.1. Firstly, the "#app-wrapper" and all elements with class "box" are assigned "transform-style" of "preserve-3d" in order to position the children of those elements in their own 3D space. Then, all faces of a cube are given position "absolute", their width and height, background colour and animation that gives each face a "box-shadow" to enhance the 3D effect. Lastly, each face is separately positioned to form a cube. The animation that moves a cube does it by using "rotate3d" (rotation in 3D space) and a translation on a Y-axis.

Listing 5.26 The second part of the code specific to animation 10

```
#app-wrapper, .box {
  transform-style: preserve-3d;
}

.box .left, .box .right, .box .front, .box .back, .box .top, .box .bottom {
  position: absolute;
  width: var(--box-width-anim-10);
  height: var(--box-width-anim-10);
  background-color: var(--color-black);
  animation: animation10CubeSideColor var(--anim-duration) linear var(--anim-delay);
}

.box .front {transform: translateZ(calc(var(--box-width-anim-10) / 2));}
.box .right {transform: rotateY(90deg) translateZ(calc(var(--box-width-anim-10) / 2));}
.box .back {transform: rotateY(180deg) translateZ(calc(var(--box-width-anim-10) / 2));}
.box .left {transform: rotateY(270deg) translateZ(calc(var(--box-width-anim-10) / 2));}
.box .top {transform: translateY(-50%) rotateX(90deg);}
.box .bottom {transform: translateY(50%) rotateX(90deg); bottom: 0;}
```

Listing 5.27 The third part of the code specific to animation 10

```
@keyframes animation10 {
  0% {
    transform: rotate3d(1, -1, 1, 0.5turn) translateY(0);
    background-color: var(--box-width-anim-10);
  }
  50% {
    transform: rotate3d(2, -1, -1, 0.5turn) translateY(-100px);
  }
  100% {
    transform: rotate3d(1, -1, 1, 0.5turn) translateY(0);
    background-color: var(--box-width-anim-10);
  }
}

@keyframes animation10CubeSideColor {
  0% {
    background-color: var(--color-box-anim-10);
    box-shadow: 0 0 0.5rem #000a inset;
  }
  100% {
    background-color: var(--color-box-anim-10);
    box-shadow: 0 0 0.5rem #000a inset;
  }
}
```

6. Results

The following section presents the results of measuring the performance of the animations described in Section 5. Graphs aim to provide a visual understanding of how each browser performed for each animation based on the growing number of elements. The tables contain precise data for each browser along with calculated standard deviation.

6.1 Animation 1

Below presented is the performance graph for animation 1 for each tested browser (Fig. 6.1). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

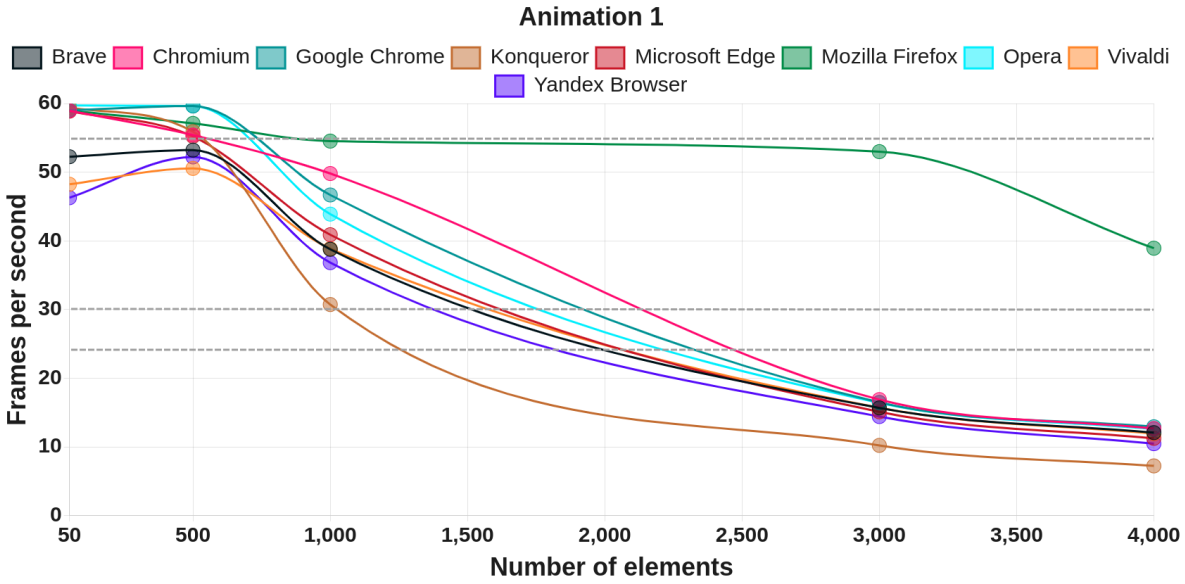

Figure 6.1 Performance graph of animation 1

Table 6.1 presents the average fps for each threshold for animation 1 for each tested browser.

	Number of elements Average fps (seconds)				
	50	500	1000	3000	4000
Brave	52.27 ± 1.14	53.23 ± 1.03	38.79 ± 0.70	15.69 ± 0.49	12.09 ± 0.66
Chromium	58.98 ± 0.58	55.42 ± 1.66	49.82 ± 1.73	16.90 ± 0.70	12.68 ± 0.89
Google Chrome	59.03 ± 0.63	59.66 ± 0.41	46.70 ± 0.88	16.50 ± 0.61	12.94 ± 0.55
Konqueror	59.40 ± 0.49	55.92 ± 0.93	30.75 ± 0.36	10.23 ± 0.19	7.24 ± 0.23
Microsoft Edge	58.90 ± 0.73	55.23 ± 4.46	40.91 ± 0.96	15.12 ± 0.50	11.26 ± 0.46
Mozilla Firefox	59.00 ± 0.71	57.12 ± 1.07	**54.55** ± 1.65	**53.01** ± 1.53	**38.94** ± 0.90
Opera	**59.75** ± 0.27	**59.67** ± 0.56	43.91 ± 0.68	16.42 ± 0.78	12.60 ± 0.50
Vivaldi	48.26 ± 3.14	50.57 ± 1.20	38.88 ± 0.81	15.71 ± 0.54	11.93 ± 0.40
Yandex Browser	46.30 ± 4.48	52.23 ± 3.13	36.84 ± 0.82	14.43 ± 0.66	10.48 ± 0.48

Table 6.1 Results table for animation 1

6.2 Animation 2

Below presented is the performance graph for animation 2 for each tested browser (Fig. 6.2). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

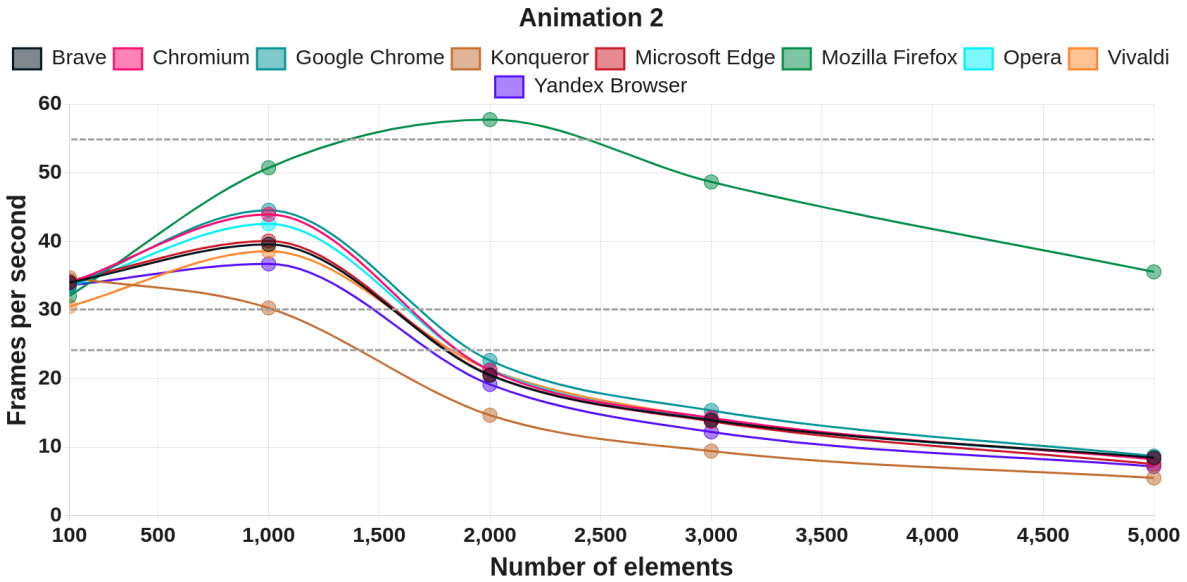

Figure 6.2 Performance graph of animation 2

Table 6.2 presents the average fps for each threshold for animation 2 for each tested browser.

	Number of elements Average fps (seconds)				
	100	1000	2000	3000	5000
Brave	34.03 ± 1.13	39.59 ± 1.81	20.50 ± 1.03	13.93 ± 0.62	8.47 ± 0.40
Chromium	33.98 ± 1.48	43.97 ± 1.55	21.21 ± 0.95	14.28 ± 0.67	8.23 ± 0.60
Google Chrome	33.47 ± 1.39	44.57 ± 1.78	22.66 ± 0.94	15.36 ± 0.85	8.70 ± 0.42
Konqueror	**34.73** ± 1.28	30.31 ± 1.29	14.67 ± 0.61	9.43 ± 0.38	5.51 ± 0.35
Microsoft Edge	34.30 ± 1.03	40.09 ± 1.51	20.56 ± 0.79	13.78 ± 0.66	7.53 ± 0.39
Mozilla Firefox	32.07 ± 0.70	**50.77** ± 2.12	**57.79** ± 1.21	**48.70** ± 1.63	**35.58** ± 2.04
Opera	33.45 ± 1.30	42.59 ± 1.25	21.32 ± 0.71	14.12 ± 0.71	8.31 ± 0.52
Vivaldi	30.54 ± 1.18	38.60 ± 1.38	21.30 ± 0.63	14.22 ± 0.62	8.37 ± 0.52
Yandex Browser	33.61 ± 1.13	36.74 ± 0.95	19.16 ± 0.67	12.23 ± 0.46	7.19 ± 0.38

Table 6.2 Results table for animation 2

6.3 Animation 3

Below presented is the performance graph for animation 3 for each tested browser (Fig. 6.3). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

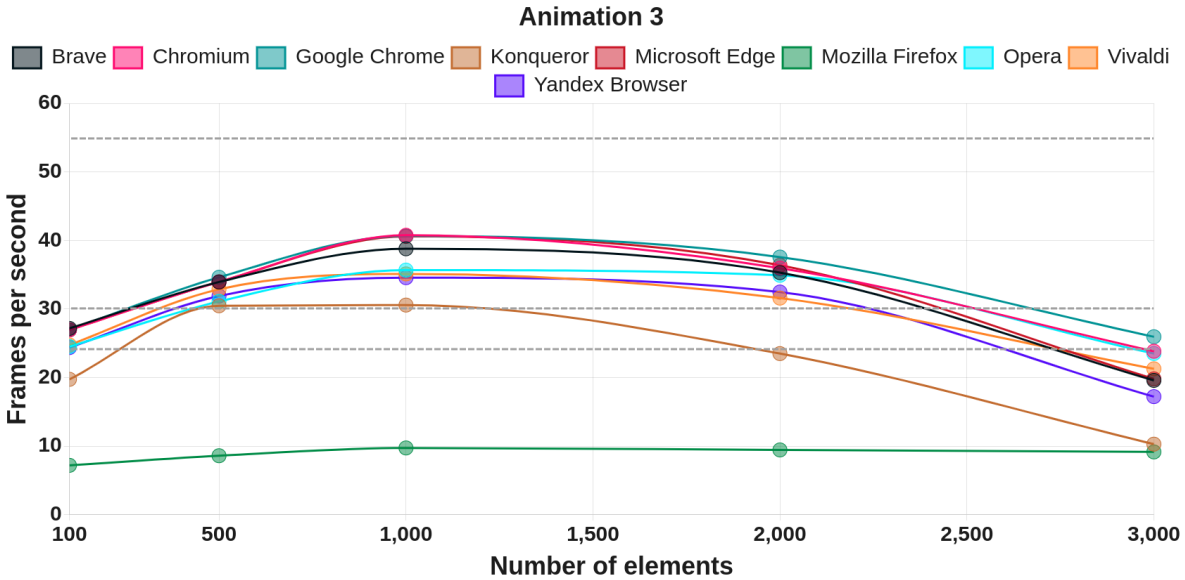

Figure 6.3 Performance graph of animation 3

Table 6.3 presents the average fps for each threshold for animation 3 for each tested browser.

	Number of elements Average fps (seconds)				
	100	500	1000	2000	3000
Brave	**27.19** ± 0.79	33.96 ± 0.63	38.80 ± 1.34	35.32 ± 1.63	19.60 ± 1.30
Chromium	26.98 ± 0.99	33.97 ± 1.03	**40.78** ± 1.21	35.98 ± 1.32	23.85 ± 1.91
Google Chrome	26.99 ± 0.65	**34.64** ± 0.58	40.68 ± 1.34	**37.59** ± 1.25	**25.97** ± 1.39
Konqueror	19.76 ± 0.48	30.49 ± 1.09	30.59 ± 0.95	23.51 ± 1.66	10.30 ± 0.40
Microsoft Edge	26.97 ± 0.82	33.98 ± 0.65	40.61 ± 1.06	36.40 ± 1.14	19.82 ± 1.19
Mozilla Firefox	7.21 ± 0.30	8.61 ± 1.35	9.74 ± 1.57	9.46 ± 1.01	9.17 ± 0.51
Opera	24.52 ± 0.88	31.12 ± 0.89	35.69 ± 1.66	34.93 ± 2.49	23.52 ± 1.24
Vivaldi	24.77 ± 0.86	32.89 ± 1.00	35.13 ± 1.28	31.61 ± 1.61	21.29 ± 1.40
Yandex Browser	24.39 ± 0.80	31.91 ± 1.08	34.59 ± 1.20	32.49 ± 1.23	17.23 ± 1.00

Table 6.3 Results table for animation 3

6.4 Animation 4

Below presented is the performance graph for animation 4 for each tested browser (Fig. 6.4). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

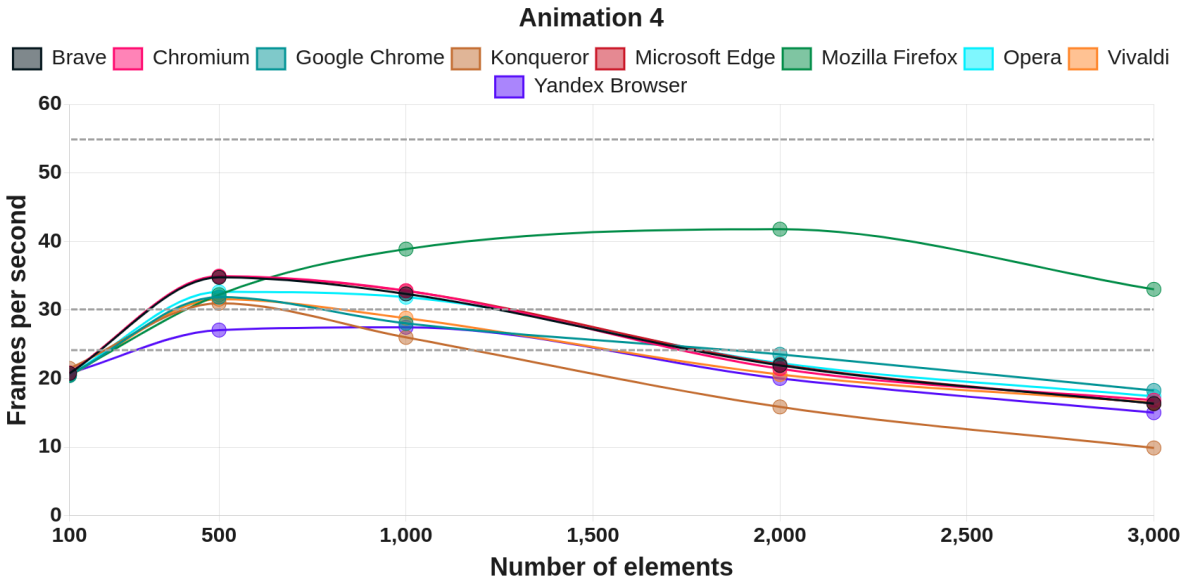

Figure 6.4 Performance graph of animation 4

Table 6.4 presents the average fps for each threshold for animation 4 for each tested browser.

	Number of elements Average fps (seconds)				
	100	500	1000	2000	3000
Brave	20.77 ± 0.49	34.78 ± 0.57	32.37 ± 1.05	21.93 ± 0.82	16.35 ± 1.23
Chromium	20.81 ± 0.39	**34.96** ± 0.19	32.83 ± 1.14	21.45 ± 0.69	16.86 ± 0.98
Google Chrome	20.44 ± 0.50	31.91 ± 1.05	28.07 ± 1.19	23.54 ± 0.94	18.28 ± 1.83
Konqueror	**21.49** ± 0.47	31.00 ± 0.58	26.02 ± 0.93	15.88 ± 0.71	9.89 ± 0.70
Microsoft Edge	20.83 ± 0.45	34.90 ± 0.33	32.81 ± 1.11	22.07 ± 0.97	16.36 ± 1.51
Mozilla Firefox	20.57 ± 0.45	32.24 ± 0.90	**38.90** ± 1.03	**41.80** ± 0.85	**33.03** ± 1.33
Opera	20.55 ± 0.41	32.66 ± 0.98	31.89 ± 0.96	22.27 ± 0.91	17.42 ± 1.20
Vivaldi	20.45 ± 0.32	31.59 ± 0.71	28.82 ± 1.24	20.61 ± 0.94	16.50 ± 1.00
Yandex Browser	20.79 ± 0.62	27.09 ± 1.58	27.50 ± 1.29	20.03 ± 0.80	15.04 ± 1.06

Table 6.4 Results table for animation 4

6.5 Animation 5

Below presented is the performance graph for animation 5 for each tested browser (Fig. 6.5). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

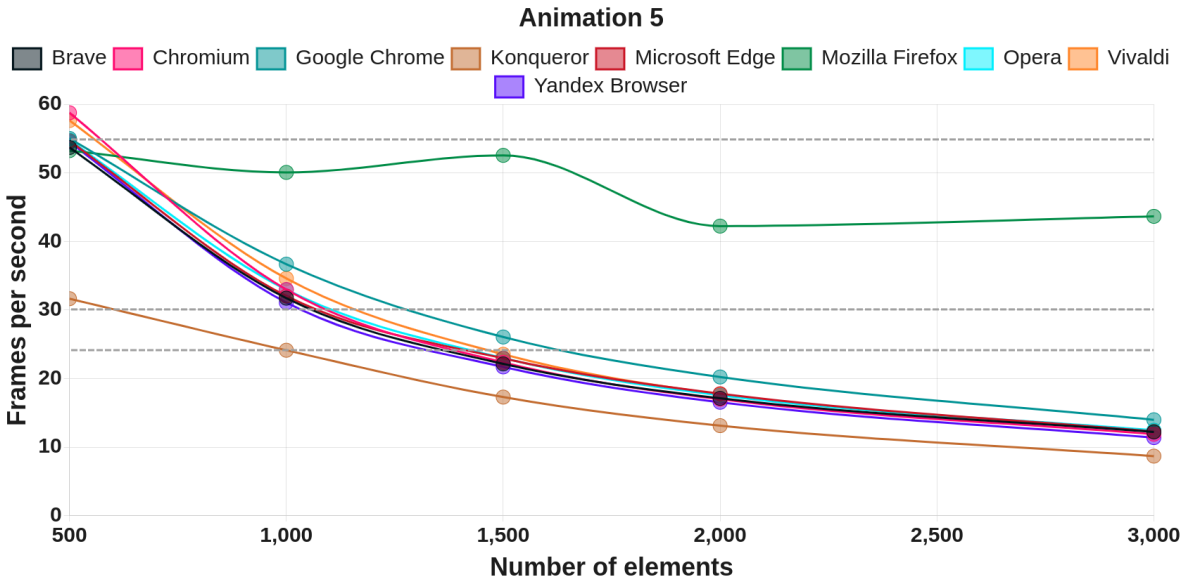

Figure 6.5 Performance graph of animation 5

Table 6.5 presents the average fps for each threshold for animation 5 for each tested browser.

	Number of elements Average fps (seconds)				
	500	1000	1500	2000	3000
Brave	53.72 ± 0.98	31.74 ± 0.73	22.16 ± 0.48	17.13 ± 0.47	12.22 ± 0.46
Chromium	**58.79** ± 0.55	33.00 ± 0.63	22.38 ± 0.61	17.00 ± 0.52	11.90 ± 0.51
Google Chrome	55.05 ± 0.88	36.70 ± 1.01	26.08 ± 0.73	20.25 ± 0.43	14.00 ± 0.55
Konqueror	31.67 ± 1.23	24.14 ± 0.58	17.31 ± 0.34	13.14 ± 0.35	8.70 ± 0.28
Microsoft Edge	54.70 ± 0.82	32.06 ± 0.71	22.97 ± 0.64	17.80 ± 0.47	12.28 ± 0.42
Mozilla Firefox	53.27 ± 1.60	**50.09** ± 1.48	**52.58** ± 1.38	**42.26** ± 1.30	**43.68** ± 1.41
Opera	54.80 ± 0.85	32.98 ± 0.68	23.00 ± 0.43	17.57 ± 0.52	12.48 ± 0.60
Vivaldi	57.66 ± 0.81	34.62 ± 0.83	23.56 ± 0.65	17.72 ± 0.49	12.34 ± 0.38
Yandex Browser	54.51 ± 1.40	31.10 ± 0.60	21.71 ± 0.46	16.56 ± 0.49	11.40 ± 0.50

Table 6.5 Results table for animation 5

6.6 Animation 6

Below presented is the performance graph for animation 6 for each tested browser (Fig. 6.6). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

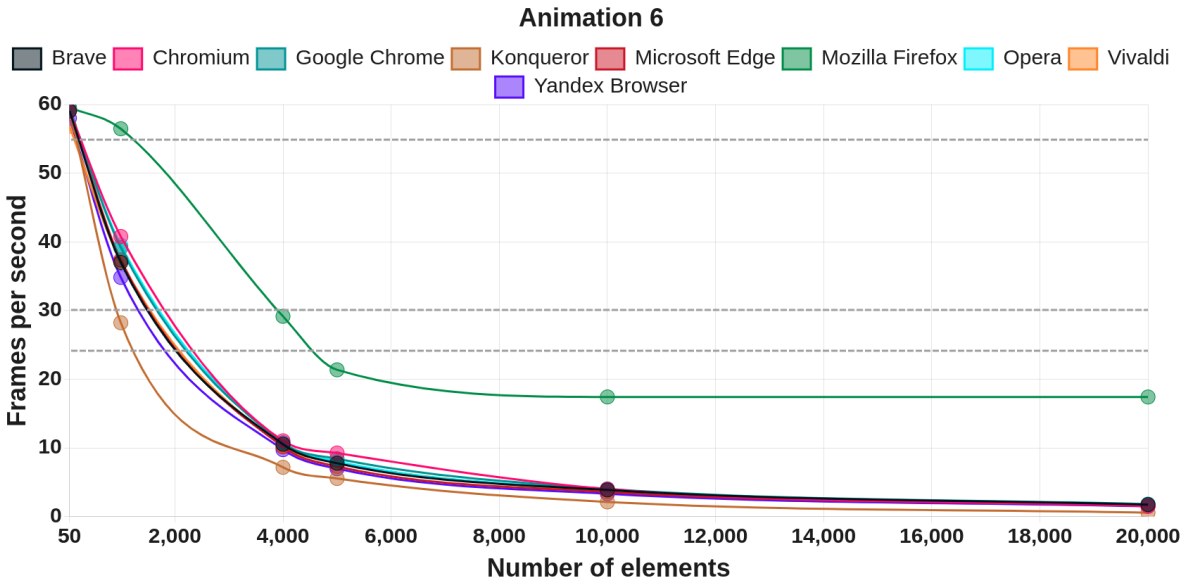

Figure 6.6 Performance graph of animation 6

Table 6.6 presents the average fps for each threshold for animation 6 for each tested browser.

	Number of elements Average fps (seconds)					
	50	1000	4000	5000	10000	20000
Brave	59.09 ± 0.40	37.01 ± 0.65	10.54 ± 0.39	7.78 ± 0.27	3.89 ± 0.16	1.75 ± 0.12
Chromium	59.06 ± 0.35	40.80 ± 0.91	11.05 ± 0.59	9.27 ± 0.41	4.02 ± 0.11	1.55 ± 0.15
Google Chrome	59.46 ± 0.40	39.16 ± 1.11	10.75 ± 0.30	8.43 ± 0.34	4.04 ± 0.19	1.82 ± 0.09
Konqueror	59.45 ± 0.43	28.23 ± 0.46	7.19 ± 0.15	5.56 ± 0.11	2.14 ± 0.11	0.58 ± 0.06
Microsoft Edge	59.44 ± 0.35	37.37 ± 0.66	10.15 ± 0.39	7.31 ± 0.14	3.59 ± 0.22	1.57 ± 0.12
Mozilla Firefox	**59.57** ± 0.17	**56.49** ± 1.31	**29.17** ± 0.51	**21.38** ± 0.65	**17.43** ± 0.94	**17.43** ± 1.76
Opera	59.19 ± 0.43	39.51 ± 0.49	10.77 ± 0.37	8.03 ± 0.26	3.89 ± 0.20	1.69 ± 0.11
Vivaldi	56.85 ± 0.33	37.21 ± 0.60	10.35 ± 0.33	7.93 ± 0.28	3.73 ± 0.13	1.67 ± 0.09
Yandex Browser	58.03 ± 1.02	34.84 ± 0.53	9.75 ± 0.37	6.99 ± 0.21	3.33 ± 0.13	1.49 ± 0.14

Table 6.6 Results table for animation 6

6.7 Animation 7

Below presented is the performance graph for animation 7 for each tested browser (Fig. 6.7). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

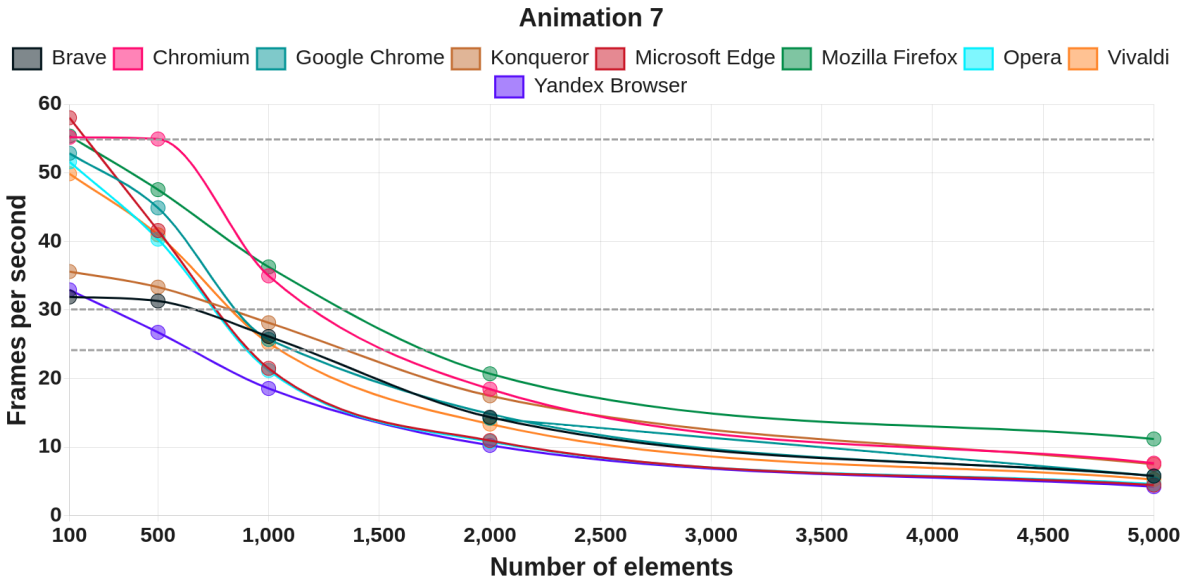

Figure 6.7 Performance graph of animation 7

Table 6.7 presents the average fps for each threshold for animation 7 for each tested browser.

	Number of elements Average fps (seconds)				
	100	500	1000	2000	5000
Brave	31.91 ± 1.89	31.33 ± 1.03	26.16 ± 1.37	14.38 ± 0.24	5.79 ± 0.16
Chromium	55.20 ± 0.60	**54.97** ± 0.79	35.00 ± 0.76	18.48 ± 0.58	7.67 ± 0.40
Google Chrome	52.88 ± 0.93	44.92 ± 1.15	25.72 ± 0.43	14.22 ± 0.26	5.82 ± 0.22
Konqueror	35.62 ± 2.92	33.36 ± 1.41	28.16 ± 0.88	17.50 ± 0.36	7.50 ± 0.21
Microsoft Edge	**58.07** ± 1.17	41.61 ± 0.43	21.50 ± 0.23	10.97 ± 0.22	4.48 ± 0.19
Mozilla Firefox	55.40 ± 1.45	47.57 ± 1.91	**36.30** ± 0.70	**20.71** ± 0.48	**11.20** ± 0.46
Opera	51.63 ± 1.22	40.34 ± 0.71	21.19 ± 0.88	10.81 ± 0.24	4.64 ± 0.18
Vivaldi	49.89 ± 1.46	40.99 ± 2.03	25.21 ± 0.79	13.38 ± 0.32	5.29 ± 0.19
Yandex Browser	32.96 ± 1.89	26.76 ± 1.46	18.57 ± 0.38	10.27 ± 0.28	4.24 ± 0.21

Table 6.7 Results table for animation 7

6.8 Animation 8

Below presented is the performance graph for animation 8 for each tested browser (Fig. 6.8). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

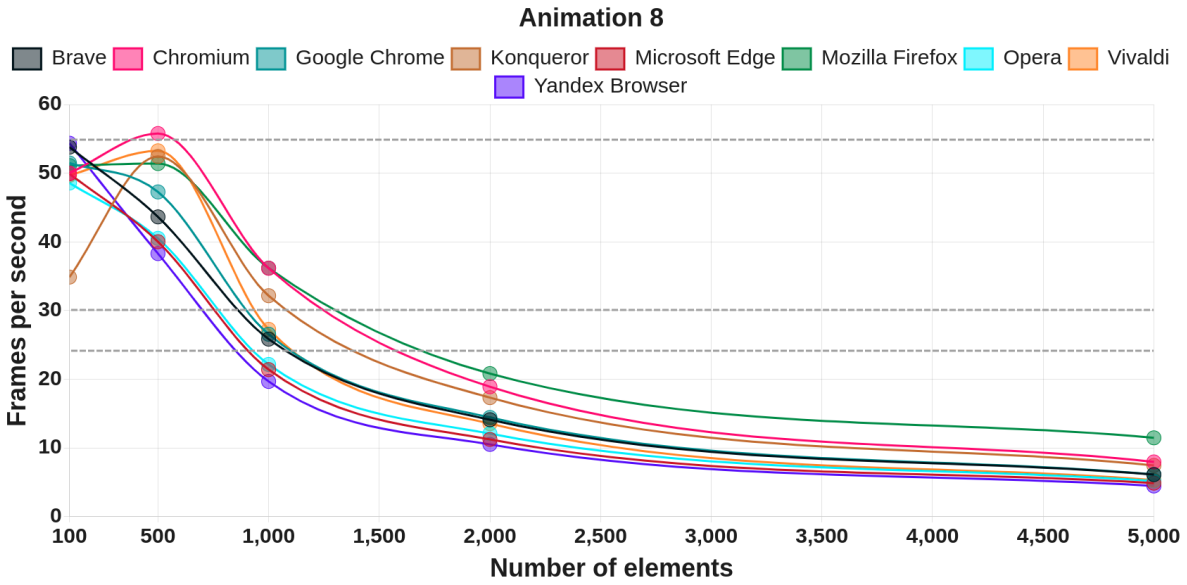

Figure 6.8 Performance graph of animation 8

Table 6.8 presents the average fps for each threshold for animation 8 for each tested browser.

	Number of elements Average fps (seconds)				
	100	500	1000	2000	5000
Brave	53.81 ± 2.01	43.66 ± 1.71	25.86 ± 0.44	14.11 ± 0.27	6.12 ± 0.16
Chromium	50.03 ± 1.78	**55.79** ± 0.61	36.18 ± 0.68	18.92 ± 0.32	7.97 ± 0.18
Google Chrome	51.50 ± 1.57	47.29 ± 1.05	26.57 ± 0.36	14.44 ± 0.35	6.15 ± 0.16
Konqueror	34.89 ± 2.40	52.46 ± 1.13	32.18 ± 0.51	17.33 ± 0.36	7.46 ± 0.19
Microsoft Edge	49.92 ± 1.85	40.03 ± 0.53	21.41 ± 0.22	11.24 ± 0.23	4.88 ± 0.22
Mozilla Firefox	51.12 ± 2.19	51.42 ± 1.78	**36.19** ± 0.62	**20.83** ± 0.45	**11.48** ± 0.52
Opera	48.59 ± 2.74	40.52 ± 0.59	22.17 ± 0.31	12.09 ± 0.17	5.18 ± 0.16
Vivaldi	49.70 ± 1.44	53.28 ± 1.04	27.28 ± 0.31	13.54 ± 0.31	5.31 ± 0.15
Yandex Browser	**54.38** ± 1.47	38.31 ± 1.30	19.69 ± 0.24	10.52 ± 0.21	4.47 ± 0.17

Table 6.8 Results table for animation 8

6.9 Animation 9

Below presented is the performance graph for animation 9 for each tested browser (Fig. 6.9). On the X-axis is the number of elements and on the Y-axis number

of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

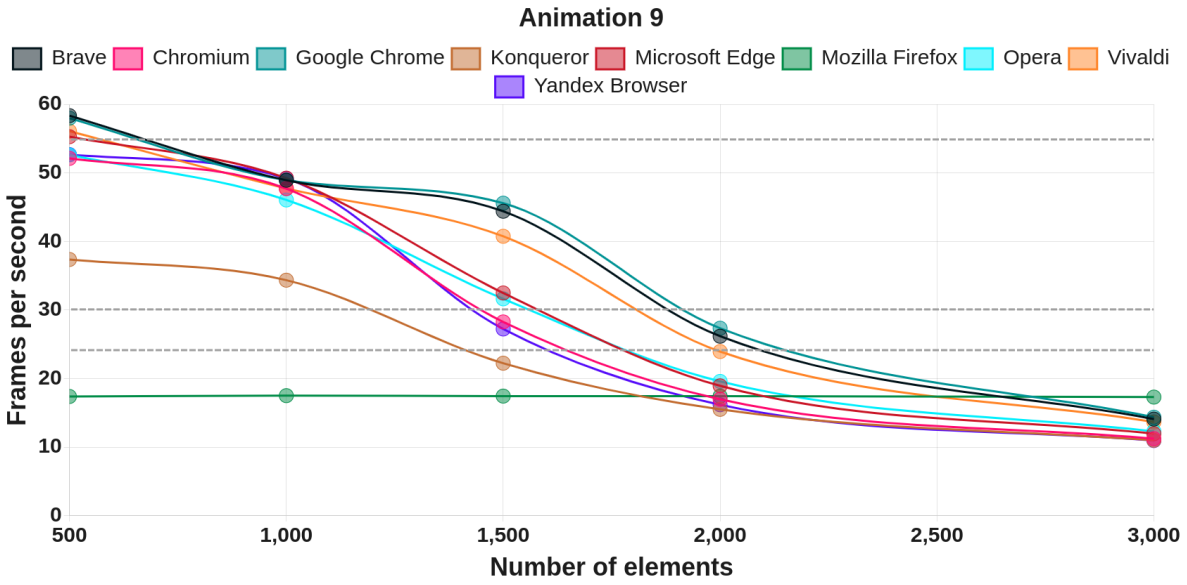

Figure 6.9 Performance graph of animation 9

Table 6.9 presents the average fps for each threshold for animation 9 for each tested browser.

	Number of elements Average fps (seconds)				
	500	1000	1500	2000	3000
Brave	**58.37** ± 0.79	48.95 ± 1.51	44.42 ± 1.97	26.19 ± 1.44	14.10 ± 0.29
Chromium	52.13 ± 1.65	47.72 ± 1.78	28.28 ± 1.65	16.99 ± 0.72	11.24 ± 0.21
Google Chrome	58.00 ± 0.83	48.97 ± 1.47	**45.60** ± 1.95	**27.36** ± 2.00	14.38 ± 0.38
Konqueror	37.39 ± 1.96	34.37 ± 1.66	22.26 ± 1.15	15.52 ± 0.37	11.01 ± 0.18
Microsoft Edge	55.30 ± 1.16	**49.26** ± 1.98	32.50 ± 1.75	18.96 ± 0.56	11.97 ± 0.25
Mozilla Firefox	17.40 ± 0.28	17.52 ± 0.33	17.45 ± 0.38	17.44 ± 0.38	**17.31** ± 0.31
Opera	52.64 ± 1.52	46.07 ± 1.69	31.64 ± 1.61	19.60 ± 0.61	12.31 ± 0.23
Vivaldi	56.11 ± 1.16	47.76 ± 1.86	40.77 ± 1.70	23.96 ± 1.41	13.68 ± 0.33
Yandex Browser	52.69 ± 1.13	49.21 ± 1.77	27.24 ± 1.43	16.19 ± 0.27	10.97 ± 0.16

Table 6.9 Results table for animation 9

6.10 Animation 10

Below presented is the performance graph for animation 10 for each tested browser (Fig. 6.10). On the X-axis is the number of elements and on the Y-axis number of frames per second. Each point on the graph represents a measured value that is frames per second for a threshold.

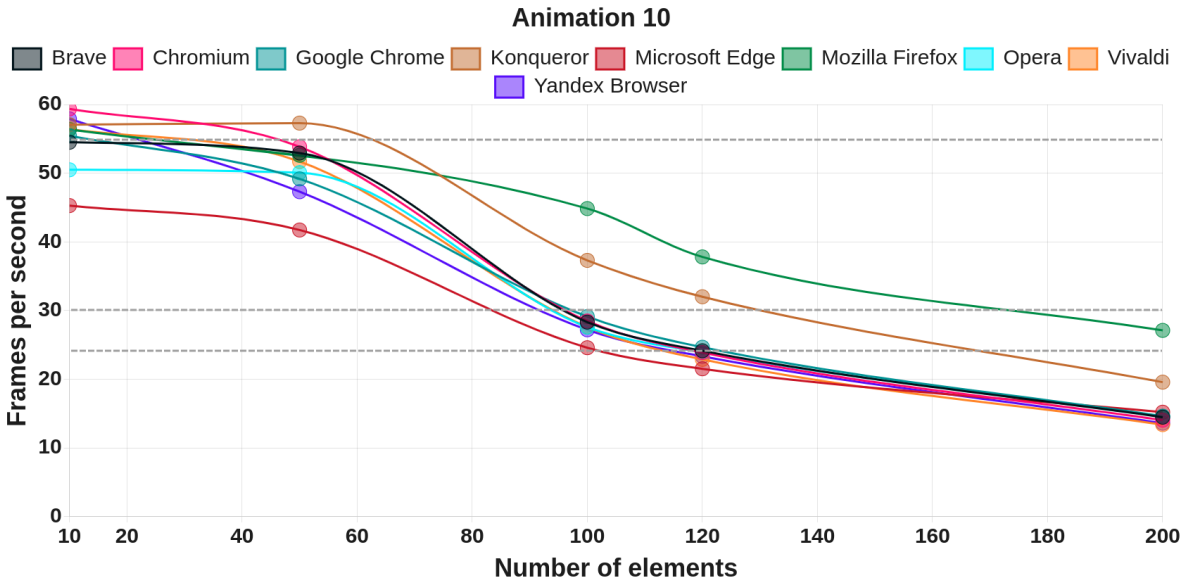

Figure 6.10 Performance graph of animation 10

Table 6.10 presents the average fps for each threshold for animation 10 for each tested browser.

	Number of elements Average fps (seconds)				
	10	50	100	120	200
Brave	54.52 ± 1.53	52.96 ± 1.27	28.33 ± 0.58	24.16 ± 0.45	14.49 ± 0.37
Chromium	**59.38** ± 0.58	53.87 ± 1.73	28.49 ± 0.48	23.90 ± 0.47	14.06 ± 0.30
Google Chrome	55.47 ± 1.31	49.19 ± 1.46	29.13 ± 0.55	24.67 ± 0.46	14.64 ± 0.28
Konqueror	57.08 ± 1.50	**57.31** ± 0.91	37.32 ± 0.92	32.02 ± 0.43	19.59 ± 0.33
Microsoft Edge	45.30 ± 3.03	41.74 ± 2.19	24.60 ± 1.10	21.53 ± 1.59	15.21 ± 1.12
Mozilla Firefox	56.39 ± 1.31	52.62 ± 2.71	**44.87** ± 1.18	**37.82** ± 0.94	**27.12** ± 0.62
Opera	50.51 ± 1.40	50.09 ± 1.26	27.60 ± 0.54	23.93 ± 0.35	14.53 ± 0.37
Vivaldi	56.43 ± 0.89	51.72 ± 1.07	27.61 ± 0.48	22.90 ± 0.29	13.40 ± 0.22
Yandex Browser	57.92 ± 1.06	47.32 ± 1.99	27.21 ± 1.15	23.35 ± 0.85	13.61 ± 1.04

Table 6.10 Results table for animation 10

7. Discussion

Three levels of animation smoothness (fps) have been assumed:

- **Level 1** for fps ≥ 55,
- **Level 2** for fps between 30 and 54,
- **Level 3** for fps between 24 and 29.

The first level means an ideal animation performance, the second level below which some people claim the individual frames of an animation can be distinguished, that is the animation is no longer fully smooth and the last level of 24 fps, [100] below which the animation is unanimously lagging. Based on the results presented in Section 6 the following observations can be deduced:

Animation 1: Six out of nine browsers started in Level 1 and fell to Level 2 between 500 and 1000 rendered elements (Fig. 6.1). The remaining three browsers started in Level 2 but close to Level 1. All browsers but Mozilla Firefox fell to Level 3 for 1000 to 2200 elements and below Level 3, that is – started visibly lagging after adding around 400 elements more to each browser. Mozilla Firefox came out to be the most performant, staying in Level 2 even for the largest number of elements. Opera, Google Chrome and Chromium performed the best after Mozilla Firefox and Konqueror performed visibly worse than other browsers. The remaining browsers attained similar performance which is expected as most of the browsers tested are based on the Blink rendering engine.

Animation 2: Comparable results with animation 1 were obtained for animation 2 with respect to the order of the most performant browsers. All browsers started in Level 2 attaining 30 to 35 fps (Fig. 6.2). Then, the fps of all browsers except Mozilla Firefox and Konqueror grew between 35 to 45 fps, peaking at 1000 rendered elements. The browsers fell to Level 3 between 1500 and 1700 elements and below it between 1700 and 1900 elements. Konqueror presented a steady decrease in fps reaching Level 1 for 1000 elements and falling below it for 1400 elements. Mozilla Firefox reached Level 1 for 1400 elements, peaked at 2000 elements, then, fell back to Level 2 at 2500 elements and stayed in Level 2 even for the largest number of elements.

Animation 3: Mozilla Firefox performed the worst maintaining around 9 fps throughout each threshold (Fig. 6.3). This surprising result is discussed later. Konqueror scored second worst and started below Level 3 with 20 fps, then grew to the verge of Level 2 and 3 between 450 and 1250 elements and started losing fps reaching below Level 3 for 1900 elements. Other browsers performed similarly with Google Chrome, Brave and Chromium at the top. The browsers started in Level 3 with 24 to 27 fps, then grew to Level 2 between 200 and 400 elements. They peaked for 1000 elements, fell to Level 3 between 2200 and 2700 elements and then below Level 3 between 2700 and 3200 elements.

Animation 4: All browsers started below Level 3 attaining around 20 fps (Fig. 6.4). Then, the fps of all browsers except Mozilla Firefox and Yandex Browser grew to Level 2 between 31 to 35 fps, peaking at 500 rendered elements. The browsers fell back to Level 3 between 700 and 1300 elements and below it between 1200 and 1850 elements. Mozilla Firefox peaked at 200 elements reaching 42 fps and didn't fall back to Level 3 even for the largest tested threshold. Yandex peaked at 1150 elements reaching 27 fps and thereby remained in Level 3 instead of reaching Level 2. Konqueror performed

worse than the majority of the browsers with around a 4 fps difference starting at 1200 elements.

Animation 5: Mozilla Firefox performed the best remaining in the upper part of Level 2 and in the middle part of Level 2 starting at 2000 elements (Fig. 6.5). Konqueror scored the worst and started slightly above Level 3 with 32 fps, then fell down to Level 3 at 600 elements and below it at 1000 elements. Other browsers performed similarly with Google Chrome at the top of the remaining browsers. The browsers started in Level 1 and Level 2 with 54 to 59 fps, then fell to Level 3 between 1050 and 1300 rendered elements and then fell below it between 1350 and 1650 elements.

Animation 6: Comparable results with animation 1 were obtained for animation 6 with respect to the order of the most performant browsers, but with the difference that Chromium scored second place (Fig. 6.6). All browsers started in Level 1 falling to Level 2 between 100 and 250 rendered elements. Mozilla Firefox fell to Level 2 at 1200 elements, to Level 3 at 4000 elements and below Level 3 at 4500 elements. Konqueror fell to Level 3 at 900 elements and below Level 3 at 1200 elements. The other browsers fell to Level 3 between 1400 and 1800 elements and below it between 1800 and 2500 elements.

Animation 7 and 8: In animation 7 Konqueror, Brave and Yandex Browser started in the bottom part of Level 2 attaining between 32 and 35 fps, while the other browsers started in Levels 1 and 2 attaining between 50 and 57 fps (Fig. 6.7). Yandex Browser performed the worst falling to Level 3 at 250 elements and below it at 600 elements. Konqueror and Brave caught up with the other browsers and all those browsers reached Level 3 between 600 and 1400 elements and below it between 900 and 1750 elements.

In animation 8 Konqueror started the worst attaining 35 fps, but still in Level 2 and then it caught up with the rest of the browsers (Fig. 6.8). The other browsers started in the upper part of Level 2 attaining 48 to 55 fps. Chromium, Vivaldi and Konqueror peaked at 500 elements, Mozilla Firefox remained at the same level of 51 fps till 500 elements and other browsers started falling immediately. The browsers fell to Level 3 between 700 and 1400 elements and then below it between 800 and 1700 rendered elements.

In both animations, Mozilla Firefox and Chromium performed the best while Yandex Browser, Opera and Microsoft Edge were the worst. What's interesting to notice is that in both animations graphs above 1400 elements look almost identical. This is most likely caused by the similarity in the animations themselves.

Animation 9: Mozilla Firefox performed the worst maintaining around 17 fps throughout each threshold (Fig. 6.9). This surprising result is discussed later. Konqueror scored second worst and started in Level 2 with 37 fps, then fell to Level 3 at 1200 elements and then below it at 1400 elements. Other browsers started in Levels 1 and 2 attaining 52 to 58 fps and performed similarly until 1000 elements. Then, Google Chrome, Brave and Vivaldi separated and performed better than the other browsers falling to Level 3 between 1800 and 1900 elements and below it between 1980 and 2200 elements, while the other browsers fell to Level 3 at 1400 to 1600 elements and below it at 1640 to 1780 elements.

Animation 10: Microsoft Edge performed the worst starting in Level 2 at 45 fps and falling to Level 3 at 85 elements (Fig. 6.10). Then, it went down below Level 3 at 100 elements. Konqueror and Mozilla Firefox performed the best both starting in Level 1. Then, Konqueror fell to Level 2 at 63 rendered elements and Mozilla Firefox at 25 elements.

Konqueror and Mozilla Firefox performed the best and Microsoft Edge the worst. Konqueror continued falling reaching Level 3 at 130 elements and Mozilla Firefox at

174 elements. Mozilla Firefox remained in Level 3 even for the largest tested threshold and Konqueror fell below Level 3 at 165 elements. Opera started in Level 2 at 50 fps but then caught up with the majority of the browsers. Other browsers started in Level 1 or just below it and then continued falling to Level 3 reaching it between 92 and 98 elements. Finally, they fell below Level 3 between 103 and 124 elements.

The first thing worth noting is that for some animations a peak fps is reached for the second threshold and in most cases it's a significant growth of 10 – 20 fps between the first and the second measured threshold. This phenomenon occurs for all or almost all browsers in animations 2, 3 and 4, for browsers: Brave, Vivaldi and Yandex Browser in animation 1 and for browsers: Konqueror, Chromium and Vivaldi in animation 8. This might be a result of some in-browser optimizations for animating specific CSS properties.

The second unusual phenomenon is Mozilla Firefox scoring the highest or in the top in eight of the animations and surprisingly the worst in two animations reaching around 9 and around 17 fps consistently for all thresholds. This result might be explained by a lack of optimization for animating the scaling and the background or gradient background.

Also, it can be noticed, that in animations 7 and 8, that is animating the CSS property "width" by setting the value in pixels instead of percentage seems to be less performant for some browsers. This is contrary to what might be expected as one could assume that units do not influence the animation performance or that setting the value in pixels would be more performant.

8. Summary

The thesis's topic gave rise to numerous challenges such as inventing the methodology of measuring animation fluidity, performing the measurements for each browser, animation and threshold 30 times and analysing the results. However, the overall outcome can be considered a success as everything turned out to work correctly.

8.1 Further research possibilities

The following are possibilities for extending the work shown in the thesis:

- Adding more browsers
- Adding many more CSS properties
- Adding more 3D animations
- Adding more thresholds for additional result accuracy
- Optimising the automation scripts
- Choosing a better screen recorder as obs crashes occasionally
- Performing separate tests specifically for Windows and macOS-compatible browsers
- Performing separate tests for mobile-oriented browsers
- Potentially creating a dedicated automated software for measuring animation performance

9. Table of figures

Figure 3.1Browser components [12,16]……..…….7
Figure 3.2 WebKit engine's general flow [39]……...…..……10
Figure 3.3 Gecko engine's general flow [39]……...11
Figure 3.4 2015 to 2022 global desktop browser market share. Image source: [47].....12
Figure 4.1 The main interface of Brave……..…….14
Figure 4.2 The main interface of Chromium……..…….14
Figure 4.3 The main interface of Google Chrome……..…….15
Figure 4.4 The main interface of Konqueror…….……..…….15
Figure 4.5 The main interface of Microsoft Edge…….……..…….16
Figure 4.6 The main interface of Mozilla Firefox…….……..…….17
Figure 4.7 The main interface of Opera……..…….17
Figure 4.8 The main interface of Vivaldi…….18
Figure 4.9 The main interface of Yandex Browser……..……...…18
Figure 4.10 Classification of browsers based on the underlying browser and JavaScript engines……..……...….19
Figure 4.11 The main interface of OBS Studio…….……....….21
Figure 4.12 The main interface of WebStorm…….……....….21
Figure 4.13 The main interface of Kate…….…….……....….22
Figure 4.14 The main interface of mpv…….……..…….…..….22
Figure 4.15 The main interface of melt…….……...……...….23
Figure 4.16 The process of performing measurements……...….24
Figure 5.1 Depiction of animation 1……...….33
1) Phase of translating the element to the right
2) Phase of translating the element to the starting position
Figure 5.2 Depiction of animation 2…….……...34
1) and 2) Phases of rotating the element over 360 degrees
Figure 5.3 Depiction of animation 3……....36
1) The elements are shrunk (scaled down)
2) The elements are scaled back up
Figure 5.4 Depiction of animation 4……....37
1) and 2) The elements are skewed on both diagonals
Figure 5.5 The first stage of animation 5……..……..38
Figure 5.6 The second stage of animation 5…….39
Figure 5.7 Depiction of animation 6…….…...……....40
Figure 5.8 Depiction of animation 7…….……...41
Figure 5.9 Depiction of animation 8…….……...……....42
Figure 5.10 Depiction of animation 9…….…….……43

Figure 5.11 Depiction of animation 10……..…….……..…….45
Figure 6.1 Performance graph of animation 1…….…….……48
Figure 6.2 Performance graph of animation 2…….……..…….……..49
Figure 6.3 Performance graph of animation 3…….…….50
Figure 6.4 Performance graph of animation 4…….…….……....51
Figure 6.5 Performance graph of animation 5……...……...53
Figure 6.6 Performance graph of animation 6……..……....54
Figure 6.7 Performance graph of animation 7…….…….……....55
Figure 6.8 Performance graph of animation 8……..56
Figure 6.9 Performance graph of animation 9……..57
Figure 6.10 Performance graph of animation 10……..……58

10. Table of listings

Listing 5.1 HTML file of the project containing animations……...……....28
Listing 5.2 The beginning of the JavaScript file of the project containing animations…….…..29
Listing 5.3 Basic CSS styles that apply to html and body tags……..…..30
Listing 5.4 CSS styles that apply to div with the id of "app-wrapper"…….……..31
Listing 5.5 CSS styles that apply to "app-wrapper" and "box" elements. Definition of animations for white and black backgrounds.……...…..…..32
Listing 5.6 The first part of the code specific to animation 1……..…….……..34
Listing 5.7 The second part of the code specific to animation 1…….…..…..34
Listing 5.8 The first part of the code specific to animation 2……...…..…..35
Listing 5.9 The second part of the code specific to animation 2...……...…35
Listing 5.10 The first part of the code specific to animation 3……...…..……...…….36
Listing 5.11 The second part of the code specific to animation 3……..……..37
Listing 5.12 The first part of the code specific to animation 4…….…..…..37
Listing 5.13 The second part of the code specific to animation 4……...……..…..38
Listing 5.14 The first part of the code specific to animation 5……...……...…..39
Listing 5.15 The second part of the code specific to animation 5…….…..…..39
Listing 5.16 The first part of the code specific to animation 6……...…..……....40
Listing 5.17 The second part of the code specific to animation 6……...…...…...…..41
Listing 5.18 The first part of the code specific to animation 7……...…….....…….42
Listing 5.19 The second part of the code specific to animation 7……...…....…….42
Listing 5.20 The first part of the code specific to animation 8……...……..…….42
Listing 5.21 The second part of the code specific to animation 8……...……..…….43
Listing 5.22 The first part of the code specific to animation 9……...……..…..…..43
Listing 5.23 The second part of the code specific to animation 9……...……...…...…….44
Listing 5.24 The third part of the code specific to animation 9……..……...…...…..…..44
Listing 5.25 The first part of the code specific to animation 10……...…..…..45
Listing 5.26 The second part of the code specific to animation 10…….…...…...…….46
Listing 5.27 The third part of the code specific to animation 10…….……47

11. List of tables

Table 5.1 Threshold definitions for each animation……...….....27
Table 6.1 Results table for animation 1…….....….....49
Table 6.2 Results table for animation 2…….....……..50
Table 6.3 Results table for animation 3…….....……..51
Table 6.4 Results table for animation 4…….....……..52
Table 6.5 Results table for animation 5…….....……..53
Table 6.6 Results table for animation 6…….....……..54
Table 6.7 Results table for animation 7…….....……..55
Table 6.8 Results table for animation 8…….....……..56
Table 6.9 Results table for animation 9…….....……..57
Table 6.10 Results table for animation 10…….....……..58

12. References

[1] MDN Web Docs, CSS: Cascading Style Sheets accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Web/CSS>
[2] World Wide Web Consortium (W3C), All CSS specifications, accessed 3 September 2023, <https://www.w3.org/Style/CSS/specs.en.html>
[3] Keith Grant, CSS in Depth, published 8 March 2018
[4] W3Schools, CSS Tutorial, accessed 3 September 2023, <https://www.w3schools.com/css/>
[5] MDN Web Docs, Using CSS animations, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations>
[6] Wikipedia, CSS animations, accessed 3 September 2023, <https://en.wikipedia.org/wiki/CSS_animations>
[7] vikashgautam11 for GeeksforGeeks, What is a Web Browser?, accessed 3 September 2023, <https://www.geeksforgeeks.org/web-browser/>
[8] Don Norman and Jakob Nielsen for Nielsen Norman Group (NN/g), The Definition of User Experience (UX), accessed 3 September 2023, <https://www.nngroup.com/articles/definition-user-experience/>
[9] MDN Web Docs, Rendering engine, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Glossary/Rendering_engine>
[10] Wikipedia, Usage share of web browsers, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Usage_share_of_web_browsers>
[11] Fred Churchville for TechTarget, What is user interface (UI)?, accessed 3 September 2023, <https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI>
[12] Venkat.R for Medium, Behind Browser (Basics — Part 1), accessed 3 September 2023, <https://medium.com/@ramsunvtech/behind-browser-basics-part-1-b733e9f3c0e6>
[13] Wikipedia, Browser engine, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Browser_engine>
[14] MDN Web Docs, Populating the page: how browsers work, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work>
[15] Wikipedia, Comparison of browser engines, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Comparison_of_browser_engines>
[16] Ritik Singh for BrowserToUse, What is a Web Browser? How Does it Work?, accessed 3 September 2023, <https://browsertouse.com/blog/1190/what-is-web-browser-how-it-works/>
[17] Jamie Juviler for HubSpot, What Is GUI? Graphical User Interfaces, Explained, accessed 3 September 2023, <https://blog.hubspot.com/website/what-is-gui>
[18] Wikipedia, Gecko (software), accessed 3 September 2023, <https://en.wikipedia.org/wiki/Gecko_(software)>
[19] Mozilla Foundation, Get Firefox for desktop, accessed 3 September 2023, <https://www.mozilla.org/en-US/firefox/new/>
[20] Google – Chromium, Blink (Rendering Engine), accessed 3 September 2023, <https://www.chromium.org/blink/>
[21] Google – Google Chrome, Google Chrome, accessed 3 September 2023, <https://www.google.com/chrome/>
[22] WebKit – Apple, WebKit, accessed 3 September 2023, <https://webkit.org/>
[23] Apple, Safari, accessed 3 September 2023, <https://www.apple.com/safari/>
[24] MDN Web Docs, HTML: HyperText Markup Language, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Web/HTML>
[25] WHATWG, HTML Standard, accessed 3 September 2023, <https://html.spec.whatwg.org/>
[26] Javier Garza and Stephen Ludin, Learning HTTP/2: A Practical Guide for Beginners, published in 2017
[27] freeCodeCamp, Just in Time Compilation Explained, accessed 3 September 2023, <https://www.freecodecamp.org/news/just-in-time-compilation-explained/>
[28] Google – V8, V8 JavaScript engine, accessed 3 September 2023, <https://v8.dev/>
[29] Mozilla Foundation, SpiderMonkey JavaScript/WebAssembly Engine, accessed 3 September 2023, <https://spidermonkey.dev/>
[30] Apple – JavaScriptCore, JavaScriptCore | Apple Developer Documentation, accessed 3 September 2023, <https://developer.apple.com/documentation/javascriptcore>
[31] Rufai Mustapha for freeCodeCamp, What is HTTP? Protocol Overview for Beginners, accessed 3 September 2023, <https://www.freecodecamp.org/news/what-is-http/>

[32] Cloudflare, What is HTTPS?, accessed 3 September 2023,
<https://www.cloudflare.com/learning/ssl/what-is-https/>
[33] Dropbox, What is FTP?, accessed 3 September 2023,
<https://experience.dropbox.com/resources/what-is-ftp>
[34] Kaspersky, What Are Internet Cookies and What Do They Do?, accessed 3 September 2023,
<https://www.kaspersky.com/resource-center/definitions/cookies>
[35] W3Schools, HTML Web Storage API, accessed 3 September 2023,
<https://www.w3schools.com/html/html5_webstorage.asp>
[36] web.dev, IndexedDB API, accessed 3 September 2023, <https://web.dev/indexeddb/>
[37] Wikipedia, PDF, accessed 3 September 2023, <https://en.wikipedia.org/wiki/PDF>
[38] Amazon – AWS, What is XML? - Extensible Markup Language (XML) Explained, accessed 3 September 2023, <https://aws.amazon.com/what-is/xml/>
[39] Tali Garsiel and Paul Irish, How browsers work, accessed 3 September 2023,
<https://web.dev/howbrowserswork/>
[40] MDN Web Docs, Document Object Model (DOM), accessed 3 September 2023,
<https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model>
[41] Microsoft, Get to Know Microsoft Edge, accessed 3 September 2023,
<https://www.microsoft.com/en-us/edge?form=MA13FJ>
[42] MDN Web Docs, User-Agent – HTTP, accessed 3 September 2023,
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent>
[43] Google – Chromium, Chromium, accessed 3 September 2023, <https://www.chromium.org/Home/>
[44] Opera, Opera Web Browser | Faster, Safer, Smarter, accessed 3 September 2023,
<https://www.opera.com/>
[45] Brave Software, Brave: Secure, Fast, & Private Web Browser with Adblocker, accessed 3 September 2023, <https://brave.com/>
[46] Vivaldi Technologies, Vivaldi Browser | Powerful, Personal and Private web browser, accessed 3 September 2023, <https://vivaldi.com/>
[47] Statista, Global desktop internet browsers market share 2015-2022, accessed 3 September 2023,
<https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/>
[48] Brave Software, Transparency Data Feed, accessed 3 September 2023,
<https://brave.com/transparency/>
[49] Wikipedia, Google, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Google>
[50] Synopsys, The Chromium (Google Chrome) Open Source Project on Open Hub: Languages Page, accessed 3 September 2023, <https://openhub.net/p/chrome/analyses/latest/languages_summary>
[51] KDE, Konqueror – KDE Applications, accessed 3 September 2023, <https://apps.kde.org/konqueror/>
[52] KDE, Home – KDE Community, accessed 3 September 2023, <https://kde.org/>
[53] KDE, Dolphin – KDE Applications, accessed 3 September 2023, <https://apps.kde.org/dolphin/>
[54] Wikipedia, KHTML, accessed 3 September 2023, <https://en.wikipedia.org/wiki/KHTML>
[55] KDE TechBase, Projects/WebKit/Part, accessed 3 September 2023,
<https://techbase.kde.org/Projects/WebKit/Part>
[56] Wikipedia, KJS, accessed 3 September 2023, <https://en.wikipedia.org/wiki/KJS>
[57] Microsoft, About Microsoft | Mission and Vision, accessed 3 September 2023,
<https://www.microsoft.com/en-us/about>
[58] Wikipedia, Internet Explorer, accessed 3 September 2023,
<https://en.wikipedia.org/wiki/Internet_Explorer>
[59] Microsoft – Windows Blogs, Introducing Windows 11 | Windows Experience Blog, accessed 3 September 2023, <https://blogs.windows.com/windowsexperience/2021/06/24/introducing-windows-11/>
[60] Mozilla Foundation, Mozilla Foundation - Homepage, accessed 3 September 2023,
<https://foundation.mozilla.org/en/>
[61] GNU, Linux and GNU – GNU Project – Free Software Foundation, accessed 3 September 2023,
<https://www.gnu.org/gnu/linux-and-gnu.en.html>
[62] Opera, About Opera | 25+ Years of Building Opera, accessed 3 September 2023,
<https://www.opera.com/about>
[63] Wikipedia, Presto (browser engine), accessed 3 September 2023,
<https://en.wikipedia.org/wiki/Presto_(browser_engine)>
[64] Wikipedia, Vivaldi Technologies, accessed 3 September 2023,
<https://en.wikipedia.org/wiki/Vivaldi_Technologies>
[65] Wikipedia, Yandex Browser, accessed 3 September 2023,
<https://en.wikipedia.org/wiki/Yandex_Browser>
[66] Yandex, Yandex, accessed 3 September 2023, <https://yandex.com/company/>

[67] Wikipedia, Yandex Search, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Yandex_Search>
[68] Wikipedia, 360 Secure Browser, accessed 3 September 2023, <https://en.wikipedia.org/wiki/360_Secure_Browser>
[69] UC Browser, UC Browser – Fast video downloader, 20GB free cloud storage, download UC Browser, accessed 3 September 2023, <https://www.ucweb.com/>
[70] Apple Insider, macOS | Sonoma, Ventura, Monterey, accessed 3 September 2023, <https://appleinsider.com/inside/macos>
[71] Wikipedia, Sogou, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Sogou>
[72] Maxthon, Maxthon Browser, accessed 3 September 2023, <https://www.maxthon.com/en/>
[73] Lawrence Rake for Digital Detective, QQ Browser, accessed 3 September 2023, <https://kb.digital-detective.net/display/BF/QQ+Browser>
[74] Coc Coc, Coc Coc Browser | The web browser for Vietnamese people, accessed 3 September 2023, <https://coccoc.com/en>
[75] Wikipedia, Microsoft Windows, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Microsoft_Windows>
[76] MDN Web Docs, Polyfill, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Glossary/Polyfill>
[77] Wikipedia, Trident (software), accessed 3 September 2023, <https://en.wikipedia.org/wiki/Trident_(software)>
[78] Manjaro, Manjaro, accessed 3 September 2023, <https://manjaro.org/>
[79] philm for Manjaro Linux Forum, Manjaro 22.1 Talos released, accessed 3 September 2023, <https://forum.manjaro.org/t/manjaro-22-1-talos-released/139155>
[80] KDE, Plasma – KDE Community, accessed 3 September 2023, <https://kde.org/plasma-desktop/>
[81] Arch Linux, Arch Linux, accessed 3 September 2023, <https://archlinux.org/>
[82] Marius Nestor for Softpedia, Why a Rolling Release Model is the Way to Go for Any OS, accessed 3 September 2023, <https://news.softpedia.com/news/Why-a-Rolling-Release-Model-is-the-Way-to-Go-for-Any-OS-482089.shtml>
[83] OBS Studio, Open Broadcaster Software – OBS Studio, accessed 3 September 2023, <https://obsproject.com/>
[84] JetBrains, WebStorm: The Smartest JavaScript IDE, by JetBrains, accessed 3 September 2023, <https://www.jetbrains.com/webstorm/>
[85] Amazon – AWS, What Is An IDE (Integrated Development Environment)?, accessed 3 September 2023, <https://aws.amazon.com/what-is/ide/>
[86] JetBrains, JetBrains: Essential tools for software developers and teams, accessed 3 September 2023, <https://www.jetbrains.com/>
[87] Kate, Kate Editor, accessed 3 September 2023, <https://kate-editor.org/>
[88] mpv, mpv, accessed 3 September 2023, <https://mpv.io/>
[89] MLT, MLT – Home, accessed 3 September 2023, <https://www.mltframework.org/>
[90] Wikipedia, Command-line interface, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Command-line_interface>
[91] Jordan Sissel on GitHub, xdotool – X11 automation tool, accessed 3 September 2023, <https://github.com/jordansissel/xdotool>
[92] Wikipedia, X.Org Server, accessed 3 September 2023, <https://en.wikipedia.org/wiki/X.Org_Server>
[93] FFmpeg, FFmpeg, accessed 3 September 2023, <https://ffmpeg.org/>
[94] MediaArea, MediaInfo, accessed 3 September 2023, <https://mediaarea.net/en/MediaInfo>
[95] Python Software Foundation, Welcome to Python.org, accessed 3 September 2023, <https://www.python.org/>
[96] mrdoob on GitHub, JavaScript Performance Monitor, accessed 3 September 2023, <https://github.com/mrdoob/stats.js>
[97] Wikipedia, Frame rate, accessed 3 September 2023, <https://en.wikipedia.org/wiki/Frame_rate>
[98] MDN Web Docs, Window: requestAnimationFrame() method, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame>
[99] MDN Web Docs, What are browser developer tools?, accessed 3 September 2023, <https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Tools_and_setup/What_are_browser_developer_tools>
[100] Wistia, What is Frame Rate for video? - Wistia Blog, accessed 3 September 2023, <https://wistia.com/learn/production/what-is-frame-rate>