Measuring the performance of techniques for dynamic 2D animation in web browsers

M. BEŇO AND M. ÖLVECKÝ

Abstract

The study is evaluate the efficacy of diverse 2D animation techniques on web pages, with a particular emphasis on dynamic CSS variable notation (in single-line and two-line formats), jQuery, anime.js, and Velocity.js. The analysis entailed the translation of objects within an abstract model of C programming comprehension, with the objective of measuring the average execution time and the variance across multiple web browsers. The findings indicated that single-line CSS variable notation was the most efficient, exhibiting superior performance compared to two-line notation by 44.07% and 48.91% for 1,000 and 50 objects, respectively. It is noteworthy that anime.js exhibited the least efficient performance with 25 objects, demonstrating a 49.66% discrepancy in performance. Native browser technologies demonstrated superior outcomes compared to JavaScript libraries, which exhibited slower processing times despite being perceived as high-performance solutions. Despite its reputation as an outdated technology, jQuery exhibited competitive rendering speeds. The findings of this study underscore the significance of optimization in animation techniques and rendering efficiency.

Mathematics Subject Classification 2000: 81U30
Key Words and Phrases: 2D animation, CSS class variables, JavaScript library, code processing speed.

1. INTRODUCTION

The creation of interactive animations, whether two-dimensional (2D) or three-dimensional (3D), on websites represents a significant opportunity to capture the attention of a typical internet user. In this context, the term "web animation" is used to describe a visual change of elements that occurs over time. Animation creates a specific optical illusion of movement, transforming a static page into a dynamic one with interactive elements. In addition to interactivity, other key when creating web animation include the dynamism of the web page, enhancement of user experience, and optimization for display in the web browser. The use of animation on web pages has facilitated the visualization of a range of phenomena, not only in the context of learning, but also as a means of presenting data in the form of interactive graphs and diagrams. Animations are frequently employed in the context of animated presentations, interactive narratives, and marketing initiatives, where they are used as visual banners. In recent years, the World Wide Web has undergone a series of

10.2478/jamsi-2024-0009

transitions in the design and creation of interactive animation. The initial phase, referred to as Web 1.0, was based on HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) technologies. The main animation formats used were SWF (Shockwave Flash Movie) and GIF (Graphics Interchange Format). In this context, the term "Web 1.0" is used to describe static HTML pages. The estimated load time for a GIF animation is 5-30 seconds, although the exact timeframe depends on the complexity and size of the animation. SWF animations took slightly longer to load than other formats due to the longer loading time of the plug-in (Flash Player) and the content itself. The loading time for SWF animations ranged from 10 to 60 s. These times can also be attributed to the available connection speed, which ranged from 56 kbps to 1.5 Mbps. The second stage, Web 2.0, is distinguished by the dynamic presentation of user-generated content. CSS3 and JS library capabilities were often used for animation purpose. This stage emphasized user interaction, with a particular emphasis on the user experience (UX). The loading speed of the animations on the website was considerably slower than that of the other animation types, due to the technology used. The load time for SWF animations was in the range of 2-15 seconds, for JS animations 1-5 seconds, and for CSS3 animations less than 1 second. This discrepancy was likely due to the speed of the internet connection, which was in the range of 1-10 Mbps. The current stage of web development is considered to be Web Version 3.0, which is referred to as the Semantic Web. Technologies for animation are primarily sophisticated techniques for creating two-dimensional and three-dimensional animations, as well as WebGL (Web Graphics Library) and the incorporation of virtual reality elements directly into web pages. These web pages are mainly personalized and often include elements of artificial intelligence. The loading speed of a web animation varies depending on the technique used. For CSS3 animations, the loading speed was set to 100 ms. For WebGL, the loading speed depends on the animation complexity ranging from 0.5 to 3 s. For the JS animations, the loading speed ranged from 1 to 10 ms. Thus, animations are a fundamental aspect of human-computer interaction, on which the impact of serious games on humans is currently being investigated. A number of studies are currently being conducted to examine user experience and

interaction. These studies indicate that animations on web pages can be used effectively when appropriate optimization methods are employed (Shekhar, 2023), (Hostovecky, 2024), (Valentova, 2023), (Jurinova, 2023).

1.1. Related works

Computer-generated animations are employed in a variety of contexts to enhance the comprehension of complex concepts and to illustrate the functioning of intricate systems. These include educational programs, medical procedures, mechanical processes, engineering projects, and other areas of study. The findings of the studies conducted by Hanif (2020), Hu (2021), and Chen (2020) support the assertion that the effective integration of computer animation in educational settings can facilitate the improvement of both knowledge and skills. This is because learners who engage with these animations can grasp the concept of dynamic visualization of cognitive processes. However, as evidenced by the findings of the research conducted by Beňo and Ölvecký (2019), improvements in knowledge and understanding do not necessarily equate to an increase in knowledge retention. In addition to the aforementioned findings, computer animation has been identified as a highly effective technique, particularly in the online domain, as it can facilitate an immersive learning environment.

The findings of Ritonga et al. (2020) show that the use of computer animation enhances critical thinking skills of students as they prepare for their professional careers. Furthermore, the results of the research conducted by Ahmad (2021), Alsalhi (2023) illustrate the effectiveness of incorporating computer-generated animation into educational frameworks to enhance cognitive competencies and critical analytical skills at all levels of education, including professional development initiatives. Similarly, experiments conducted by Okamoto (2024) and Loknar (2023) demonstrated that animations that use single-line CSS variable notation consume on average 30% less CPU time than anime.js and Velocity.js. Our experiment produced similar results for the rendering speed of 2D animations.

2. IMPLEMENTATION DETAILS

One of the fundamental aims of this paper is to highlight the establishment of a framework that contains the necessary elements, which can then be transferred and transformed as required. A significant consideration is the accessibility of the source code, which can be continuously improved and refined. The results of the experiment conducted by Iegorova (2020) indicate that one of the current challenges in creating interactive 2D animations on websites is optimization in terms of fast loading as well as overall performance. There are several solutions that use appropriate tools, taking into account the target requirements and time optimization, to address the animation of objects on a page. In particular, the following are some fundamental techniques for creating the structure of elements:

- a static solution using a CSS class
- a dynamic solution using a CSS class and by sending a CSS variable to the class from the JavaScript environment
- a solution that uses native jQuery functions directly
- a solution that uses jQuery in conjunction with animation libraries (Sharples).

Each of these techniques has its advantages and limitations, which must be considered when creating 2D animations. This paper presents specific solutions and describes their optimizations for display in different web browsers. This paper concludes by outlining the main factors that influence the performance and speed of interactive animations on web pages. The experimental results can be used by web developers, user interface designers, and other professionals to improve the rendering of 2D animations on web pages and ensure compatibility across multiple browsers and mobile devices.

To gain a deeper understanding of the aforementioned methods, it is essential to not only to describe their fundamental principles in theoretical terms but also to analyze their application in specific practical scenarios. This analysis will help to identify the strengths, shortcomings or limitations of each approach. Through a detailed examination of specific solutions, it is possible to illustrate how these

techniques work in practice while providing insights that can inform further development.

2.1. The jQuery environment using animate()

In terms of openness (mentioned and explained above) and simplicity, the jQuery library was used for the basic manipulation of DOM (Document Object Model) objects. jQuery has a simple and intuitive interface as well as an extensive ecosystem of add-ons. However, it is characterized by its ease of use and a language syntax that is suitable even for beginners. It allows developers to easily select elements and manipulate the DOM. An important aspect is that jQuery has extensive browser compatibility. It supports a wide range of web browsers, including older versions of Internet Explorer. Nevertheless, its development is stable, the community support is large enough and new features that support current trends are gradually incorporated into new versions (Dubey, 2024; Su, 2021).

The jQuery.animate() method allows developers to create dynamic animations on web pages. The basic syntax is as follows: $(selector).animate({properties}, duration, easing, complete). The advantages of using animate() are mainly in the intuitive API, suitable for beginners. However, jQuery.animate() also has some limitations, especially in the performance of rendering complex animations, especially on less powerful devices, the lack of hardware acceleration (CSS animations can take advantage of this acceleration), the size of the entire jQuery library, especially in terms of downloaded data. Despite these limitations, there are some techniques that can overcome them, in particular by using different requests, minimizing DOM changes, taking advantage of CSS transformations (especially in terms of hardware acceleration), and last but not least by optimizing selectors (Dubey, 2024; Su, 2021).

2.2. Animation libraries

In response to the increasing demand for dynamic web content, a number of animation libraries have been developed to create 2D and 3D animations. Some of the most widely used are GSAP (GreenSock Animation Platform), Anime.js,

Velocity.js, Popmotion, Three.js, and AniJS. Based on the experimental results, the Anime.js and Velocity.js libraries were selected for further analysis. It should be noted, however, that their use may be limited to specific contexts due to inherent limitations in their design. Nevertheless, they were considered sufficient for the optimization requirements of this study.

2.2.1. Anime.js

Anime.js is a lightweight animation library that operates based on a single robust API (Application Programming Interface). It can be used to animate HTML through the use of JS, CSS, SVG (Scalable Vector Graphics) and DOM attributes. Notable features of this library include its versatility (it supports animation of CSS properties, DOM attributes, and JS objects), compact size (approximately 16 KB, facilitating fast web page loading), straightforward syntax, and sophisticated features (timelines, nested animations, and control over animation progress).

The use of the anime.js library in conjunction with CSS transformation features allows for the optimization of 2D animations, a process that is made possible by hardware acceleration. The library is able to animate groups of objects with minimal impact on overall performance. It can synchronously control the playback of the animation itself and offers a number of other features. The strength of this library lies in its up-to-dateness, with new versions being released regularly by the developers on GitHub (Gotskaya, 2019).

2.2.2. Velocity.js

Velocity.js is a synthesis of the most effective elements of jQuery and CSS transitions. Despite a lower rating on GitHub compared to anime.js, it is widely used by users of WhatsApp and Mailchimp. In addition to its basic animation capabilities, the main strength of Velocity.js lies in its capacity for hardware acceleration. Velocity.js able to facilitate the navigation of web browser windows. It can work in conjunction with the jQuery library that has been loaded in the browser, or alternatively, it can work independently with Vanilla JS. It is also able to undo previous animations. The advantage of Velocity.js is that it uses a custom animation stack that replaces $.animate() jQuery instances with velocity $.animate(). This

approach provides a performance boost on web browsers and mobile devices. However, a disadvantage of this library compared to anime.js is its lack of recent updates, with the last version being four years old (Steyer, 2022).

2.3. CSS3 classes with transform function

The transform CSS property allows the element to be rotated, scaled, skewed, or scrolled. In the experiment, only simple animations were investigated, mainly the translation (sliding) of a given element in the horizontal plane. Therefore, only an example of the translateX transform function, which has the following form, would suffice:

transform: translateX(t_x);

or the equivalent notation transform: translate(t_x, 0),where t_x represents the displacement value in relative or absolute units (Steyer, 2022).

In the case of using a notation that represents all possible kinds of transformations, there is a solution in CSS3 in the form of a transformation matrix of the form:

transform: matrix(a, b, c, d, t_x, t_y),

where the letters a, b, c, d represent the numeric value of the linear transformation description and t_x, t_y represent the translation to be applied.

Note that in the DOM document, as in graphical editors, the zero point is located in the top left-hand corner. If vertical scrolling needs to be addressed, scrolling down is addressed by increasing the value of p_y. Positive coordinates are below and to the right of the origin, while negative coordinates are above and to the left (Mishra, 2021).

2.3.1 Use of variables for passing values to the CSS3 class

In CSS, variables are defined entities with specific values that can be reused in a document. They can be defined using the @property at-rule or syntax with a property (e.g. --primary-color: blue;). They are accessed using the var() function.

The big advantage is usability, readability and semantics. They allow you to define a value in one place and then refer to it in multiple places. Properties defined using two hyphens (--) are subject to cascading and inherit their value from their parent (Mishra, 2021).

According to (Semerádová, 2020), several factors influence the rendering and transformation performance of 2D animations:

a) Number of objects per animation type:
   - Simple animations typically contain up to 10 objects. These 2D animations include button movement, text animation, rotation, color change and so on. They are less demanding on computing resources.
   - Complex animations contain a large number of objects (tens to hundreds). Typically, they are used for chain reactions or transitions between scenes. The aim is to increase interactivity or achieve visually complex animation.

b) System performance:
   - The implementation of real-time rendering is a significant factor in enhancing user experience, particularly with regard to the smooth playback of animations. For simple animations, it is crucial to optimize the number of objects to a maximum of 100 to minimize the load on the web browser. Rendering of additional objects increases the demand for system resources.
   - Optimization is a crucial factor in enhancing performance, including the use of hardware acceleration (particularly in the context of graphics acceleration), the simplification of geometric elements, or the processing of objects in a single render cycle. These techniques enable the handling of a greater number of objects without a notable decline in computational efficiency.

c) Technologies used:
   - For simple animations with a limited number of objects, it is recommended to use CSS transformations. This technology uses hardware acceleration, facilitated by the GPU Assistant, which significantly improves overall performance.

- Canvas and WebGL are optimal for animations with a large number of objects. However, these technologies require more effort and optimization from the developer.

3. PRACTICAL SOLUTIONS FOR 2D ANIMATIONS

In order to implement the transformation (Figure 1), it was necessary to determine how to label the DOM object. Since objects are created dynamically, the use of a direct identifier in the DOM hierarchy was selected as a means of obtaining and reusing the unique name of a given DOM element.

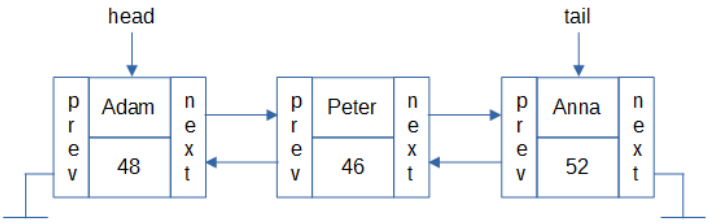

Figure 1. Dynamically created objects

The unique identifier (ID) solution was found to be feasible because the animation is performed only on the entire DOM element. However, once a nested part of a given element, or a nesting in the hierarchy of a given element, is selected, the result is that the use of a given library does not result in the elements being moved by a given value.

The solution to this problem was to label the given element with a new class and to use a pseudo-class of the position of the nested element, namely the pseudo-class nth-of-type(n), where n is the indexing number. The pseudo-class compares elements based on their position among siblings of the same type (tag name) (Steyer, 2022).

Note that the implementation of :nth-selectors in jQuery is strictly derived from the CSS specification, starting indexing at number 1. However, for other selectors, such as first() or eq(), the indexing is similar to jQuery or JavaScript, starting at 0 (Steyer, 2022).

3.1. Using jQuery library with animate()

The jQuery library contains a dedicated function, animate(), which is specifically designed for trace formations (see Figure 2). One disadvantage of this solution is that jQuery does not directly support the transform function, instead replacing it with the left function. The advantage of this solution is that the meaning is preserved, but the disadvantage is that the offset size is reset. For example, if a scroll value of 50px is specified, the only way to reset the element in question is to call the left function again with the same value but a negative sign. It is not possible to use the ternary operator to return a specified value. Given the intricate and comprehensive nature of the subject matter, it is pertinent to acknowledge that direct transformation via the transform function is feasible by incorporating the jquery.transform2d.js library. However, due to its considerable size, it was not used in the experimental process.

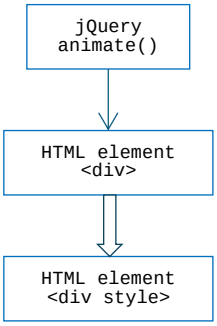

Figure 2. 2D animation with jQuery

The proposed and tested solution:

```
$('.elm:nth-of-type(n)').animate({
    left: "+=50"
}, 0,);
```

The advantage of using the jQuery library is that in the case of multiple elements, each using the loop function, both a class and a unique identifier can be used to label an element:

```
let arrayH = $("#elms").children();
$.each(arrayH, function(index){
    $(this).animate({
    left: "+=50"
    }, 0);
});
```

The solution results in an embedded inline style directly in the HTML element, e.g. on element 2:

```
<div class="elm" id="2" style="left: 50px;">
```

As a side note, jQuery as well as Vanilla JS can directly call the CSS transformation function.

```
$('.elm:nth-of-type('+i+')').css('transform', 'translateX(50px)');
```

While the result is the same as calling a CSS class with a given transformation function, it's a workaround for jQuery's element animation function.

3.2. Using anime.js and Velocity.js

The second option is to use an animation library, in this case the anime.js and Velocity.js libraries. anime.js (Figure 3) is the most widely used animation library for 2D transformations, but Velocity.js is optimized directly for jQuery.

The advantage of this solution is that you can design and create even more complex animations without having to worry about solving matrix notations or other helper functions. The libraries provided are robust and very easy to learn and intuitive to use. In addition to animating individual elements, Anime.js allows you to call an array of elements and then perform transformations on multiple elements at once. It also comes with good documentation and examples that make it easy to use the features. The disadvantage of this solution is the passing of individual parameters to function libraries and the subsequent processing of animations. This increases the execution time of individual animations.

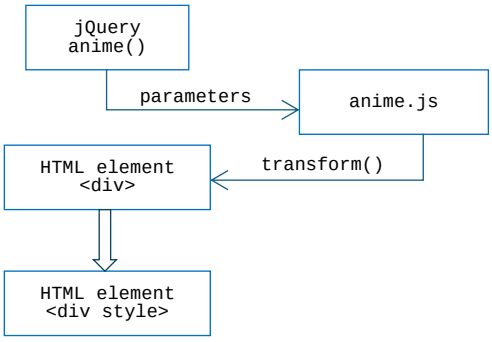

Figure 3. 2D animation with Anime.js

The difference between Anime.js and Velocity.js in using transform functions is that Anime.js strictly uses the transform function, but Velocity.js can also use the left function in addition to this function, similar to jQuery. However, using the transform function has the advantage that if you need to reset an element to its original setting, you can just use the ternary operator with a null value setting. Another way to deal with element style nulling is directly from the jQuery environment, where we specify an empty set ($('.element:nth-of-type(n)').attr('style','');)) when defining a style. One of the advantages of using this solution is performance optimization, as there is no need to call the animation library again. This optimization can have a significant impact on the speed of animation rendering (Gotskaya, 2019).

The proposed and tested solution with anime.js:

```
anime({
    targets: '.elm:nth-of-type(n)',
    translateX: 50,
    duration: 0
});
```

As illustrated in the example, anime.js directly handles the loading of the element within the target item, and it is not feasible to link the library functions directly to the element. Velocity.js, on the other hand, overcomes this problem by calling the library functions after defining the element to which the transformation functions will be applied.

The proposed and tested solution with Velocity.js:

```
$('.elm:nth-of-type(n)').velocity({
    translateX: "50px"          //left: "+=50"
},0);
```

The solution results in an embedded inline style transformation directly in the HTML element, e.g. on element 3:

```
<div class="elm" id="3" style="transform: translateX(100px);">
```

Using the left function within the library gives a similar result to the animate() function in jQuery. However, this approach is not without its drawbacks. Primarily, it requires the use of a class with position selectors when defining the DOM element to which the transformation function will be applied.

3.3. Using CSS class

Cascading Style Sheets (CSS) are an essential part of the process of creating web pages and applications. They modify the visual appearance, obfuscate or make elements invisible, and also allow for the animation of transitions from one CSS style configuration to another. The class is comprised of two distinct components: a style, which defines the CSS animation, and a set of key-frames, which define the initial and final states of the animation style, and potentially also the intermediate trace points.

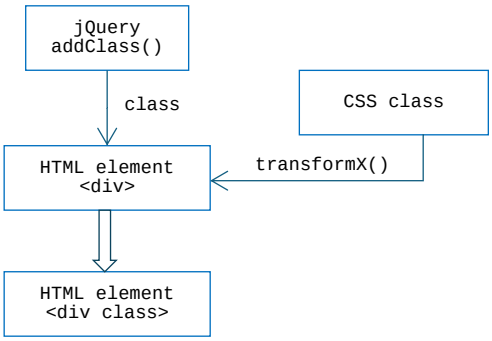

Figure 4. 2D animation using CSS class

The advantage of this solution is that there is no need for a third party library, be it jQuery or JavaScript. CSS classes are easy to use for animations (Figure 4), they are able to handle even moderate system loads effectively, and can also take advantage of hardware acceleration to render animations (Steyer, 2022). In general, simple animations can be defined with a few lines of CSS and don't require a large amount of knowledge to implement. The opposite is true for complex animations, where just a basic knowledge of CSS is no longer sufficient. However, as mentioned above, this is an acceptable option for simple animations.

The solution is very simple, just assign a CSS class to the DOM element that will contain the transformation function.

```
$('.elm:nth-of-type(n)').addClass("elmanim");
```

with the definition of a new CSS class:

```
.elmanim{
    transform: translateX(50);
}
```

This is a very practical solution, as no additional inline style of the transformation function is created for a given element, only a new CSS class is defined.

```
<div class="elm elmanim" id="2"">
```

Furthermore, cancellation is a straightforward process, simply removing the class from the element using the removeClass() function. However, a potential drawback is that the size of the value is directly written into the class, which could be addressed by using CSS variables. This solution offers a similar advantage to that of jQuery's animate() function, in that it allows for the use of both unique identifiers and classes with selectors to target the element.

3.3.1. Using a CSS variable

In the event that a dynamic solution is required, it is essential to exercise control over the transfer of values to the transformation function. The implementation (Figure 5) makes use of a newly defined variable within the CSS class, using the keyword "var" (with the name defined subsequently by the use of two hyphens). The value is passed by defining the variable numerically in the jQuery environment and assigning it to a DOM element. It should be noted that jQuery allows the values to be passed simultaneously on a single line, eliminating the need to call the same element twice and reducing transformation time.

```
$('.elm:nth-of-type(n)').css("--translX", 50).addClass("elmanim");
```

A CSS class using a variable and a transformation function will take the form:

```
.elmanim{
    transform: translateX(var(--translX));
}
```

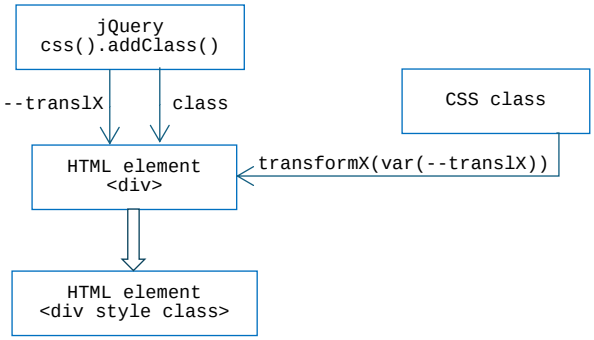

Figure 5. 2D animation with CSS variable

This is a very practical solution where, for a given element, the definition of a new CSS class simply creates an additional inline style variable with a given value.

```
<div class="elm elmanim" id="2" style="--translX: 0;">
```

This solution has the advantage that if you need to retrieve the original position of the element, you can simply remove the CSS class using removeClass() without the need for additional adjustments to individual transformations or styles, or resetting the value of the CSS variable.

```
$('.elm:nth-of-type(4)').css("--translX", 0)
```

Given the focus of the experiment on dynamic animations, i.e. those with the ability to change parameters, the use of static animations via the CSS class was deemed unnecessary and thus excluded from further consideration.

4. THE METHODOLOGY USED TO TEST CODE EXECUTION SPEED

To maintain consistency in the test environment and ensure uniformity of the test conditions, testing was performed on the localhost network in the private window of each web browser. The computer was not connected to the Internet and additional services were disabled, as these could introduce inconsistencies into the testing process. This approach allowed us to avoid problems related to connection speed and other effects of services and programs that may alter animation speed.

For the testing phase, we selected the most common and currently used open-source web rendering engine, namely web browsers.

- Chrome and Brave using the Blink rendering engine
- Firefox using the Gecko rendering engine
- Vivaldi using the Goana rendering engine (a fork of Webkit).

The selection of web browsers was based on the current usage rates of browsers based on the aforementioned rendering engines. A series of tests were carried out on a selected number of objects. To find out the effectiveness of the animation library solution compared to the native solution using or the solution using dynamic variable CSS, tests were performed on both a larger and a smaller number of objects. Based on the results, it is possible to anticipate the behavior of individual engines using other simple transformation tools.

A total of 100 iterations were performed for each animation solution. At the end of each trial, the resulting data was entered into an array, from which the mean animation time, median, and variance were calculated. The variance value was used as a control value to eliminate erroneous results. If the variance of the values of a solution method exceeded a predetermined threshold, the test was repeated with the new values, and the results were included in the analysis. The thresholds were as follows: 100 objects, variance > 1; 200 objects, variance > 2; 300 objects, variance > 5; 500 objects, variance > 10; 800 objects, variance > 15; and above 800 objects, variance > 30. To increase objectivity, each test was repeated five times, and the result corresponding to the mean of the measured variables (median) was selected for processing. The median is advantageous because, unlike the mean, it is less susceptible to significant deviations when there are outliers in the dataset. The general formula for the values of X and the size of N for an even number of members has the form $Me(X) = \frac{X_{(N/2)} + X_{(N/2)+1}}{2}$, for odd $Me(X) = \frac{X_{(N/2)}}{2}$

The hardware configuration to be tested included the following system parameters: The system under test comprised an AMD Ryzen 5 4600 H processor, an Nvidia GeForce RTX2060 graphics card, 8 GB of RAM, and a Windows 10 operating system.

To facilitate the interpretation of the measurement results, two graphs are presented, each showing the overall result. The first graph shows the solution with a higher number of objects (above 200), while the second graph shows the solution with a lower number of objects (below 200). In the first graph, the number of objects is continuously reduced by 100, whereas in the second graph, the reduction is by 25 objects, allowing for a more detailed examination. This figure shows the improved performance of engine browsers at lower numbers, which is sufficient for most solutions.

If one of the most critical characteristics of Internet applications is their speed, it is therefore important to focus on the results of testing this characteristic and to evaluate the results obtained according to the technologies chosen. Since all the proposed and tested solutions produce inline stylesheets, it was interesting to see what results would be obtained. For the sake of interest, the tests were extended to include the two-line CSS variable notation, i.e. setting the value of the CSS variable in the DOM object + adding a CSS class to the given object. This solution simulates writing in Vanilla JS environment, whose solution is written in the comments of the above code:

```
$(‘.elm:nth-of-type(n)‘).css(“--translX“, 50).
                    //element.style.setProperty(‘--css-variable‘, value);
$(‘.elm:nth-of-type(n)‘).addClass(“elmanim“);
                    //element.classList.add(“my-class“);
```

Based on the results of these tests, it was then decided which technology would be used to solve the transformation. Garaizar (2019) claims that using the browser to directly execute DOM animations, rather than manipulating inline stylesheets via JavaScript, is a significantly more expedient and efficient approach. However, specific values and other relevant statistics are not presented. Given the presence of an inline style in each of the individual dynamic solutions for the proposed and tested solutions, it is important to observe the overall results after testing.

For testing purposes, the JavaScript function performance.now() returns a timestamp in milliseconds (ms). The result of the test represents the difference in the elapsed time between the timestamps (Garaizar, 2019).

```
const t0 = performance.now();
transform()
const t1 = performance.now();
```

5. MEASUREMENT RESULTS AND DISCUSSION

In contrast to the research reported in the introduction to this paper, the present study focuses on the technical implementation of computer animation with performance implications for selected object creation and transformation in selected web browsers web browsers for object creation and transformation. The results of the research by Yan et al. (2021), Mendki (2020), Macedo et al. (2021), and Okamoto (2024) showed the rendering of web animations using WebAssembly in comparison with JavaScript, when optimized on different hardware platforms (e.g., ARM processors) in terms of the efficiency of the use of operating memory, showed comparable results to our experiment. In contrast to WebAssembly, rendering 2D animations using a single-line CSS variable notation was better than the other methods than the JavaScript libraries anime.js and Velocity.js. The results show that WebAssembly uses more memory than JavaScript. WebAssembly also renders animations faster. We also see different results compared to our study when we use Node.js to measure the performance of animations with our Applied Javascript Performance (now) methodology. A comparison of the research conducted (Loknar, 2023; Li, 2020; Goukouni, 2022) and this study shows that in object creation and transformation using WebGL. we can compare the performance of rendering and animating objects alone with js animation techniques. This trend of comparing other methods with JS libraries has been analyzed in detail in several research papers. The experiments in this study focus on other ways to optimize 2D animation rendering, such as single and double line CSS variable notation, or the use of jQuery compared to JS libraries primarily designed for animation in web pages. Successive

measurements were made on selected browsers including Chrome, Brave, Vivaldi and Firefox.

5.1. Testing on the browser Chrome (engine Blink)

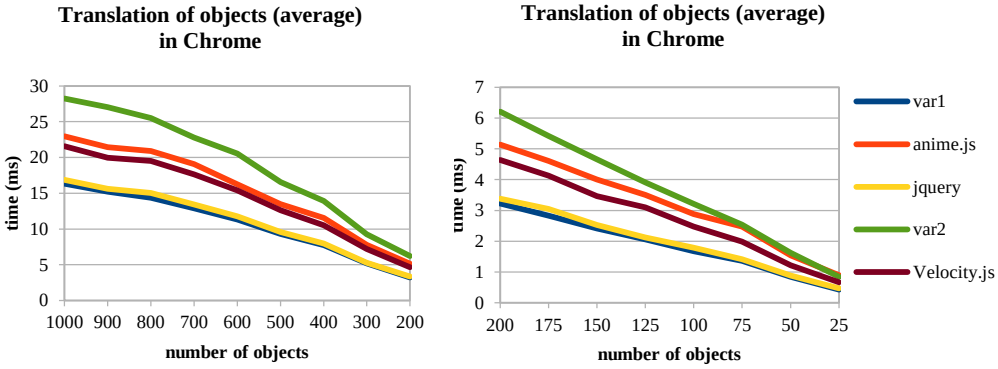

Figure 6. Graphical representation of the transformation in the browser Chrome

Table I. Speed measurement results (in ms) as a function of the number of rendered objects in the browser Chrome

Chrome	**1000**	**900**	**800**	**700**	**600**	**500**	**400**	**300**	**200**	**175**	**150**	**125**	**100**	**75**	**50**	**25**
var1	16.35	15.23	14.37	12.91	11.34	9.33	7.77	5.19	3.22	2.83	2.41	2.06	1.67	1.36	0.84	0.43
anime.js	22.97	21.42	20.89	19.06	16.23	13.48	11.55	7.78	5.14	4.61	4.0	3.5	2.88	2.48	1.54	0.89
jquery	16.89	15.61	15.02	13.45	11.77	9.59	7.99	5.29	3.38	3.05	2.53	2.12	1.79	1.41	0.88	0.47
var2	28.22	27.01	25.5	22.77	20.54	16.55	13.92	9.28	6.21	5.42	4.66	3.91	3.21	2.54	1.62	0.84
Velocity.js	21.55	19.97	19.5	17.63	15.4	12.61	10.53	7.2	4.64	4.13	3.46	3.1	2.48	1.99	1.22	0.67

The first browser tested was Chrome with the Blink engine. Figure 6 shows that the difference between using animation libraries and single-line notation increases as objects are added. The animation libraries anime.js and Velocity.js maintain a roughly constant difference in a given browser (from 1.42 ms for 1000 objects to 0.22 ms for 25 objects). However, a very interesting result (Table I) was obtained when using less than 75 objects in the form of a two-line CSS dynamic variable notation (2.54ms for 75 objects, the other techniques tested achieving lower values). As shown in Figure 2, the anime.js solution started to blend into the two-line CSS

variable notation. For example, at 1000 objects the difference was 5.25 ms, at 500 objects 3.07 ms, at 100 objects 0.33 ms, while at 25 objects anime.js was already 0.05 ms slower than the two-line notation. With less than 25 objects, the anime.js animation library was the worst solution with 11.87 ms. The best solution in all tests was to use the single-line CSS dynamic variable notation, i.e. when rendering objects from 25 to 1000. The difference for 1000 objects was in the range of 0.54-11.87 ms, and for 25 objects it was in the range of 0.04-0.46 ms. The measured differences were similar to those reported in other studies (Okamoto, 2024; Loknar 2023). A surprise was the result of the native JavaScript function of the jQuery Animate() library, which was slightly slower than the single-line dynamic variable CSS notation (from 0.54 ms (1000 objects) to 0.04 ms (25 objects)). However, we expected it to be the fastest, even at the cost of the setup reset issues mentioned in the previous section.

5.2. Testing on the browser Brave (engine Blink)

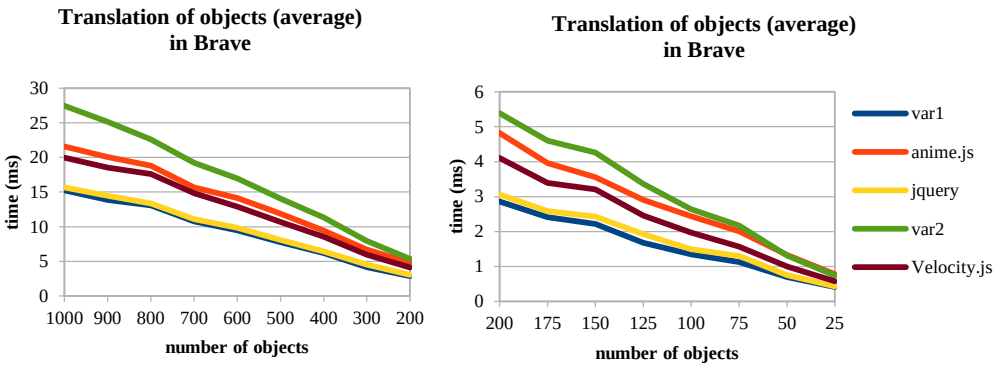

Figure 7. Graphical representation of the transformation in the browser Brave

Table II. Speed measurement results (in ms) as a function of the number of rendered objects in the browser Brave

Brave	**1000**	**900**	**800**	**700**	**600**	**500**	**400**	**300**	**200**	**175**	**150**	**125**	**100**	**75**	**50**	**25**
var1	15.25	13.88	13.06	10.77	9.5	7.78	6.21	4.18	2.86	2.41	2.22	1.68	1.35	1.13	0.7	0.41
anime.js	21.58	20.07	18.8	15.67	14.14	11.89	9.45	6.73	4.82	3.96	3.56	2.91	2.44	2.01	1.33	0.78
jquery	15.67	14.45	13.37	11.09	9.87	8.08	6.42	4.55	3.06	2.59	2.43	1.93	1.5	1.3	0.75	0.43
var2	27.44	25.15	22.6	19.21	16.95	14.03	11.35	7.96	5.39	4.6	4.26	3.36	2.64	2.17	1.31	0.74
Velocity.js	19.97	18.53	17.61	14.82	12.9	10.66	8.54	5.99	4.11	3.39	3.21	2.46	1.97	1.57	1.0	0.58

To corroborate the results of the Blink engine, an additional browser was selected for analysis: the Brave browser (Figure 7), which has robust security features as standard. Nevertheless, the results (Table II) of the comparative analysis showed that the discrepancies between the different techniques were similar to those observed in Chrome. The optimal solution was the use of a single-line dynamic variable CSS notation, followed by the use of a native jQuery function (from 0.42 ms (1000 objects) to 0.02 ms (25 objects)). The native jQuery function was followed by the Velocity.js JavaScript libraries, with anime.js showing almost identical performance throughout the test period. In contrast, the two-line notation of the dynamic CSS variable showed the worst results. The primary difference between the initial test and subsequent iterations was the variation in individual speed. In Brave, the single-line notation of the CSS variable showed a significant acceleration, from 6.73% to 4.65%. Similarly, in anime.js, the single-line notation showed a significant improvement, ranging from 6.05% to 12.36%. The above graphs show that with 75 objects or less, the anime.js animation library has very similar results to the two-line CSS variable notation, as it was in the initial tests.

5.3. Testing on the browser Vivaldi (engine Goana, fork Webkit)

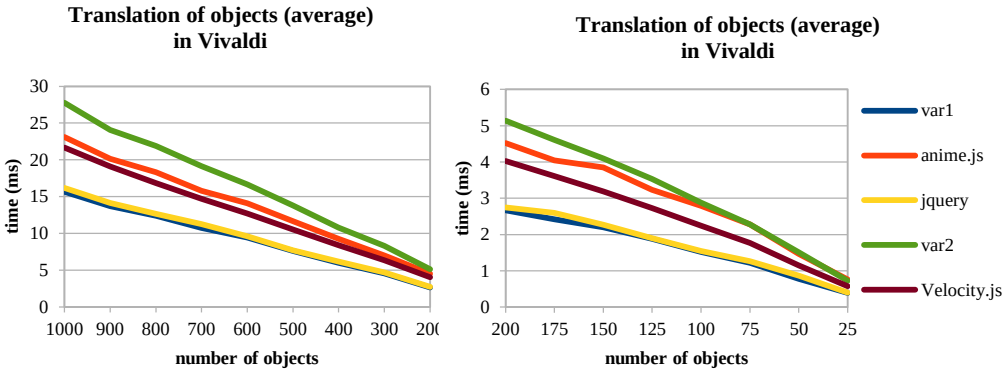

Figure 8. Graphical representation of the transformation in the browser Vivaldi

Table III. Speed measurement results (in ms) as a function of the number of rendered objects in the browser Vivaldi

Vivaldi	1000	900	800	700	600	500	400	300	200	175	150	125	100	75	50	25
var1	15.67	13.71	12.4	10.73	9.45	7.61	6.01	4.57	2.67	2.42	2.2	1.88	1.52	1.22	0.78	0.39
anime.js	23.09	20.15	18.32	15.81	14.08	11.68	9.26	7.02	4.52	4.04	3.85	3.23	2.79	2.28	1.47	0.76
jquery	16.19	14.13	12.67	11.28	9.62	7.71	6.17	4.7	2.75	2.6	2.27	1.9	1.55	1.26	0.87	0.4
var2	27.76	24.06	21.87	19.13	16.67	13.79	10.78	8.29	5.14	4.61	4.1	3.53	2.88	2.28	1.51	0.73
Velocity.js	21.68	19.09	16.85	14.71	12.69	10.54	8.38	6.34	4.03	3.61	3.19	2.73	2.24	1.77	1.15	0.58

The third browser tested was Vivaldi (Figure 8), which uses the Goana engine (a fork of Webkit). While Vivaldi was slower than Brave (Table III), it was faster than Chrome. As shown in the graphs above, both the anime.js and Velocity.js animation libraries show an almost constant discrepancy at high object counts, similar to that observed in Chrome. However, further investigation and data collection is required to confirm these findings. However, a more pronounced discrepancy was observed at lower object counts, where the solution using the anime.js animation library showed a noticeable delay starting at 100 objects. The graph in Figure 8 shows that the use of a single-line dynamic CSS variable notation is slightly better than the native jQuery function.

5.4. Testing on the browser Firefox (engine Gecko)

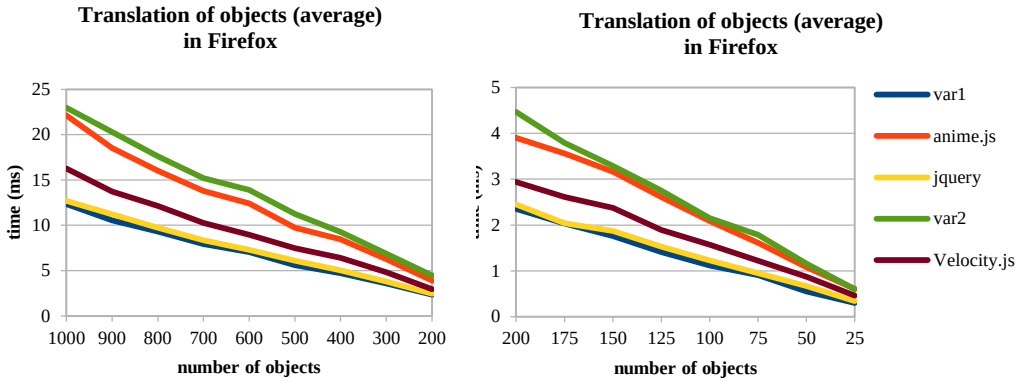

Figure 9. Graphical representation of the transformation in the browser Firefox

Table IV. Speed measurement results (in ms) as a function of the number of rendered objects in the browser Firefox

Firefox	1000	900	800	700	600	500	400	300	200	175	150	125	100	75	50	25
var1	12.35	10.55	9.3	7.93	7.06	5.61	4.72	3.58	2.35	2.03	1.76	1.41	1.12	0.91	0.55	0.3
anime.js	22.12	18.53	16.03	13.79	12.4	9.72	8.45	6.29	3.9	3.56	3.16	2.61	2.09	1.61	1.08	0.61
jquery	12.69	11.2	9.74	8.36	7.3	6.1	5.05	3.82	2.45	2.04	1.87	1.53	1.23	0.95	0.67	0.34
var2	22.99	20.3	17.62	15.21	13.91	11.24	9.26	6.88	4.47	3.79	3.29	2.75	2.15	1.79	1.16	0.6
Velocity.js	16.27	13.72	12.14	10.25	8.95	7.47	6.42	4.81	2.94	2.61	2.37	1.89	1.57	1.22	0.87	0.46

The final browser tested was Firefox (Figure 9), which uses the Gecko engine. In the experiment carried out (Table IV), this browser showed optimal performance for 2D animations for single-line dynamic variable CSS notation when rendering 1,000 objects, with results ranging from 19.02% (Vivaldi) to 24.47% (Chrome), and when rendering 25 objects, with results ranging from 23.08% (Vivaldi) to 30.23% (Chrome). The results of the tests show a remarkable discrepancy in the use of the libraries tested. The use of the Velocity.js animation library showed similar performance to single-line CSS dynamic variable notation and native jQuery functions. However, the anime.js animation library showed a marked decline from the beginning of the test phase, with results comparable to two-line CSS dynamic

variable notation. The discrepancy between the performance of the anime.js animation library and the other solutions tested became more pronounced at lower object counts. In fact, the anime.js solution was ineffective for simple animations when only 150 objects were used.

In conclusion, the single-line dynamic variable CSS notation was shown to be a more robust solution in terms of rendering time and object transformation compared to JavaScript libraries such as anime.js or Velocity.js. The single-line CSS notation renders objects directly in the browser thread, reducing the time required for additional JavaScript function calls and minimizing the impact of main thread blocking. This solution produces smoother animations with a higher frame rate per second, which is considered paramount for mobile devices operating in constrained computing environments. As a result, code optimization is solved at the outset, eliminating the need for further code tuning, especially for simple 2D animations. It is also worth noting that most current web browsers offer native support for hardware acceleration, particularly in the context of graphics acceleration. This support contributes to the efficiency of the code animation mechanism. In particular, the results of similar measurements by Okamoto (2024) and Loknar (2023) showed that animations using single-line CSS variable notation consumed on average 30% less CPU time than anime.js and Velocity.js.

The results of the experiment conducted show that the method of rendering objects and their subsequent transformation using single-line CSS variable notation showed superior performance, with a range of 19.02% (Vivaldi) to 24.47% (Chrome), compared to the alternative method. The transformation in each of the web browsers tested, while rendering 25 objects, achieved a result ranging from 23.08% (Vivaldi) to 30.23% (Chrome), with superior performance compared to the alternative method of creating 2D animations. The single-line CSS variable notation represents a significant departure from traditional approaches to dynamic animation through declarative programming. This approach offers the advantage of predictable application behavior, which facilitates subsequent code testing and debugging. The advantage of using CSS animations is that they maintain the accessibility of web pages even when JavaScript animation libraries fail to load. Compared to JavaScript

libraries, which require more CPU power and can cause rendering delays, rendering and transforming objects in CSS allows browsers to process animations more quickly at the rendering stage, resulting in a more seamless user experience. The single-line CSS variable notation contributes to an agile approach to animation development by facilitating iterations and optimizing visual effects, which is consistent with the high level of interactivity and appeal of the application.

The results of this study can be effectively used in a variety of contexts, particularly in areas where data visualization is a necessity, including educational applications and interactive narratives. The ability to effectively present data through animation is a key factor in ensuring optimal user experience with a given web application. Educational applications can include interactive components for adaptive assessments, interactive exercises or simulations that respond to user actions, thereby facilitating an effective and immersive learning experience. In addition, web technologies can facilitate the incorporation of multimedia content, including videos, animations and 3D models, into the learning process, thereby helping to visualize complex concepts or procedures. The creation of interactive narratives in digital games using immersive techniques, where the user is not just a passive observer but an active participant in shaping the narrative. The use of diverse techniques for creating animation, interactivity and dynamic content generation allows the development of a distinctive experience that allows the user to make dynamic decisions about the progression of the narrative, thereby increasing engagement and the overall user experience of the story.

Unlike freely available JavaScript libraries such as Three.js, Lottie was designed primarily for 3D animation, while anime.js and Velocity.js were designed for 2D animation. Our solution, based on dynamic CSS variables, has a much lower computational complexity and a simple implementation for easy transformations. Conversely, the GreenSock Animation Platform (GSAP) library is a versatile and flexible tool for creating complex animations and interactions (Musich, 2022). It provides complex functionality and efficiency in managing multiple animations and timing synchronization, but often requires additional resources and can add complexity when animation requirements are relatively simple. In contrast, dynamic

CSS variables allow users to achieve visually appealing effects and animations without the need to write extensive JavaScript code. Transformations can be implemented directly in CSS without additional knowledge of the CSS library, and variable values can be flexibly changed based on user interaction, resulting in smooth and fluid animations. This allows the CPU and GPU to optimize rendering, resulting in better overall performance, especially on mobile devices or slower computers where optimization is critical. As mentioned above, the advantage of dynamic CSS variables is their clarity and simplicity, which developers can benefit from. A basic understanding of CSS concepts is sufficient to significantly reduce the learning curve compared to the complexity of libraries such as GSAP. This allows developers to implement dynamic solutions and integrate visual effects more efficiently without having to learn complex programming paradigms. The advent of dynamic CSS variables represents a significant technological advance in the field of web animation optimization, offering distinct advantages in the context of standardized visual transformation. Their implementation reduces computational complexity while providing an efficient tool for implementing simpler animation strategies. Although specialized animation libraries such as GSAP retain a comparative advantage for more complex scenarios, dynamic CSS variables represent a progressive alternative that optimally integrates the principles of computational efficiency, implementation simplicity, and high-performance design in a modern web environment.

The experimental results, obtained in different browsers using different web engines, yielded unexpected results in the context of the single-line dynamic variable CSS notation. Based on systematic empirical observations and extensive surveys of web application developers, Cascading Style Sheets (CSS) emerges as the dominant and most computationally efficient technological paradigm for rendering two-dimensional animated sequences within web page environments. It is somewhat paradoxical that numerous studies have focused on user experience (UX) and user interface (UI), but have neglected to assess the speed at which a CSS variable value is communicated to a class. The measurement results showed a notable discrepancy between the single-line and two-line approaches to CSS variable writing (value

passing and class setting), with the former showing a significantly faster processing time. This example illustrates the benefits of a single DOM object approach. As evidenced by the literature and research studies conducted by Loknar (2023), Park (2016), Madsen (2013), Yeo (2019), and Obbink (2018), the optimal order is as follows: In terms of performance, CSS is the clear winner, with Velocity.js and anime.js providing similar results. jQuery, however, is not as efficient. As mentioned above, simply invoking the CSS class represents a static solution. In contrast, the results of the experiment represent a dynamic solution in the form of a CSS variable, where the class and inline style of the variable are assigned to the object with a value. A notable difference in the methodology used for a particular DOM element is the discrepancy between the number of entries in the jQuery and JavaScript environments. Based on the measurement results above, a revised sequence for addressing dynamic 2D animation can be proposed as follows: single-line CSS variable notation > jQuery > Velocity.js > Anime.js > two-line CSS variable notation. Rendering objects and transforming them using CSS notation is an effective way to develop mobile browser-optimized applications. This approach is characterized by simplicity and power.

This technology allows developers to maintain clean and valid code, reducing the need for maintenance or further modification. In the context of developing applications for mobile devices, where every millisecond is critical (Li, 2020; Su, 2021), the single-line CSS variable notation represents a significant optimization of the code used to display animations in real time. The minimal access to the DOM object not only reduces time costs, but also streamlines the animation management process. Mobile devices often operate with limited computing and graphics capabilities and limited access to device memory. As a result, single-line variable notations in CSS provide an efficient rendering style for a web browser. This allows dynamically processed animations to be rendered more smoothly, reducing the load on the device itself. Given current trends in web design, where mobile devices are the dominant platform, single-line CSS notations are proving to be a flexible and powerful solution.

6. LIMITATIONS AND FUTURE RESEARCH

Despite their widespread use, conventional technologies such as HTML DOM and Canvas are susceptible to performance limitations when faced with more sophisticated animation. This observation is particularly relevant for developers seeking to improve the efficiency of games and interactive educational applications.
As mentioned above, the average refresh time depends on several variables. JavaScript animations often face performance constraints, especially when managing large numbers of objects. Their performance can vary considerably between different web browsers. Comparable research by Macedo (2021), Mendki (2020), Yan (2021) using WebAssembly shows that JavaScript performance degrades as the number of objects increases, resulting in a reduction in animation rendering time. The time series of HTML DOM and Canvas implementations show a consistent increase in time as the number of objects increases, suggesting that JavaScript faces challenges in maintaining efficiency under heavier loads (Goukouni, 2022). The results suggest that for simple animations (up to 200 objects) with simple transformations, this may not be a significant issue. However, for collision detection and management, JavaScript implementation can be a critical factor in reducing web browser performance (Pires, 2023; Dubey, 2024). Function invocation via WebAssembly from JavaScript environments has been observed to have potentially inefficient energy characteristics, primarily due to repeated context switching between heterogeneous computing environments. This is a significant limiting factor for event-driven application architectures (Yan 2021). WebAssembly is emerging as a transformative binary instruction format with significant potential to improve web application performance and computational efficiency; however, further interdisciplinary research is imperative to comprehensively evaluate its long-term architectural implications, cross-platform compatibility, and potential limitations in different computing ecosystems. As a result, its current implementation may result in suboptimal CPU utilization due to its interaction with JavaScript. Therefore, the selection of an appropriate implementation method is paramount to the efficient management of web animations.

The test itself and the results obtained are limited to the web browsers and versions used in the experiment. Each browser has its own characteristics, certain parameters, including the ability to use RAM, processor clock and graphics card when rendering and transforming objects to create animations. The performance of individual hardware components, even specific animations, can affect the test itself. These factors limit the ability to extend the results to other platforms and configurations. It is therefore necessary to extend the scope of the tests to other platforms and browsers in order to make the results universal and to ensure the accuracy and validity of the results.

In recent years, web animation has played an important role in enhancing the interactivity of web applications and digital products. The most commonly used techniques include tweening (interpolation between animation frames), CSS animation, SVG animation, and JavaScript-driven animation. Each of these methods has specific performance characteristics that can affect the smoothness and timing of animations on different devices and under different network conditions. Overall, these methods can be divided into two groups. Tweening and SVG animation can be classified as static animation, and CSS and JavaScript animation can be classified as dynamic animation. For example, CSS animations are often perceived as less resource-intensive and more efficient, while JavaScript-driven animations offer more flexibility but can degrade performance on slower connections.

Further research could analyze how different animation techniques affect perceptions of fluidity, focusing on metrics such as frames per second (FPS) and latency between user interaction and animation response. In addition to technical aspects such as optimizing animations for different devices and platforms, subjective factors such as users' emotional reactions to smooth or jerky animations should also be considered. Animations with low FPS or delays can have a negative impact on the user experience, which can lead to a deterioration of the overall app experience.
Implementing adaptive solutions, such as dynamically adjusting animation quality based on bandwidth conditions, can significantly mitigate the negative effects of poor connectivity on the user experience. This can be achieved by reducing the resolution or frame rate of animations.

This research can significantly contribute to a deeper understanding of the relationship between animation techniques, system performance and subjective user perception, particularly in the context of evolving Internet connectivity conditions. Optimizing these elements will facilitate the development of more adaptive and flexible applications that provide a consistently high quality user experience, regardless of the technical limitations of the user. The results of this research could have wide-ranging applications, not only in web development, but also in areas such as data visualization, educational applications, game design and other industries where ensuring positive user interaction with technology is critical.

When researching and testing animation and user experience (UX), it is important to consider not only the technologies and devices available today, but also the historical circumstances that may affect the way the user experience works. Factors such as slow internet connections and older browser versions should be considered in future testing. These issues can have a significant impact on animation performance, interaction smoothness and user satisfaction. It is therefore crucial that applications and websites are adapted to these conditions.

7. CONCLUSION

The experimental results show that rendering objects and transforming them using CSS is the optimal solution for implementing simple animations, especially in the context of Web 3.0 technologies. The special features of this technology are the fast response time and the adaptability of the design, which, together with the interactivity, improves the user experience of the resulting web application. The fast response time is mainly due to the optimization of the reduction of the processor, RAM and GPU load with regard to the responsive design of web animations, which is of key importance to developers. The fast rendering of objects and the subsequent visual effects that can be achieved using CSS help to improve the user experience and, as a result, maintain the competitiveness of the web applications being created. Therefore, the rendering and transformation of objects using CSS is crucial and must be considered, especially in situations where the conservation of hardware resources and the actual load on the computing system are important.

The results show that dynamic CSS variable resolution with single-line notation is the most effective technique, outperforming the other methods tested. This highlights the superiority of native browser technologies for simple 2D animation, particularly object translation. In addition, the performance of jQuery was superior to that of the JS animation libraries, including anime.js and Velocity.js. This suggests that the proposed library remains highly efficient and well optimized for simple 2D animation types. It is noteworthy that the dynamic solution using a CSS variable with two-line notation in jQuery or JavaScript (passing value + setting class) showed inferior performance compared to the Velocity.js animation library. This result highlights the benefits of using the jQuery library in single-line notation for DOM object access, as opposed to relying solely on vanilla JS notation. This is because the two-line notation for dynamic CSS variable resolution requires additional processing of the DOM object by the web browser. The anime.js animation library showed the worst performance in the implemented experiment, due to suboptimal hardware acceleration-based optimization compared to native CSS 2D animations. Conversely, the advantages of the JS animation libraries (anime.js and Velocity.js) were evident in their ease of use as well as the complexity of their functions, which went beyond simple transformation solving.

ACKNOWLEDGEMENTS

This research was supported by the grant KEGA 020UCM-4/2022 (Adaptive platform for the development of statistical literacy).

REFERENCES

Ahmad, N. J., Yakob, N., Bunyamin, M. A. H., Winarno, N., & Akmal, W. H. (2021). The effect of interactive computer animation and simulation on students' achievement and motivation in learning electrochemistry. *Jurnal Pendidikan IPA Indonesia*, *10*(3), 311-324.

Alsalhi, N. R., Alqawasmi, A., Abdelkader, A. F., Alqatawneh, S., & Salem, O. (2024). The Effect of Using PhET Interactive Simulations on Academic Achievement of Physics Students in Higher Education Institutions. *Educational Sciences: Theory & Practice*, *24*(1), 65-75.

Arslan, K., & Karakuş, N. (2024). Environmental Teaching Supported by Web 2.0-Based Digital Games for a Sustainable Life. *Sustainability*, *16*(22), 9691.

Beňo, M., & Ölvecký, M. (2019). The impact of virtual learning tools on the student's remembering knowledge. In 2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA) (pp. 64-68). IEEE. https://doi.org/10.1109/ICETA48886.2019.9040120

De Macedo, J., Abreu, R., Pereira, R., & Saraiva, J. (2021, November). On the runtime and energy performance of webassembly: Is webassembly superior to javascript yet?. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)* (pp. 255-262). IEEE.

Dubey, A., Chaturkar, K., Pawar, R., Sarode, A., & Shirbhate, D. D. (2024). A Comprehensive Review of JavaScript Frameworks. *Grenze International Journal of Engineering & Technology (GIJET)*, *10*.

Garaizar, P., & Reips, U. D. (2019). Best practices: Two Web-browser-based methods for stimulus presentation in behavioral experiments with high-resolution timing requirements. *Behavior Research Methods*, *51*, 1441-1453.

Goh, H. A., Ho, C. K., & Abas, F. S. (2023). Front-end deep learning web apps development and deployment: a review. *Applied Intelligence*, *53*(12), 15923-15945.

Gotskaya, I. (2019). Approaches towards the Comparison and Utilization of JavaScript Animation Libraries.

Goukouni, B. Y., Aamir, M., Ali, W., Dayo, Z. A., Abro, W. A., Ishfaq, M., & Yurong, G. (2022). Methods Tested to Optimize the Performance of WEBGL Applications. In *Sensing Technology: Proceedings of ICST 2022* (pp. 339-354). Cham: Springer International Publishing.

Hanif, M. (2020). The Development and Effectiveness of Motion Graphic Animation Videos to Improve Primary School Students' Sciences Learning Outcomes. *International Journal of Instruction*, *13*(3), 247-266.

Hosťovecký, M., Korečko, Š., & Sobota, B. (2024) Petri nets for Adaptive learning scenarios in Serious games. Journal of Applied Mathematics, Statistics and Informatics, 20(1), 67-84.

Hu, B. (2021, April). Animation Based Narrative Strategy and Shaping of Image Animation-based Narrative Strategy and Image Shaping in the Information Age. In Journal of Physics: Conference Series (Vol. 1852, No. 4, p. 042016). IOP Publishing.

Irina, Iegorova., Maryna, Komina. (2020). Development of methods for effective application of animation in the WEB. 60-64. doi: 10.20998/2413-4295.2020.04.09

Chen, D. (2020, July). An analysis of mainstream animation education in Japan and the United States using computer 3D technology. In Journal of Physics: Conference Series (Vol. 1578, No. 1, p. 012026). IOP Publishing.

Jurinová, J. (2023). Improving the quality of education in the development of algorithmic and critical thinking of students of "Applied Informatics" – case study. International Journal of Engineering Pedagogy (iJEP), 13(7), 79–95. https://doi.org/10.3991/ijep.v13i7.37749

Li, L., Qiao, X., Lu, Q., Ren, P., & Lin, R. (2020). Rendering optimization for mobile web 3d based on animation data separation and on-demand loading. *IEEE Access*, *8*, 88474-88486.

Madsen, M., Livshits, B., & Fanning, M. (2013). Practical static analysis of JavaScript applications in the presence of frameworks and libraries. ESEC/FSE 2013.

Mendki, P. (2020, October). Evaluating webassembly enabled serverless approach for edge computing. In *2020 IEEE Cloud Summit* (pp. 161-166). IEEE.

Mishra, D. P., Rout, K. K., & Salkuti, S. R. (2021). Modern tools and current trends in web-development. *Indonesian Journal of Electrical Engineering and Computer Science*, *24*(2), 978-985.

Mušič, J., & Tomc, H. G. (2022). A comparison of current solutions for creating web animations on Apple hardware. Journal of Print and Media Technology Research, 11(2), 129-140.

Obbink, N.G., Malavolta, I., Scoccia, G.L., & Lago, P. (2018). An extensible approach for taming the challenges of JavaScript dead code elimination. 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 291-401.

Okamoto, S., Kohana, M., & Kamada, M. (2024, September). An Evaluation of Implementation Options for Web-Based Interactive Animations. In *International Conference on Network-Based Information Systems* (pp. 120-128). Cham: Springer Nature Switzerland.

Park, J., Kim, H., & Moon, I. (2018). The JavaScript and Web Assembly Function Analysis to Improve Performance of Web Application. International Journal of Advanced Science and Technology.

Park, H., Cha, M., & Moon, S. (2016). Concurrent JavaScript Parsing for Faster Loading of Web Apps. ACM Transactions on Architecture and Code Optimization (TACO), 13, 1 - 24.

Pires, J. G. (2023). Machine learning in medicine using JavaScript: building web apps using TensorFlow. js for interpreting biomedical datasets. *medRxiv*, 2023-06.

Semerádová, T., & Weinlich, P. (2020). *Website quality and shopping behavior: Quantitative and qualitative evidence*. Springer. https://doi.org/10.1007/978-3-030-44440-2

Sharples, Mike. (2019). Learning from animations : Watch and interact with short animations. 114-117. doi: 10.4324/9780429485534-19

Shekhar, Rathor., Mingyu, Zhang., Taehoon, Im. (2023). Web 3.0 and Sustainability Challenges and Research Opportunities. Sustainability, doi: 10.3390/su152015126

Stanić Loknar, N., Koren Ivančević, T., Bekavac, M. & Ivančić Valenko, S. (2023). Technologies for Web Animations and their Positive and Negative Sides Regarding Web Page Metrics. *Tehnički glasnik, 17* (4), 566-571. https://doi.org/10.31803/tg-20230621113640

Steyer, R. (2022). Transitions and Animations: Moving Things. In *Building web applications with Vue. js: MVVM patterns for conventional and single-page websites* (pp. 193-201). Wiesbaden: Springer Fachmedien Wiesbaden.

Su, Y., Chen, G., Li, M., Shi, T., & Fang, D. (2021). Design and implementation of web multimedia teaching evaluation system based on artificial intelligence and jQuery. *Mobile Information Systems*, *2021*(1), 7318891.

Valentová, M., & Brečka, P. (2023). Assessment of Digital Games in Technology Education. Int. J. Eng. Pedagog., 13, 36-63.

Yan, Y., Tu, T., Zhao, L., Zhou, Y., & Wang, W. (2021, November). Understanding the performance of webassembly applications. In *Proceedings of the 21st ACM Internet Measurement Conference* (pp. 533-549).

Yeo, J.H., Oh, J., & Moon, S. (2019). Accelerating web application loading with snapshot of event and DOM handling. Proceedings of the 28th International Conference on Compiler Construction.

Ying, Z. (2021, April). Interactive film and television animation special effects production techniques in visual design. In Journal of Physics: Conference Series (Vol. 1881, No. 2, p. 022020). IOP Publishing.

Yu, G., Liu, C., Fang, T., Jia, J., Lin, E., He, Y., ... & Huang, Q. (2023). A survey of real-time rendering on Web3D application. *Virtual Reality & Intelligent Hardware*, *5*(5), 379-394.

Zhang, Y., Li, X., Yang, Z., Hu, S., Fu, X., & Shen, W. (2023). Recognition and statistical method of cows rumination and eating behaviors based on Tensorflow. js. *Information Processing in Agriculture*.

Miroslav Beňo
Institute of Computer Technologies and Informatics,
Faculty of Natural Sciences,
University of SS. Cyril and Methodius, 917 01 Trnava, Slovak Republic
Email: miroslav.beno@ucm.sk

Miroslav Ölvecký
Institute of Computer Technologies and Informatics,
Faculty of Natural Sciences,
University of SS. Cyril and Methodius, 917 01 Trnava, Slovak Republic
Email: miroslav.olvecky@ucm.sk