JPMTR-2110
DOI 10.14622/JPMTR-2110
UDC 004.92:004.7|004.03

Research paper | 163
Received: 2021-08-15
Accepted: 2022-04-08

A comparison of current solutions for creating web animations on Apple hardware

Jaka Mušič and Helena Gabrijelčič Tomc

University of Ljubljana, Faculty of Natural Sciences and Engineering,
Department of Textiles, Graphic Arts and Design,
Chair of Information and Graphic Arts Technology

helena.gabrijelcic@ntf.uni-lj.si

Abstract

The content available online consists of titles, texts, images, and multimedia, including animations. Some animations are immediately noticeable, while others are less obvious, yet they play an important role in the interactivity of a website. When implemented correctly and working properly, animations can greatly enhance the user experience. In the theoretical part of this paper, the basic technologies of the web are introduced. These are Hypertext Markup Language, Cascading Style Sheets (CSS) and JavaScript. The history of web animations is presented as well as the perception of animations in terms of the qualities required by the human eye to perceive them as fluid. In the experimental part, methods for creating animations were selected for testing. These were CSS animations, Web Animations Application Programming Interface (WAAPI) and Lottie animations. For each of these methods, an identical animation was designed and implemented on a website. The animations were then tested with a different number of elements in two browsers on multiple devices and Apple hardware. During creation and testing, an analysis was made of how difficult it is to design and implement the animations on a website, and how powerful they are: the ability to display a high number of frames per second, how much memory they require on the graphics card, and how much data must be transferred from the server for them to play properly. The results were recorded and summarised. Some recommendations were made in the results, i.e. which method to use depending on the intended use case. The CSS and WAAPI animations and transitions were found to be more suitable for complex web animations, while Lottie was found to be useful for simpler animations.

Keywords: interactive website, cascading style sheets (CSS), web animations application programming interface (WAAPI), Lottie animations

1. Introduction

We are constantly surrounded by countless amounts of information, both in physical space and on the internet. It can thus become extremely exhausting to find the information we are looking for or to critically evaluate the value of the information we have. One of the tools we can use to enrich an online document with data and highlight the importance and value of certain information is animation. This can be in the form of micro-animations (for example, the transformation of the menu icon into a backspace icon) or animations that involve more intense movement (Pratt, et al., 2010). These are, for example, the animated graphic elements that accompany or guide us through the website. They make websites more attractive, but can also contribute to the user experience by speeding up the recognition of graphic elements (Head, 2016). Animations have evolved since the beginning of web development, and every year a new solution for creating them appears. Moreover, the demand for animations is growing as websites and webpages change from static to dynamic. However, these solutions are not always of the same quality and complexity to use, and can also be costly in terms of hardware (Chinnathambi, 2017).

Web technologies have been evolving since 1989, when Tim Berners-Lee, then an employee of the European Council for Nuclear Research (CERN), had the idea for the World Wide Web and invented the Hypertext Transfer Protocol (HTTP) to properly display webpages, from servers to client devices, that is, in the browser of a computer or mobile device (Chrome, Safari, Firefox, etc.) (McCullough, 2018).

When we want to access a webpage, an HTTP request is made in this way, but not necessarily in this order, except for the first step: the browser requests the page, the server provides the Hypertext Markup Language (HTML) file, and if needed the browser requests the style file, the request is answered by returning the cascading style sheet (CSS) file, the browser requests images and additional files specified in the HTML file, the server provides images and additional files, the browser then also requests scripts and receives JavaScript (JS) files from the server (McCullough, 2018; Berners-Lee, 2021). We can now use the latest versions of CSS not only to design but also to animate content. For this purpose, two properties are available, namely animation and transition. In the first one, we can use keyframes to display the position and other properties for a particular frame. This animation can be called multiple times and can be given duration, delay and repeat properties. A transition, on the other hand, is more used for mouse movements, clicks, and mouse transformations (Weyl, 2016; W3Schools, 2021).

1.1 From early beginnings to modern animations

Animation is the perception of movement and the illusion of change using sequences of images that differ only slightly from one another (Ferreira, 2017). Simply put, animation is the movement of graphics or the visualisation of change over time, and while the basic ideas have remained the same, the methods of animation have changed over the years.

Dynamic elements on websites have been driven by Graphics Interchange Format (GIF) technology. The GIF format was introduced in 1987, and was the first technology to animate the previously static frames seen online. This format is now more than 30 years old and is still widely used. Due to its structure, it can represent 256 different colours (a JPEG can represent 16 million), and an advantage of this format is that a file can contain several consecutive frames. Today they are often used in reaction GIFs (a genre of meme), which can be used as a response instead of text in various online contexts. There are various online collections where users can find the perfect reaction GIF, such as Giphy (Shamms Mortier, 1997).

In the late 1990s, Adobe developed Flash technology, which was promoted as “a standard for delivering rich web content with powerful impact. Graphics, animations, and user interfaces can be rendered on all browsers and platforms” (Adobe, 2021). However, Flash has now fallen out of use and has not been supported since January 1, 2021, because it required considerable hardware resources that mobile devices were not able to provide at the time, and also had security vulnerabilities (Carrera, 2010).

1.2 SVG animations

Scalable Vector Graphics (SVG) animations are based on HTML elements, but add something unique, such as the path element. Their advantage is very good support in various browsers, except Internet Explorer. However, they are relatively expensive to use, and are becoming less common due to better alternatives being available (Drasner, 2017; Bellamy-Royds, Cagle and Storey, 2017).

1.3 CSS3 animation technology

Cascading Style Sheets version 3 (CSS3), was introduced, along with HTML5, to meet the needs of developers for better animation technology. Together these two new systems paved the way for interactive websites, creating a large number of interesting projects, although some issues still remained and thus many developers resorted to libraries written specifically for animation (Weyl, 2016; Chinnathambi, 2017).

1.4 Web Animations API

Web Animations Application Programming Interface (WAAPI) is one of the newer technologies for creating web animations. It is built on top of JS and is based on two models, for timing and animation. The timing model is the basis for working with the Application Programming Interface (API). Each website and each document has its own timeline that extends from page loading to infinity, or to the point when the tab is closed. The animations are distributed along the timeline in an order specified by the startTime parameter. An animation model is a sequence of images arranged according to a time model. This model thus ensures that animations or changes to elements are executed in the order specified by the time model (MDN Web Docs, 2021).

1.5 Libraries

There are many libraries on the internet that help us create or add animations to elements. One of them is animate.css, which provides us with preset CSS animations that we can easily attach to elements using classes, identifiers, or element names. However, the most advanced option is the GreenSock Animation Platform (GSAP) library, which allows users to create their own animations, is based on JS, and also supports various plug-ins (GreenSock, 2020).

1.6 Lottie animations

Lottie animations are JavaScript Object Notation (JSON) animation files that make it possible to publish and display animations similar to static files (images). The files are small and of the vector type, so there are no size limitations due to resolution. Lottie is also a library for Android, iOS, React, and the web, which converts animations created in Adobe After Effects (which must be exported as a JSON file using the Bodymovin plug-in) and then rendered natively by Lottie on the device (Bassett, 2015).

1.7 User aspects and objectives of the research

For the user of a website or mobile app, the technical side or implementation of a system is not as important as its usability, usefulness, learnability, and so on. For animations, this means they need to load quickly and run smoothly. The latter aspect is particularly important, as it greatly affects the user experience (Head, 2016; Pierce, 2016; Brundrett, 2016).

The human eye has a light-sensitive tissue called the retina, located at the back, opposite the lens. Its photoreceptors are responsible for converting an image into an electrical signal and sending it to the brain.

When an object moves through our field of vision, the photoreceptors are stimulated. However, the brain cannot track objects that move faster than sixteen times a second, and this is why most videos and movies are shot and played at 24 frames per second. Since this is more than we can perceive, we see the separate images as a single movement rather than a sequence of shots.

The eye also has a limit to the amount of motion it can detect. Although change is a continuous flow of information and is not perceived by the eye in individual frames, for most people this limit is 60 Hz or less (or about 60 frames per second). Recent research suggests that we might see up to 75 Hz, or that the lower limit of the speed at which we can perceive change is as low as 13 milliseconds, but such measurements are very difficult to make, so there is no definitive answer (Potter, et al., 2014). Most digital displays have a refresh rate of 60 Hz, which seems to be sufficient at the present (Merleau-Ponty, 2013). For the purposes of this study, the lower limit was more important because it determines how the animation is perceived: whether it is smooth or choppy.

This research was planned with the aim of uncovering the best balance among simplicity, usability, and complexity of web animations for designers, developers, users and devices. The main goal of the research was to find a solution for web animation design. We examined the design method, the complexity of implementation in the web environment, and the impact on hardware resources: how many frames per second the browser can display, how much memory is used on the graphics card, and how much data must be transferred from the server to the device for the animation to work.

The objectives of the research were to review some recent solutions for creating web animations (CSS3 animations, WAAPI, and Lottie animations), to test and compare these, and to analyse and select the most suitable solution in terms of the relationship between operational efficiency and ease of production.

Before conducting the research, we hypothesised the following:

- Lottie animations offer the best balance between simplicity and performance,
- WAAPI animations will consume more hardware resources than CSS animations,
- Safari will play animations at more frames per second than other browsers.

2. Experimental

In the experimental part, the animations were created, analysed, and tested in three different technologies: CSS3 animations, WAAPI, and Lottie animations.

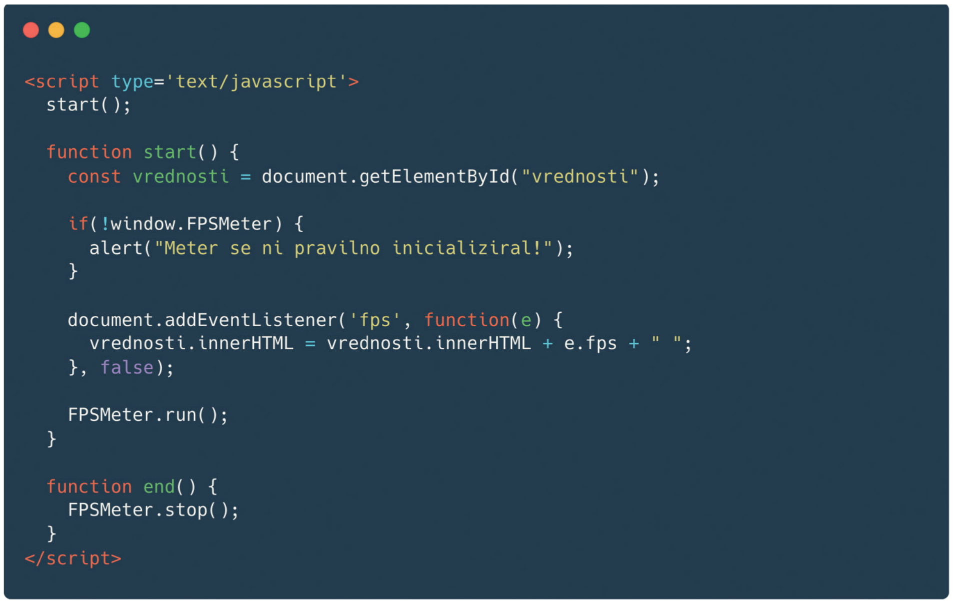
The visual appearance of the animations was the same, the differences were only in the design, development and rendering process. The animations were tested on a simple website created for this purpose, with different technologies to choose from. The tool also displayed the number of frames per second that the animation could show.

2.1 The development of the tool

There are several ways to test the quality of animations. We could test how much space they take up on the graphics card RAM, how much processing power they require, or how much data is transferred from the server, but the most important is the rate of frames per second at which they can be displayed, since this is the only aspect that visitors perceive. Therefore, this is the most important metric we used to test the quality of our animations.

At the time of writing, none of the popular browsers had developer tools that provided a detailed overview of the information we needed to test the technologies. Therefore, we developed a tool that displays and measures the performance of animations. This also opened up the possibility of testing animations on mobile devices. Figure 1 shows the entire implementation of the functional part of the frame per second (fps) rate measurement.

In Figure 1 FPSMeter evaluates the framerate of an animation embedded in a web page using CSS transitions. We used fpsmeter from Corvoysier (2014). The principle uses CSS itself and three steps to evaluate the



```
<script type='text/javascript'>
  start();

  function start() {
    const vrednosti = document.getElementById("vrednosti");

    if(!window.FPSMeter) {
      alert("Meter se ni pravilno inicializiral!");
    }

    document.addEventListener('fps', function(e) {
      vrednosti.innerHTML = vrednosti.innerHTML + e.fps + " ";
    }, false);

    FPSMeter.run();
  }

  function end() {
    FPSMeter.stop();
  }
</script>
```

Figure 1: Script that takes care of calling and manipulating the frame rate measurement script

actual rendering frame rate of a page. The steps are the insertion of the CSS animated item in a page, calculation of the computed position of the CSS animated item at regular intervals and, finally, for every elapsed second there is a counting of the number of different positions occupied by the item (Corvoysier, 2011).

2.2 The making of animations

The canvas size was 1 920 pixels wide and 1 080 high. For the animation, we designed eight graphic elements with different shapes (four circles and four rectangles), sizes and colours. In the initial state, the graphic elements were placed randomly on the canvas without covering each other (Figure 2).

The animation consists of six keyframes (Figure 2), with the first and last keyframes being identical, allowing for an infinite loop. At each keyframe, we specify the position, rotation, size, and colour of the elements. Between the keyframes, we let the technologies compute the transition paths. This template was then made for Lottie in Adobe After Effects and exported with the help of a free plug-in Bodymovin into the required JSON file and developed in CSS and WAAPI. All three types of animation were then implemented in the tool for testing.

Tables A1 and A2 in Appendix show the changes in the circular (Table A1) and rectangular (Table A2) graphic elements during six levels of testing.

The animation is 2 500 ms (or two and a half seconds) long, and the keyframe comes every 500 ms. We chose this length so that the timing would be as fast as possible, that the animations is as demanding as possible for the browser, and at the same time slow enough for pleasant viewing.

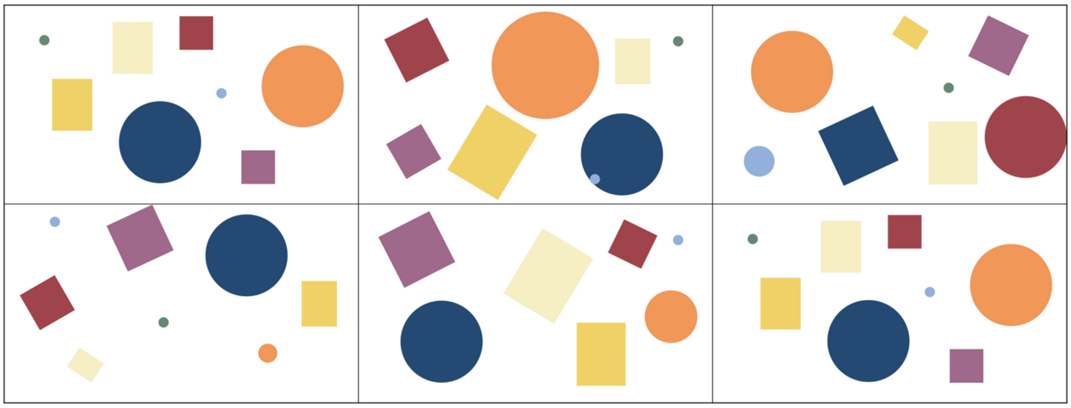
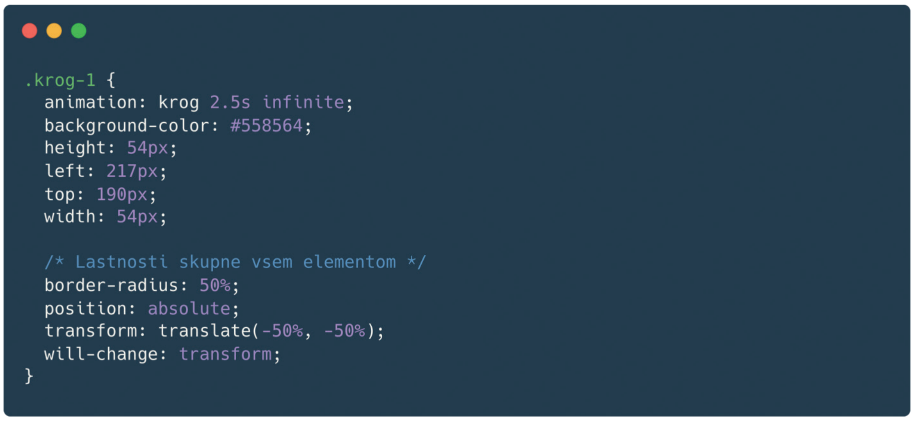


Figure 2: Keyframes of the animation used in the experiment

2.2.1 CSS animations code

While the Lottie animations have less programming, the CSS animations have everything written in code. For CSS animations, we used the method of keyframes in CSS, where we set six key levels that were tied to the percentage of elapsed time rather than time. We started the process again by creating eight graphic elements and setting the basic parameters. We had to determine the position from the left and top, and for those values to be valid we also had to determine the absolute positioning and the position correction by half in both directions. At the same time, we had to set the size, colour, and radius of the frame (50 %) in the case of a circle. Animation information (name, length and loop) was also required for the animation to work. To improve responsiveness, we added information to the browser about which property will change (will-change property). The complete syntax for one graphic element is shown in Figure 3.

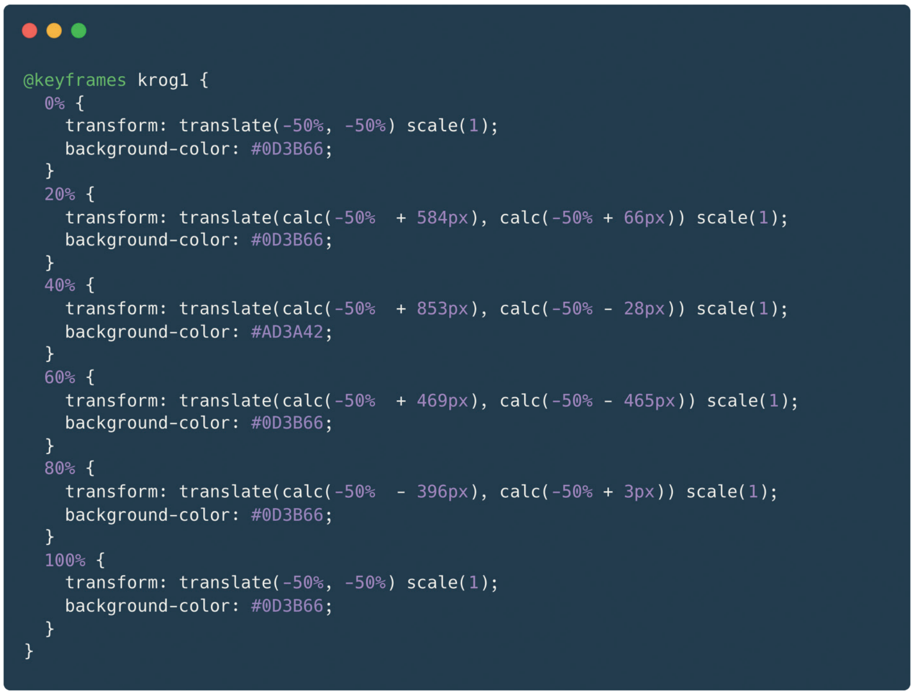
Keyframes were defined to allow the browser to track changes in graphic elements. We had some main key-frames in our animation, so we split the frames into CSS keyframes in segments of 20 % each to cover all the main frames, since the first and last frames had to be identical to allow a seamless loop. We also used the CSS transform property for movement instead of the traditional positioning property, as this ensured that the browser used GPU acceleration for rendering instead of the slower browser rendering engine. The key to this property was not to enter the offset value,



```css
.krog-1 {
  animation: krog 2.5s infinite;
  background-color: #558564;
  height: 54px;
  left: 217px;
  top: 190px;
  width: 54px;

  /* Lastnosti skupne vsem elementom */
  border-radius: 50%;
  position: absolute;
  transform: translate(-50%, -50%);
  will-change: transform;
}
```

Figure 3: CSS code for the circular graphic element in the animation



```css
@keyframes krog1 {
  0% {
    transform: translate(-50%, -50%) scale(1);
    background-color: #0D3B66;
  }
  20% {
    transform: translate(calc(-50%  + 584px), calc(-50% + 66px)) scale(1);
    background-color: #0D3B66;
  }
  40% {
    transform: translate(calc(-50%  + 853px), calc(-50% - 28px)) scale(1);
    background-color: #AD3A42;
  }
  60% {
    transform: translate(calc(-50%  + 469px), calc(-50% - 465px)) scale(1);
    background-color: #0D3B66;
  }
  80% {
    transform: translate(calc(-50%  - 396px), calc(-50% + 3px)) scale(1);
    background-color: #0D3B66;
  }
  100% {
    transform: translate(-50%, -50%) scale(1);
    background-color: #0D3B66;
  }
}
```

Figure 4: Example of CSS animation properties for one circular graphic element

but to change the (delta) position. Due to the correction of the centre alignment, the position change was calculated as 50 % of the size and the offset was subtracted from or added to that. The complete syntax for the animation is shown in Figure 4.

This procedure was repeated for all eight graphic elements, and elements were prepared in the HTML file to be animated.

2.2.2 Web Animations API code

This method is also based on writing the code, but some part of it is common to that of the CSS set. The HTML elements are the same, as is the basic CSS (Figure 5).

We have redefined the basic properties (size, position, colours ...) of the elements, only this time we have left out the animation property. This was written in JS. The animation part (in JS) consists of two parts of animation data: i) keyframes and ii) animation time and style.

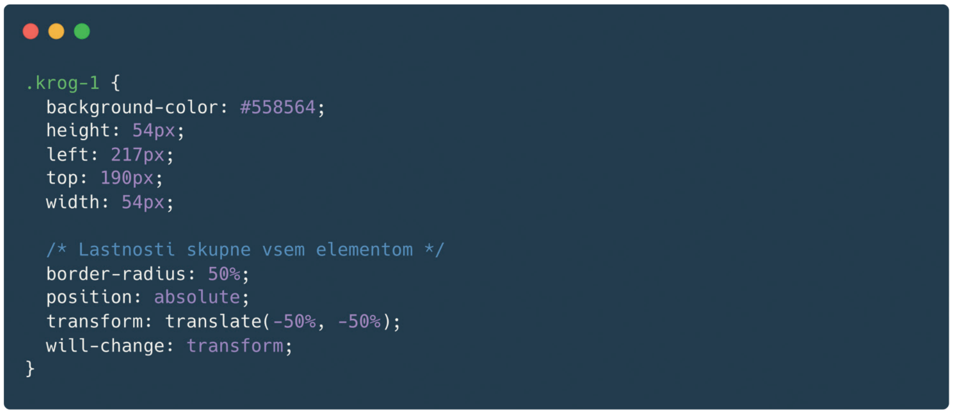
The animation data is unique for each element we want to move independently. The animation data can be global (as in our case, since we animate all elements at the same time and for the same amount of time) or unique for each element. We have defined six levels that match those of the CSS method, only the syntax was slightly different. This time, the percentages were written as a value between 0 and 1, for example, 0.6 instead of 60 %. The properties of the animation are shown in Figure 6.

In addition to the class, a unique identifier was added so that HTML elements could be animated. The IDs in JS were called and assigned to the animation properties, or the predefined variables were called. With that, the animation was determined.

2.2.3 Lottie animation code

Lottie animations are JSON files that we export from Adobe After Effects using the Bodymovin plug-in.

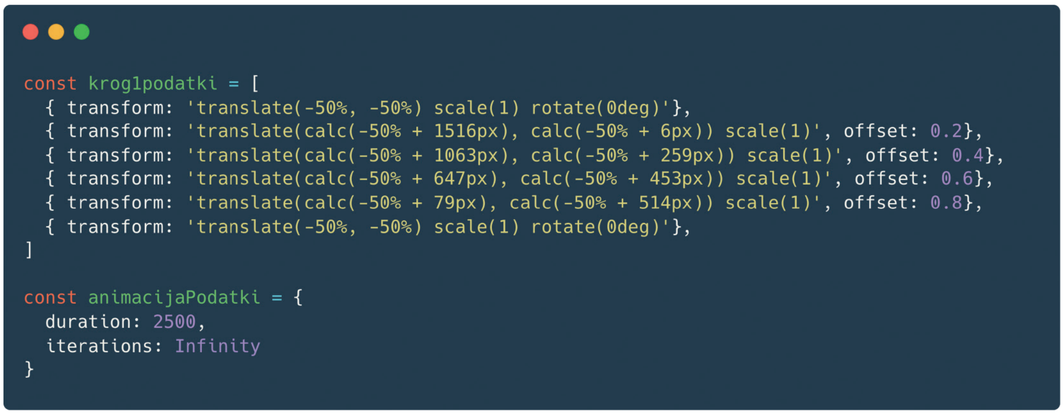
In After Effects, we created eight graphic elements and assigned them colours and start positions. In the timeline, we set keyframes, layers, position changes, and other properties for each element. After completing the animation in After Effects, we exported it to the JSON format using the Bodymovin plug-in. For the pur-



```css
.krog-1 {
  background-color: #558564;
  height: 54px;
  left: 217px;
  top: 190px;
  width: 54px;

  /* Lastnosti skupne vsem elementom */
  border-radius: 50%;
  position: absolute;
  transform: translate(-50%, -50%);
  will-change: transform;
}
```

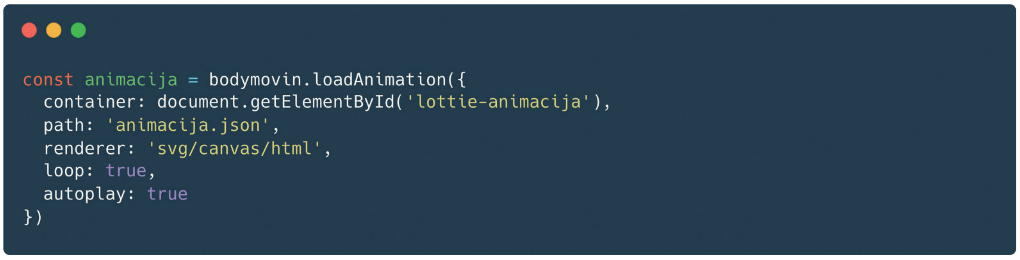
Figure 5: Example of CSS code for animation with WAAPI



```javascript
const krog1podatki = [
  { transform: 'translate(-50%, -50%) scale(1) rotate(0deg)'},
  { transform: 'translate(calc(-50% + 1516px), calc(-50% + 6px)) scale(1)', offset: 0.2},
  { transform: 'translate(calc(-50% + 1063px), calc(-50% + 259px)) scale(1)', offset: 0.4},
  { transform: 'translate(calc(-50% + 647px), calc(-50% + 453px)) scale(1)', offset: 0.6},
  { transform: 'translate(calc(-50% + 79px), calc(-50% + 514px)) scale(1)', offset: 0.8},
  { transform: 'translate(-50%, -50%) scale(1) rotate(0deg)'},
]

const animacijaPodatki = {
  duration: 2500,
  iterations: Infinity
}
```

Figure 6: Example of the JavaScript code configuration part for WAAPI animation



```
const animacija = bodymovin.loadAnimation({
  container: document.getElementById('lottie-animacija'),
  path: 'animacija.json',
  renderer: 'svg/canvas/html',
  loop: true,
  autoplay: true
})
```

Figure 7: JavaScript code with the Lottie animation settings

poses of this study, we chose a JSON file as the result of the rendering. To animate the JSON file on the website, a bit more code was required. We first imported a JS library that takes care of the execution of the animations and can convert the JSON file into SVG elements. We created an object that the animation binds to. We gave the library information about which element the animation should appear in, the path to the JSON file, the rendering mode, and other optional parameters such as loops and autoplay (Figure 7). The library also allows buttons to be associated with actions such as starting and stopping playback. These can be used in a playback environment for a better user experience but are not critical to the operation of the animations and do not change the display of the animations. With this code, the implementation of the Lottie animations was complete and ready for display in the browser.

2.3 Performance testing

After all the animations were implemented, the web pages with the animations were transferred to the server in a structure that allowed easy testing by type of animation and number of graphic elements. We created four identical animations for each of the technologies, with a different number of elements, stacked on top of each other.

When testing performance, we were inspecting three aspects: how many frames per second are displayed, how much graphics card RAM is used, and how much data is transferred from the server (Figure 8).

The base animation features eight shapes, with different parameters. As we wanted to better compare the different solutions, we increased the number of shapes by multiplying them and stacking them on top of each other in groups of eight. We created additional animations with 176, 368, and 560 elements. Those animations became complex enough for devices and browsers to not be able to display them optimally anymore. We looped each animation four times (each would play for 10 seconds) and repeated measurements three times for each browser and device.

The tests were performed in the following browsers: Microsoft Edge version 89.0.774.76 and Safari version 14.0.3 on a laptop MacBook Pro 15 (2018 – 2,2 GHz 6-Core Intel Core i7) with the operating system macOS 11.2.3 (Big Sur); on an iPad Pro 11 (2018) in Safari and Microsoft Edge (46.3.7); on a mobile phone iPhone 12 Pro with iOS 14.4.2 in Safari and Microsoft Edge (46.3.7); and on a low-budget phone Nokia 7.1 with Android 10 in Microsoft Edge (46.03.4).

The Nokia 7.1. device was tested to have a reference for a low-end device operating under the same stress of the animations.

Microsoft Edge was chosen because it is based on Chromium, has a modern rendering engine, and is widely available on many devices. The Chrome browser was not tested, as at the time of the research Microsoft Edge included better reporting of resource usage (more data in the report) compared to Chrome.

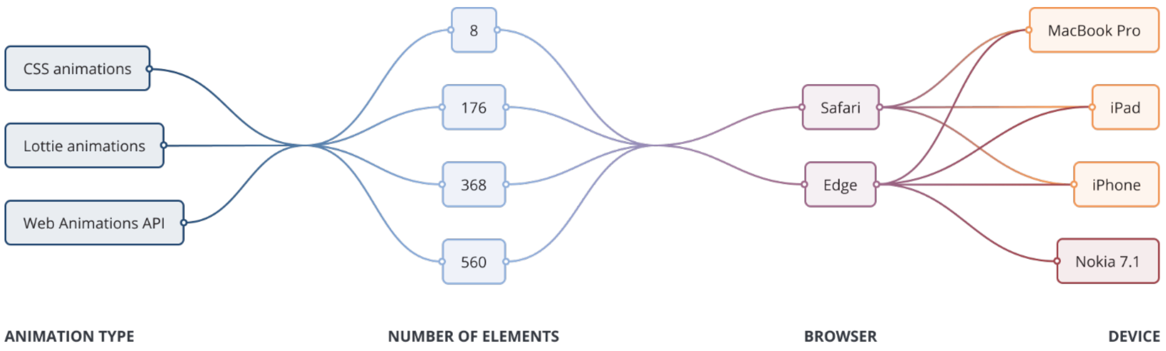


Figure 8: Parameters examined in the experiment

Safari was included in order to make comparisons with a browser that was already built into the system, and could theoretically perform better because of the deep integration of software with hardware.

3. Results and discussion

3.1 Performance

The frames displayed per second depending on play time are presented in Figure 9 as the average values for different devices in 560-element CSS, WAAPI and Lottie animations and in the Edge browser, and this is followed by explanations for the different devices used.

This was also the limit of what our devices could output. As expected, due to the lower end of the specifications with not very capable hardware, the Nokia phone displayed the smallest number of frames on average. For the largest number of elements, the iPhone displays the most graphic elements on average, followed by the iPad and MacBook. The minimum number of frames displayed per second was 9.7 and the maximum was 58.0. At 560 elements, the Nokia phone scored the worst with an average of 12.2 frames per second.

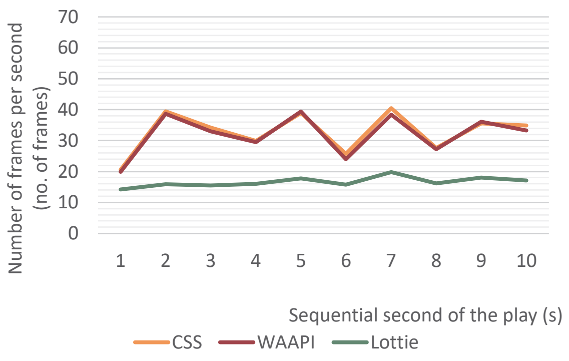


Figure 9: Average frames displayed on the devices per second depending on the play time with 560-element animations in the Edge browser

However, this device was very consistent in its display when compared with other devices. With the other devices, the variations are greater, averaging about 35 frames per second.

At 560 elements, the laptop (MacBook) plays the animation without major problems, only the frame rate is quite low. It ranges from 21.3 to 34.3 per second, but that is still enough for what appears to be smooth

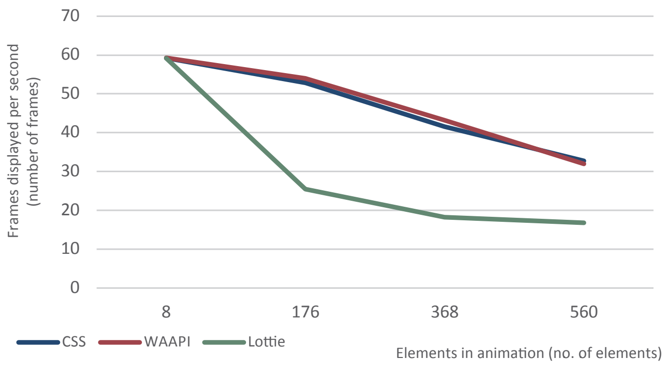


Figure 10: The average number of frames displayed per second depending on the number of elements in the animation for different types of animations in the Edge browser

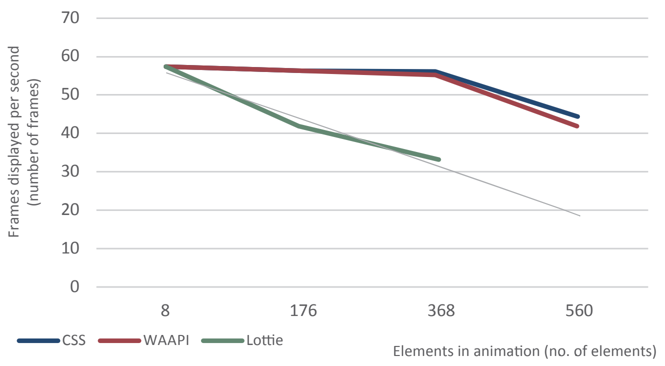


Figure 11: The average number of frames displayed per second depending on the number of elements in the animation for different types of animations in the Safari browser with added grey projection line

Table 1: An overview of the resource consumption of all animations

No. of graphic elements	**Memory usage (MB)**			**Amount of transferred data (kB)**		
	CSS	**WAAPI**	**Lottie**	**CSS**	**WAAPI**	**Lottie**
8	15.9	15.3	28.7	11.9	12.7	20.8
176	75.9	78.5	346.4	19.0	19.8	450.0
368	150.6	150.6	536.9	27.0	27.8	941.0
560	222.7	222.7	536.9	35.0	35.8	1 400.0

motion. When animating on a Nokia phone, however, the slow operation is very noticeable, as the elements do not move and only random inserts from the animation are displayed. The frame rate averages around five per second. On the iPad, trying to replay the animation caused the webpage to crash. On the iPhone, it did not work much better, managing to display the first four seconds at between 12 and 16 frames per second, then the page crashed.

In Figures 10 (Edge browser) and 11 (Safari browser), we can track the trend of decreasing quality by adding the number of graphic elements presented in the animation. A comparison of the results in Figures 10 and 11 shows that the decrease in the quality or the number of frames per second displayed is more severe in the Edge browser. This is particularly noticeable for CSS and WAAPI animations, where the number did not change or changed imperceptibly in the first three stages in Safari. The quality decreased slower for Lottie animations as well in Safari, compared to Edge, although the animation completely broke at 560 elements, whereas in Edge it could still render and play, even if at a low frame rate. For a better overview, we have added a projection line to the Safari graph (Figure 11), so we can predict how many frames per second the browser would be able to render.

Based on the results, we can also assume that the quality of the technologies themselves, based on the frames per second, declined. For a low number of elements, all the solutions chosen for creating the animations are appropriate, as they achieved similar results across the tested devices and two browsers. This was not the case with the increase in the number of elements displayed in an animation. As is evident in Figures 10 and 11, Lottie animations saw a dramatic fall in frames per second with the increase in the number of elements, so they are less suitable for complex animations. The CSS animations and WAAPI produced similar results throughout the experiment, so they are both viable choices.

When testing, we also monitored how demanding the technologies are for the hardware, measuring the amount of graphics card memory required for the rendering of the animations, as well as how much data was transferred from the servers as the page loaded. For the measurements of the amount of graphic card memory the report by the browser on a per-tab basis (GPU memory usage) was used.

Table 1 presents an overview of the resource consumption of all tested animation technologies. In this table showing memory usage and data transfer we can see that both CSS and WAAPI animations used very similar amounts of memory and data downloaded. In the Lottie animations we notice that with eight elements the situation is similar to that with the other two animation technologies, while for 176, 368 and 560 elements there is a clear rise of memory usage. With only 368 elements Lottie had no more memory available. With regard to data transfer, Lottie also took the lead. Where CSS and WAAPI animations downloaded about 35 kilobytes of data for 560 elements, Lottie transferred some 40 times more – 1400 kilobytes (1.4 megabytes).

3.2 Ease of build

All three methods have their advantages and disadvantages. One aspect that can be used to determine the usefulness of the solutions is their learning curve.

All three solutions use the three core technologies of the web – HTML, CSS, and JS. None of the animation technologies require advanced knowledge of the core languages, although WAAPI requires a little more knowledge of JS since it is based on that.

Lottie animations are the easiest to create from a design perspective, because the editor (After Effects) is visually oriented. This means that apart from a few clicks when saving keyframes, we can do all the designing by moving the shapes around, move around the timeline, change the properties of the shapes, and save the changes. The slightly harder part is implementing the animation in the website. However, this only requires a few lines of code. Lottie is therefore the first choice for animators and anyone who does not know much about coding. It is also the fastest method and allows for more complex animations that would take much more time to develop in CSS or WAAPI.

The other two methods rely entirely on programming, and the animator must be at least somewhat familiar with web technologies (HTML, CSS, and JS – the latter only in the case of using WAAPI). For beginners, CSS may be the better choice, as they only need to use two languages.

The CSS animations require some basic knowledge of inserting elements into Document Object Model (DOM) and using classes in CSS with @keyframe. The CSS animations and WAAPI animations work in a similar way, using keyframes depending on 100 % of the available time. We also need to set the playback time and the type of transformations separately. JavaScript opens up a vast number of additional possibilities for website development. Therefore, WAAPI is perhaps more suitable for advanced developers or complex websites, as it allows many manipulations with animations. The creation of an animation is much more time-consuming than simply drawing the changes, since every change has to be written in the code, while in After Effects we just move the element, and the program does the rest.

3.3 Compatibility

Lottie animations are based on the use of SVG elements and JS code, so they have very good compatibility with browsers. Only browsers older than Internet Explorer (IE) 8 cannot display them, but the market share of these browsers is less than 0.04 % at the time of writing. The CSS3 animations are also supported in all major browsers newer than IE 9.

Web Animations API is a newer technology and already supported by all modern and updatable browsers. Older versions of Chrome, Firefox, Edge and Safari cannot display them, however, nor can any version of Internet Explorer.

4. Conclusions

The results show that we can reject the first hypothesis, i.e. Lottie animations have the best balance between simplicity and performance, since we found that the balance between the simplicity and performance of Lottie animations is not the best among all the tested methods for creating animations, although their creation is extremely simple.

We can also reject the second hypothesis, i.e. WAAPI animations will consume more hardware resources than CSS animations. This is because CSS and WAAPI animations work very similarly, and thus their use of hardware resources is comparable. We can, however, confirm the third hypothesis. Safari, due to its integration with the hardware that is possible on Apple devices, can better maintain the quality of animations when they become more complex.

Based on the results, we can make some recommendations, and the overview of the creation and operation of the animations presented in this work can be used as a guide when choosing the right method for displaying animations online, although not all the available methods were tested in this study. We recommend Lottie animations as the most appropriate method for simple animations (e.g., turning the menu icon into a back arrow) because they are easy to create and implement in a visual environment.

Complex animations are the easiest to design in Adobe After Effects, as the tool is visually based. The created animation is easily exported into a JSON file needed for Lottie to work, and then implemented in a website. This ease of creation comes with some drawbacks, though. Many elements make high demands on device resources, so the fps might drop if we have too many, which can lead to a few dropped frames, creating a bad user experience. For more complex work CSS animations are best, because the quality is maintained as the number of elements grows. Production is more difficult as it relies solely on programming, but this method is valuable because smooth operation is paramount to the user experience. Although the operation is similar to WAAPI, we suggest choosing CSS as it is a general recommendation not to burden browsers with executing JavaScript code when an activity can be created in CSS with similar effort.

The research presented in this work provides a good insight into comparing the performance of different online animations and can serve as an example for further research in this area. The investigation could be extended to other common methods and libraries for creating animations, such as SVG animations, animations with the canvas element, WebGL and others, as this could give a more complete picture of the state of online technologies, while the recommendations would have a wider scope.

References

Adobe, 2021. *Adobe Flash Player*. [online] Available at: <https://get.adobe.com/flashplayer/about/> [Accessed August 2021].

Bassett, L., 2015. *Introduction to JavaScript object notation: a to-the-point guide to JSON.* Sebastopol, CA: O'Reilly Media.

Bellamy-Royds, A., Cagle, K. and Storey, D., 2017. *Using SVG with CSS3 and HTML5: vector graphics for web design*. Sebastopol, CA: O'Reilly Media, pp. 41–70.

Berners-Lee, T., 2021. *Tim Berners-Lee*. [online] Available at: <https://www.w3.org/People/Berners-Lee/> [Accessed August 2021].

Brundrett, A., 2016. Motion design: an intro to UX choreography. *User Experience Magazine*, 16(4). [online] Available at: <https://uxpamagazine.org/motion-design/> [Accessed August 2021].

Carrera, P., 2010. *Adobe Flash® animation: creative storytelling for web and TV.* Sudbury, MA: Jones & Bartlett Learning, pp. 85–113.

Chinnathambi, K., 2017. *Creating web animations: bringing your UIs to life*. Sebastopol, CA: O'Reilly Media, pp. 11–72.

Corvoysier, D., 2011. *Effectively measuring browser framerate using CSS*. [online] Available at: <http://www.kaizou.org/2011/06/effectively-measuring-browser-framerate-using-css.html> [Accessed August 2021].

Corvoysier, D., 2014. *fpsmeter*. [computer program] GitHub, inc. Available at: <https://github.com/kaizouman/fpsmeter> [Accessed August 2021].

Drasner, S., 2017. *SVG animations: from common UX implementations to complex responsive animation*. Sebastopol, CA: O'Reilly Media., pp. 15–66, pp. 77–86.

Ferreira, M., 2017. *The history of web animation.* [online] Medium. Available at: <https://medium.com/@milberferreira/the-history-of-web-animation-63b106c97fdf> [Accessed August 2021].

GreenSock, 2020. *The standard for modern web animation*. [online] Available at: <https://greensock.com> [Accessed August 2021].

Head, V., 2016. *Designing interface animation: improving the user experience through animation*. New York, NY: Rosenfeld Media.

McCullough, B., 2018. *How the internet happened: from Netscape to the iPhone*. New York, NY: Liveright.

MDN Web Docs, 2021. *Web animations API*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API> [Accessed August 2021].

Merleau-Ponty, M., 2013. *Phenomenology of perception.* Translated from French by D.A. Landes. New York, NY: Routledge, pp. 270–311.

Pierce, P., 2016. *UI animation – an ideal tool for immersive UX*. [online] UX magazine. Available at: <https://uxmag.com/articles/ui-animation-an-ideal-tool-for-immersive-ux> [Accessed August 2021].

Potter, M.C., Wyble, B., Hagamann, C.E. and McCourt, E.S., 2014. Detecting meaning in RSVP at 13 ms per picture. *Attention, Perception & Psychophysics*, 76(2), pp. 270–279. https://doi.org/10.3758/s13414-013-0605-z.

Pratt, J., Radulescu, V.P., Guo Mu, R. and Abrams, R.A., (2010). It's alive! Animate motion captures visual attention. *Psychological Science*, 21(11), pp. 1724–1730. https://doi.org/10.1177/0956797610387440.

Shamms Mortier, R., 1997. *Gif animation web magic*. Indianapolis, IN: Hayden Books.

Weyl, E., 2016. *Transitions and animations in CSS: adding motion with CSS*. Sebastopol, CA: O'Reilly Media, pp. 7–50, pp. 55–106.

W3Schools, 2021. *HTML attribute reference*. [online] Available at: <https://www.w3schools.com/tags/ref_attributes.asp> [Accessed August 2021].

Appendix

Table A1: Changes (in six levels) of the parameters of the circles: colour, distance from the left of the screen (dis. left), distance form the top of the screen (dis. top) and size

Level		Green	Orange	Dark blue	Light blue
1.	Dis. left (px)	218	1 618	844	1 180
	Dis. top (px)	190	440	744	478
	Size (%)	100	100	100	100
2.	Dis. left (px)	1 733	1 013	1 428	1 282
	Dis. top (px)	196	326	810	944
	Size (%)	100	131	100	100
3.	Dis. left (px)	1 280	431	1 697	253
	Dis. top (px)	449	361	716	848
	Size (%)	100	100	100	300
4.	Dis. left (px)	864	1 428	1 313	275
	Dis. top (px)	643	810	279	97
	Size (%)	100	23	100	100
5.	Dis. left (px)	296	1 695	450	1 733
	Dis. top (px)	704	612	747	196
	Size (%)	100	64	100	100
6.	Dis. left (px)	218	1 618	844	1 180
	Dis. top (px)	190	440	744	478
	Size (%)	100	100	100	100

Table A2: Changes (in six levels) of the parameters of the rectangles: colour, distance from the left of the screen (dis. left), distance from the top of the screen (dis. top), rotation and size

Level		Yellow	Red	Violet	Sandy
1.	Dis. left (px)	368	1 040	1 374	696
	Dis. top (px)	541	151	879	232
	Rotation (°)	0	0	0	0
	Size (%)	100	100	100	232
2.	Dis. left (px)	725	316	300	1 486
	Dis. top (px)	798	243	797	304
	Rotation (°)	31	63	240	360
	Size (%)	147	144	120	88
3.	Dis. left (px)	1 072	790	1 548	1 302
	Dis. top (px)	149	763	217	803
	Rotation (°)	−57	155	386	540
	Size (%)	56	180	135	122
4.	Dis. left (px)	1 708	233	739	440
	Dis. top (px)	542	536	184	874
	Rotation (°)	360	240	155	−57
	Size (%)	88	120	150	56
5.	Dis. left (px)	1 316	1 486	316	1 029
	Dis. top (px)	816	217	243	390
	Rotation (°)	540	386	63	31
	Size (%)	122	110	170	147
6.	Dis. left (px)	368	1 040	1 374	696
	Dis. top (px)	541	151	879	232
	Rotation (°)	0	0	0	0
	Size (%)	100	100	100	100