Demystifying Web-based Mobile Extended Reality Accelerated by WebAssembly

Kaiyan Liu*
George Mason University
kliu23@gmu.edu

Nan Wu*
George Mason University
nwu5@gmu.edu

Bo Han
George Mason University
bohan@gmu.edu

ABSTRACT

By combining various emerging technologies, mobile extended reality (XR) blends the real world with virtual content to create a spectrum of immersive experiences. Although Web-based XR can offer attractive features such as better accessibility, cross-platform compatibility, and instant updates, its performance may not be on par with its standalone counterpart. As a low-level bytecode, WebAssembly has the potential to drastically accelerate Web-based XR by enabling near-native execution speed. However, little has been known about how well Web-based XR performs with WebAssembly acceleration. To bridge this crucial gap, we conduct a first-of-its-kind systematic and empirical study to analyze the performance of Web-based XR expedited by WebAssembly on four diverse platforms with five different browsers. Our measurement results reveal that although WebAssemlby can accelerate different XR tasks in various contexts, there remains a substantial performance disparity between Web-based and standalone XR. We hope our findings can foster the realization of an immersive Web that is accessible to a wider audience with various emerging technologies.

CCS CONCEPTS

• **Networks** → **Network measurement**; • **Computing methodologies** → **Virtual reality**; **Mixed / augmented reality**.

KEYWORDS

Web-based XR, WebAssembly, Mobile Performance

ACM Reference Format:
Kaiyan Liu*, Nan Wu*, and Bo Han. 2023. Demystifying Web-based Mobile Extended Reality Accelerated by WebAssembly . In *Proceedings of the 2023 ACM Internet Measurement Conference (IMC '23), October 24–26, 2023, Montreal, QC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3618257.3624833

1 INTRODUCTION

As an umbrella term, extended reality (XR) encompasses various immersive technologies, including augmented reality (AR), virtual reality (VR), and mixed reality (MR) [63]. It enables developers to create cross-platform XR experiences on a wide range of devices, such as headsets, smartphones, tablets, and personal computers (PCs). As of May 2023, >70% of installed browsers support the WebXR device API [24]. While modern Web browsers have become more powerful, the performance of Web-based applications may still not be comparable to that of their native, standalone counterparts [26, 55]. Thus, ensuring smooth, immersive XR experiences on diverse devices remains a challenge.

WebAssembly (abbreviated as Wasm), a portable, low-level bytecode, has recently been proposed to improve the performance of Web applications [36]. Wasm serves as a compilation target for high-level languages such as C, C++, and Rust, and executes efficiently with near-native performance on Web browsers. In contrast, JavaScript (abbreviated as JS), a scripting language, has been the de facto language of Web development for decades [69]. Nevertheless, they can work together to enhance Web applications, for example, by making JS handle user interactions and high-level logic and Wasm take charge of computation-intensive and performance-critical tasks. As of May 2023, Wasm is supported by an impressive 96.3% of installed browsers [23].

Although there have been recent efforts to understand the performance of Wasm applications [43, 69] and Web-based XR [26] independently, little has been known about how Web-based XR can benefit from Wasm to create a more immersive Web. For example, under certain circumstances, the performance of Wasm could be even worse than that of JS [69]. To fill this critical gap, in this paper, we conduct, to the best of our knowledge, the first systematic study to dissect the performance of Web-based XR applications that are accelerated by Wasm, by exploring various factors and comparing it with those developed solely with JS.

We conduct extensive measurement studies on four diverse platforms, Microsoft HoloLens 2 MR headset [12], Oculus Quest 2 VR headset [14], Samsung Galaxy S21 and S22+ smartphones, and Apple iPhone 14 Pro smartphone, with five browsers, Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, and the Oculus browser. We experiment with three primary XR-related tasks that can benefit from Wasm, 3D content loading and rendering [26], object tracking [73], and image super-resolution [47].

We summarize the key findings of our study as follows.

(1) The performance improvement brought about by Wasm for the above three web-based and XR-related tasks varies, ranging from approximately 10% to as high as 84%. This variation is influenced by factors such as the specific XR tasks, devices, and web browsers being used.

(2) Despite the notable acceleration facilitated by Wasm on XR headsets compared to smartphones, the performance of Web-based XR on headsets still falls significantly behind that of their smartphone counterparts.

(3) Even accelerated by Wasm, the processing time of XR tasks executed on browsers could still be much longer than that of standalone XR applications, leading to potential opportunities for further enhancement.

*These authors contributed equally to this work.

IMC '23, October 24–26, 2023, Montreal, QC, Canada

ACM ISBN 979-8-4007-0382-9/23/10.
https://doi.org/10.1145/3618257.3624833

Our findings contribute to a more comprehensive understanding of the influencing factors of the performance of Web-based XR accelerated by Wasm. We hope that our study can help the developers of Wasm and Web-based XR applications jointly explore avenues to further improve the immersive experiences on the Web.

2 BACKGROUND

Mobile XR refers to XR experiences on mobile devices including smartphones, tablets, and headsets such as Microsoft HoloLens 2 [12], Magic Leap 2 [21], and Oculus Quest 2 [14]. It benefits from the computing resources, various sensors, and advanced display capabilities of these devices [41] to generate interactive and immersive virtual environments (VR), overlay informative digital content onto real-world objects (AR), or blend the real environment and virtual content for users to interact with both (MR [61]). Mobile XR has practical uses in various domains, including education, training, healthcare, gaming, and entertainment [66].

Web-based XR refers to the integration of XR into Web applications, without requiring dedicated, standalone XR applications or specialized hardware. To achieve this goal, WebXR [15, 27] establishes a platform-independent bridge to the immersive experiences created in a Web browser by AR, VR, and MR applications [26, 54, 58]. The primary appeal of WebXR lies in its portability and accessibility, simplifying cross-platform development and user interaction with XR content. The general frameworks for Web-based XR, such as A-Frame [8], three.js [4], AR.js [10], and Babylon.js [6], provide libraries for streamlining the creation of 3D scenes, managing user interactions, and handling animations. Hence, Web-based XR not only eases access for users who can directly enjoy immersive experiences without installing specific applications but also reduces developers' workload with unified frameworks for different devices and platforms [58].

WebAssembly is a low-level bytecode and serves as a portable target for the compilation of high-level languages such as C, C++, and Rust [36, 69]. The principal advantages of Wasm include its ability to compile diverse source codes into a browser-executable format, the capacity to work in parallel with JS for handling computation-intensive tasks, as well as notably elevated execution time and small size of compiled code. These benefits are primarily due to its compact binary format, efficient instruction parsing, and ahead-of-time (AOT) compilation [45]. Thus, Wasm programs can be loaded and decoded more swiftly than JS ones [69], enhancing the efficiency and performance of Web applications.

Challenges. The key research challenges of Web-based mobile XR include (1) high framerate requirements for XR applications to prevent user dizziness, constrained by the limited processing power and memory of Web browsers [45], (2) high computational demand for rendering photorealistic graphical content [26, 67], (3) accurate tracking and localization to guarantee a satisfactory user experience [73], and (4) the cross-platform compatibility issue with most Web browsers, notably on mobile devices [71]. In response to these challenges, several Web-based XR engines have incorporated WebAssembly [36] for performance optimization, such as Magnum [3] and Wonderland Engine [16].

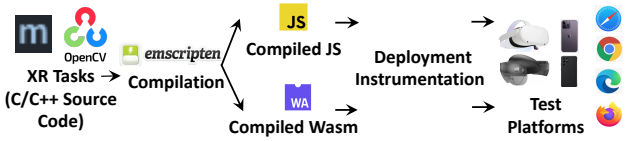

Figure 1: Overview of our measurement workflow.

However, the integration of Wasm introduces its own set of challenges, including (1) interoperability difficulties with JS, particularly regarding the handling of complex data types and the integration of Wasm modules into JS toolchains [37], (2) instability in runtime performance, influenced by various factors such as the runtime environment and the compilers [69], and (3) a higher memory consumption than JS, attributed to the linear memory model of Wasm that does not automatically reclaim unused memory [36, 69]. To address these issues, several compilers including Emscripten [2], Binaryen [9], and Cheerp [7], and frameworks such as Blazor [13] have been released. In this study, we evaluate the performance of Web-based XR enhanced by Wasm.

3 METHODOLOGY

In this section, we present our measurement methodology, with the workflow shown in Figure 1.

Hardware and Software. We employ diverse devices, including Samsung Galaxy S21/S22+ (Android 13), iPhone 14 Pro (iOS 16.0.3), Microsoft HoloLens 2 MR headset [12] (Windows Holographic OS 22621.1113), and Oculus Quest 2 VR headset [14] (OS SQ3A.220605.009.A1 based on Android) for our study. We experiment with different browsers on these devices. On the Samsung Galaxy S21, we experiment with Chrome (113.0.5672.132), Edge (113.0.1774.38), and Firefox (115.0a1) browsers. There is no official Safari for Android devices [17]. Since iOS devices lack support for Web-based XR in general [20], our exclusive focus centers on Safari (16) to highlight its XR capabilities. For HoloLens 2, the officially supported browser is Edge (113.0.1774.50). Similarly, Quest 2 supports mainly the Oculus browser (27.1.0.11.62.475067835).

XR Components. Our experiments primarily focus on three XR-related tasks: loading and rendering 3D content [26], tracking objects [73], and upsampling images (*i.e.,* super-resolution) [47]. For content loading and rendering, we obtain two 3D meshes in glTF format from Sketchfab [5] to accommodate devices with varying computation resources, and they differ in data size and model complexity. The small one [22] has a size of 8.86 MB, with 35.7K vertices, 66.7K triangles, and 14 2K textures (2K resolution). The large one [18] is 30.7 MB in size and has 165.1K vertices, 275.1K triangles, and 17 textures (2K resolution). Devices with higher computing capacity such as the Galaxy S21 and Quest 2 employ the larger model, while those constrained by computational power such as Hololens 2, or with memory limitations such as the iPhone 14 Pro utilize the smaller one. The complexity of these models, particularly the large one, exceeds that of many models in large datasets such as Thingi10K [59] and ShapeNetCore [29]. While the majority of models in these datasets have under 100K vertices, our choices challenge the boundaries of what is achievable in the realm of 3D model handling and optimization.

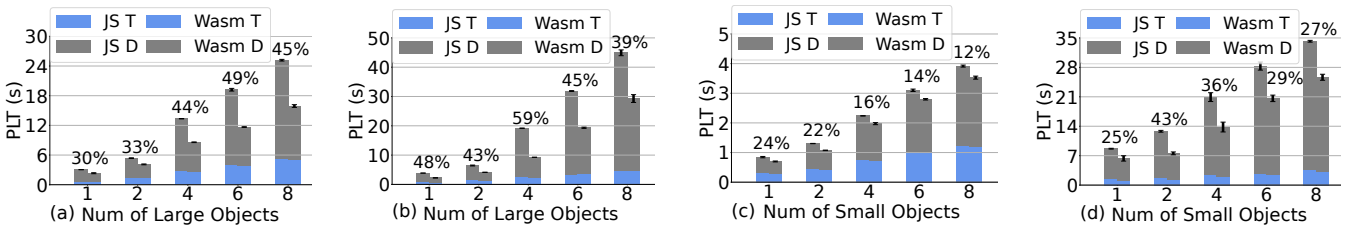

Figure 2: Comparison of page load time (PLT) for JavaScript (JS) and WebAssembly (Wasm): (a) Chrome on Galaxy S21 (Android), (b) Oculus browser on Quest 2, (c) Safari on iPhone 14 Pro (iOS), and (d) Edge on Hololens 2. The percentages are the improvements in content-decoding time. T: Transmission, and D: Decoding.

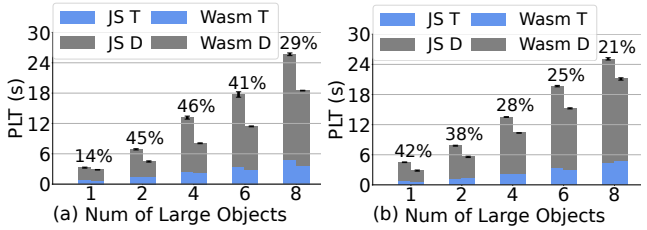

Figure 3: Comparison of page load time (PLT) for JavaScript (JS) and WebAssembly (Wasm) with different browsers on Samsung Galaxy S21: (a) Edge and (b) Firefox. The percentages are the improvements in content-decoding time. T: Transmission, and D: Decoding.

Object tracking identifies and tracks objects in the device camera by comparing the captured frames with reference images. We leverage the Lucas-Kanade tracking algorithm [52] which calculates the optical flow to estimate object motion across frames, and use 1024×768 reference images. Image super-resolution enhances the resolution of camera-captured frames, and we increase their resolution by 16×. We focus on interpolation-based super-resolution [70] that mainly utilizes CPUs for computation. For both tasks, we vary the resolution of captured frames to evaluate the processing time.

XR Frameworks. We choose the following XR-related framework/library for our study. To facilitate the loading and rendering of 3D content, we select the Magnum Engine [3], an open-source, cross-platform framework for tasks such as rendering and input handling in interactive XR applications. It offers an abstraction layer over multiple graphics APIs such as WebGL, making it compatible with platforms including Windows, macOS/iOS, Linux, and Android, as well as various Web browsers. Note that the Wonderland Engine [16] is another framework primarily designed for developing Web-based graphics applications, which enables seamless integration of Wasm. However, it is difficult to disable Wasm in Wonderland since it is an integral part of the framework, leaving us unable to compare the performance between JS and Wasm. Thus, we focus on Magnum in this paper. For object tracking and image super-resolution, we utilize corresponding functions in OpenCV [11], a widely used open-source computer vision library.

To measure the acceleration capabilities of Wasm, we first cross-compile Magnum and OpenCV using Emscripten [2], generating different JS versions with and without Wasm acceleration. Subsequently, we develop Web applications using these versions and undertake a comparative analysis of their performance.

Performance Metrics and Analysis Tools. To comprehensively understand the impact of Wasm on Web-based XR for loading and rendering 3D content, we analyze various metrics, including page load time (PLT) [65] without caching, memory usage, CPU and GPU utilization, and framerate, Furthermore, we examine the processing time for object tracking and image super-resolution with and without Wasm. To gather the necessary data for analysis, we utilize developer tools for Web browsers [1], which provide detailed information on CPU/GPU usage, duration of each stage in the page load process, and execution time for individual tasks.

4 MEASUREMENT RESULTS

4.1 3D Content Loading and Rendering

For Web-based XR, the performance of content loading and rendering plays crucial roles, as it directly influences user experience. Hence, we conduct an analysis focusing on PLT, memory usage, as well as CPU and GPU utilization, by comparing the performance of JS and Wasm across a diverse range of devices and browsers. To facilitate this, we carry out our experiments using the Magnum Engine [3], with and without Wasm.

Page Load Time refers to the duration from when the user launches the Web-based XR application until it is fully loaded (*i.e.,* the first frame is rendered). It starts from the moment a user initiates a request, such as clicking a link, and ends when all page resources, including images, cascading style sheets (CSS), and either JS or Wasm module, are fully loaded and executed, and the first frame is rendered to the user. This process comprises three significant stages: *network transmission*, during which the application fetches and loads essential assets from the server; *content decoding*, a process in which 3D models are decoded and parsed into a format that the application can use; and the *first-frame rendering*, which marks the completion of the initial visual display of the content.

Figure 2 shows the comparison of PLT for JS and Wasm across the common browsers on four devices, Chrome on Galaxy S21, the Oculus browser on Quest 2, Safari on iPhone 14 Pro, and Edge on Hololens 2, with varying numbers of to-be-loaded objects to generate different workload. Among them, the former two use the large model, and the latter two use the small one due to the performance limitations of the devices[1]. Figure 3 depicts the comparison of PLT

[1]In general, iOS devices often have lower memory capacities compared to contemporaneous Android devices [19].

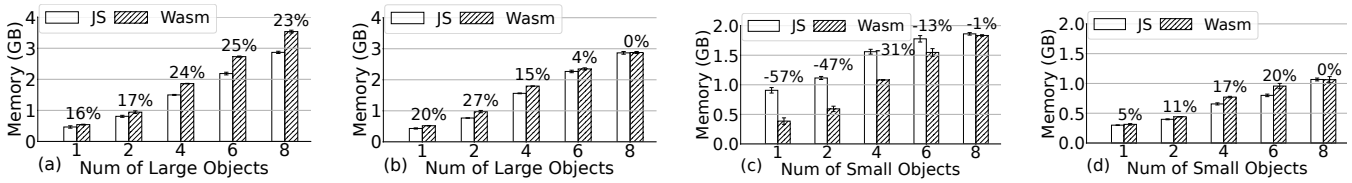

Figure 4: Comparison of memory usage for JavaScript (JS) and WebAssembly (Wasm): (a) Chrome on Galaxy S21 (Android), (b) Oculus browser on Quest 2, (c) Safari on iPhone 14 Pro (iOS), and (d) Edge on HoloLens 2.

for JS and Wasm with Edge and Firefox on Galaxy S21. We omit the frame rendering time from these figures for two reasons. First, it is short, consistently constituting less than 1% of the overall PLT. More importantly, rendering is mainly handled by GPU, which we have not yet explored to accelerate with Wasm. Based on the results revealed in these figures, we have the following findings:

(1) *Wasm can effectively accelerate content decoding, thereby shortening the decoding time within PLT on various devices.* Figure 2 demonstrates that Wasm can speed up content decoding by 17.6–46.8%, on average, compared to JS on different devices and various browsers. This is because Wasm programs, being delivered as compiled binaries, can be loaded and decoded more rapidly compared to JS, which requires parsing and compiling at runtime [36, 69]. More specifically, Wasm accelerates procedures such as shader compilation, mesh decoding, and matrix multiplications used in transformations, which require parsing and interpreting data. We also measure the framerate for rendering, which is close to 30 frames per second (FPS) for no more than four objects but decreases significantly in other scenarios. For instance, it drops to 20 FPS for Galaxy 21 and 7 FPS for Quest 2 when rendering eight objects.

As expected, Wasm has almost no impact on network transmission, since it is not computation-intensive. Note that the long multi-second PLT is caused by the large 3D models used to demonstrate the effectiveness of Wasm acceleration. In practice, it can be optimized to boost user experience by leveraging existing techniques such as (view-dependent) progressive mesh [39, 40]. Beyond measurements on a LAN, we examine the PLT on the Chrome browser with Galaxy S21 on a wide area network. Overall, the transmission time increases by 54.3-90.5% on average. However, there is no discernible alteration in decoding time, as the processes of transmission and decoding are independent of each other.

(2) *Wasm acceleration varies across different browsers and devices, presenting different levels of improvement.* Figure 2 reports that compared to JS, Wasm, on average, reduces the time for content decoding by 40.2% on Chrome using Galaxy S21, 46.8% on the Oculus browser using Quest 2, 17.6% on Safari using iPhone 14 Pro, and 32.0% on Edge using Hololens 2. Notably, the Oculus browser on Quest 2 provides the most significant Wasm acceleration. In general, the improvement for XR headsets is higher than that for smartphones. The reason is that headsets require a longer duration to decode 3D models of equivalent size compared to smartphones, leading to more room for Wasm to improve PLT.

Figure 2 shows that for the same 3D model sizes, Quest 2 exhibits a longer PLT with Wasm than Galaxy S21 (*e.g.,* 29.8s vs. 16.0s for eight large objects), while Hololens 2 takes longer time than iPhone 14 Pro (*e.g.,* 25.8s vs. 3.7s for eight small objects). This variation can be attributed to the fact that XR headsets, such as Quest 2 and Hololens 2, have constraints about power usage and thermal capacity [33]. To prevent overheating and ensure reliable operation, these devices may limit the performance of their CPU and GPU, which can subsequently impact decoding speed.

(3) *Different browsers on the same Android phone present varying degrees of support for Wasm.* By comparing Figures 3 and 2(a), we observe that, when evaluated on the Galaxy S21, Wasm reduces content decoding time, on average, by 40.2% on Chrome, 35.0% on Edge, and 30.8% on Firefox, compared to JS. In particular, Chrome not only has the best performance in PLT with Wasm among the three browsers (*e.g.,* 16.0s vs. 18.5s and 21.1s for eight objects) but also achieves the highest PLT reduction with Wasm (40.2% vs. 35.0% and 30.8%). Furthermore, we measure the impact of Wasm with the same version of the Chrome browser on both Galaxy S21 and S22+ with different CPUs, revealing that the acceleration ratios remain similar. These observations underscore the critical role of the browser's implementation and optimization of Wasm in influencing its performance.

Memory Usage. Figure 4 shows the comparison of memory usage for JS and Wasm across the common browsers on four devices. We have the following key findings from this figure:

(1) *Wasm typically consumes more memory compared to JS.* Figure 4 reports that compared to JS, Wasm, on average, increases memory usage by 21.0% on Chrome using Galaxy S21, 14.5% on the Oculus browser using Quest 2, and 16.5% on Edge using Hololens 2. To calculate the average ratio above, we disregard the one for decoding eight objects, because all browsers on various devices (except Galaxy S21) reach the limit of memory capacity in this case. The reason for the increase in memory usage is that JS employs garbage collection that dynamically oversees memory allocations and reclaims memory that is no longer utilized. In contrast, Wasm uses a linear memory model that does not automatically reclaim unused memory [36]. Consequently, while it proves advantageous for developers to employ Wasm for compute-intensive tasks, it is equally critical to enforce robust memory management strategies to prevent memory overflow across diverse devices.

(2) *Wasm consumes less memory than JS on the Safari browser on iOS devices.* Figure 4(c) illustrates that when using Safari on iPhone 14 Pro, Wasm consumes on average 37.0% less memory than JS. One possible reason is that Apple uses the JavaScriptCore engine for its Safari browser, which may have different memory allocation and resource optimization policies from other browsers (*e.g.,* Google utilizes the V8 free and open-source JS and Wasm engine

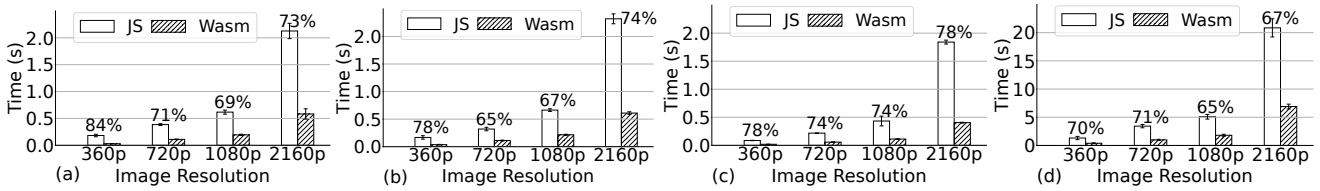

Figure 5: Comparison of processing time for object tracking: (a) Chrome and (b) Firefox on Galaxy S21 (Android), (c) Safari on iPhone 14 Pro (iOS), and (d) Edge on HoloLens 2. The resolution of reference images is 1024×768.

for Chrome). This indicates that while Wasm typically consumes more memory than JS, specific environments or implementations can flip this general trend.

(3) *Smartphones take up more memory when decoding and rendering 3D models of equivalent size compared to headsets.* Figure 4 shows that, given the same number of objects and identical 3D model sizes, Galaxy S21 consumes more decoding memory compared to Quest 2, and iPhone 14 Pro demands more memory in contrast to Hololens 2. The reason may be that being designed specifically for rendering 3D objects, XR headsets use more efficient rendering techniques or data structures that consume less memory.

CPU and GPU Utilization. We compare the CPU and GPU usage for JS and Wasm and present the detailed results in Appendix B. Our main observations are that compared to JS, Wasm does not significantly impact CPU and GPU utilization, and there exists a significant opportunity for Wasm to enhance GPU performance for Web-based XR.

4.2 Object Tracking

We evaluate the processing time of JS and Wasm for object tracking with different image resolutions on smartphones and HoloLens 2, as high-resolution images contain more data and thus require more processing power for analysis than low-resolution ones. In addition, to further explore the effectiveness of Wasm, we compare the processing times of object tracking for Wasm-enhanced and standalone applications. Note that this task is not suitable for VR, and thus we exclude Quest 2. Based on the results revealed in the experiments, we have the following key observations:

(1) *Wasm significantly reduces the processing time of object tracking compared to JS.* Figure 5 reports that compared to JS, Wasm, on average, reduces the processing time of object tracking by 74.3% on Chrome, 71.0% on Firefox using Galaxy S21, 76.0% on Safari using iPhone 14 Pro, and 68.3% on Edge using Hololens 2. Typically, object tracking necessitates the extraction of features from both captured and reference images, which calls for sophisticated algorithms and extensive matrix operations that Wasm can accelerate. This computational load could potentially slow down object tracking, a factor that becomes particularly pronounced on devices with less processing power such as Hololens 2. Thus, as shown in Figure 5(d), it has a 10× processing time compared to smartphones.

(2) *Wasm still has room for improving the performance of Web-based XR compared to standalone XR applications.* While the performance of Wasm is claimed to be close to native code [36], Table 1 indicates that there is still considerable potential for enhancing performance compared to a standalone XR application that implements the same object-tracking function with OpenCV. Currently, the processing time of Wasm is approximately twice as long as that of standalone applications for various image resolutions. This aligns with the findings of Haas *et al.* [36], where Wasm demonstrates similar performance gaps compared to standalone applications across a majority of tasks. The reason for the performance gap may be attributed to Wasm functioning within a sandbox environment [37] in the browser due to security considerations, which restricts its ability to leverage the entirety of system capabilities and resources fully. In contrast, standalone XR applications can directly harness the entire hardware and system resources of the device.

Resolution	Wasm	Standalone
360p	29.6±1.31	16.0±2.27
720p	110±5.98	37.8±3.58
1080p	194±11.7	73.0±8.95
2160p	583±96.4	257±33.6

Table 1: Comparison of processing time (ms) of object tracking for Wasm-enhanced and standalone applications.

4.3 Image Super-resolution

We analyze the processing time of image super-resolution by comparing Wasm with JS for different resolutions of input images on various devices and show the results in Figure 6. We have the following observation from Figure 6. Wasm can enhance the processing performance of image super-resolution to varying degrees (*e.g.,* 12.5–30.2%). For example, Wasm speeds up the processing time of image super-resolution, on average, by 23.2% on Chrome using Galaxy S21, 12.5% on the Oculus browser using Quest 2, 16.7% on Safari using iPhone 14 Pro, and 30.2% on Edge using Hololens 2, compared to JS. Nonetheless, the performance improvement may not be as substantial as that for object tracking, as depicted in Figure 5. The rationale is that object tracking, which involves feature extraction and matching, is inherently more computationally demanding than interpolation-based image super-resolution [53, 60]. This complexity provides greater opportunities for Wasm to optimize and reduce the processing time.

In addition, we compile the application employing different optimization levels of Wasm (*e.g.,*-O1, -O2, -O3, -Oz, -Os, and -Ofast) according to Yan *et al.* [69]. Our results indicate that the processing time of image super-resolution remains largely unaffected by these optimization levels.

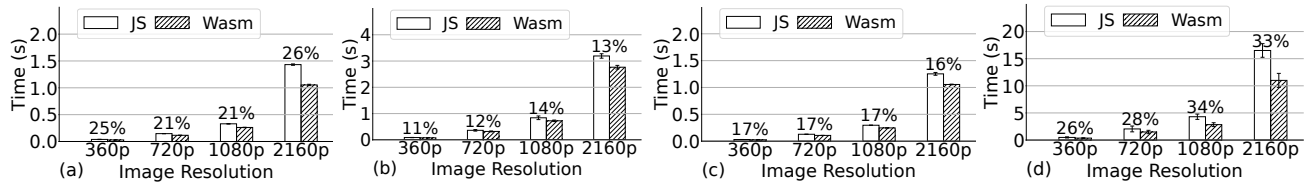

Figure 6: Comparison of processing time of image super-resolution: (a) Chrome on Galaxy S21 (Android), (b) Oculus browser on Quest 2, (c) Safari on iPhone 14 Pro (iOS), and (d) Edge on HoloLens 2. The upsampling ratio is 16×.

5 DISCUSSION

Limitations. As the first study for measuring the performance of Web-based XR accelerated by Wasm, our work bears the following limitations: (1) the adequacy of the chosen XR tasks and frameworks in representing the overall ecosystem and (2) the generalization of the measurement results.

WebAssembly has been designed to optimize a wide range of use cases, including numeric computation, data compression, cryptographic algorithms, image processing, *etc.* We focus on three extensively utilized XR-related tasks, namely 3D content loading and rendering, object tracking, and image super-resolution, which are all computation-intensive and could be accelerated by Wasm. Nevertheless, there are other tasks such as scene understanding, pose estimation, spatial mapping, and lighting estimation that all can potentially benefit from Wasm. Moreover, this study centers around the Magnum framework and the OpenCV library which are powerful yet flexible for our purpose. In the future, we plan to measure the performance improvements of other XR tasks introduced by Wasm and experiment with different XR frameworks such as the Wonderland Engine.

Regarding the generalization of our results, this study covers diverse browsers including Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, and the Oculus browser, and various mobile devices such as Android and iOS smartphones, Microsoft HoloLens 2 MR headset, and Oculus Quest 2 VR headset. However, we experiment with only Edge on HoloLens 2, the Oculus browser on Quest 2, and Safari on iPhone 14 Pro. Given that Web browsers and XR are undergoing rapid evolution, we will experiment with other combinations of browsers and devices once they are feasible.

Future Work. In addition to the aforementioned future work, we will explore the effectiveness of Wasm in optimizing the performance of WebGPU [44], the next generation of WebGL. GPU has been extensively used in XR applications not only for content rendering but also for speeding up deep-learning models such as object detection [25]. By accelerating WebGPU with Wasm, hopefully, we can enjoy a more immersive Web.

6 RELATED WORK

Mobile XR. Existing research work on mobile XR could be naturally divided into three categories: AR [25, 30, 42, 68, 72–74], VR [28, 46, 51, 67], and MR [35]. For example, Jaguar [73] is a mobile AR system that offers robust, flexible, and context-aware tracking and low-latency, accurate, and large-scale object recognition. Furion [46] is a mobile VR framework that splits the rendering pipeline for foreground interactions and background environment on the client and the server separately. DeepMix [35] is a lightweight framework for MR headsets that effectively combines edge-assisted 2D object detection and on-device estimations of 3D bounding boxes for real-world objects by leveraging depth data. In this paper, we measure the performance of Web-based XR, instead of standalone XR applications.

Web-based XR. The primary focus of current research on Web-based XR centers around AR [34, 38, 55–57] and VR [31, 32, 62, 64, 71]. Among them, Bi *et al.* [26] measured the performance of several popular Web-based XR frameworks, with a specific emphasis on 3D rendering. In addition, numerous recent endeavors have been made to utilize Wasm for optimizing Web-based XR [34, 57]. For example, Gottl *et al.* [34] presented an effective pose-tracking pipeline for standard Web browsers using HTML5 and Wasm. ARENA [57] is a platform that benefits from Wasm for secure and lightweight serverless-style computing. Compared to previous work, we conduct a first measurement study of the performance of Web-based XR accelerated by Wasm.

WebAssembly [36] has attracted remarkable interest from the research community. Several studies have been conducted to analyze the performance of Wasm and its applications [37, 43, 48–50, 69]. For example, Yan *et al.* [69] investigated the performance of Wasm applications relative to JS, by analyzing the contributing factors that influence their performance in various settings. WAIT [50] is a lightweight Wasm runtime designed for resource-constrained IoT devices, facilitating device-cloud integrated applications. In contrast, our primary emphasis lies in evaluating the effectiveness of Wasm in optimizing Web-based XR applications.

7 CONCLUSION

In this paper, we presented the first disciplined and empirical investigation to dissect the performance of Web-based XR escalated by WebAssembly. We conducted measurement studies with different metrics, including page load time, CPU/GPU utilization, and memory usage on diverse devices and browsers. Our experimental results yield intriguing observations and valuable insights, which reveal the current landscape of Web-based XR. We hope our work can furnish practical implications for different stakeholders and catalyze the future development of Web-based XR and WebAssembly.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Mallesham Dasari for their insightful comments. This work was supported in part by NSF CNS-2212296 and CNS-2235049.

REFERENCES

[1] 2008. Chrome Dev Tool. https://developer.chrome.com/docs/devtools/. [accessed on 09/24/2023].
[2] 2010. Emscripten. https://emscripten.org/. [accessed on 09/24/2023].
[3] 2010. Magnum. https://github.com/mosra/magnum. [accessed on 09/24/2023].
[4] 2010. three.js. https://github.com/mrdoob/three.js. [accessed on 09/24/2023].
[5] 2012. Sketchfab. https://sketchfab.com/feed. [accessed on 09/24/2023].
[6] 2013. babylon.js. https://github.com/BabylonJS/Babylon.js. [accessed on 09/24/2023].
[7] 2014. Cheerp. https://leaningtech.com/cheerp/. [accessed on 09/24/2023].
[8] 2015. A-Frame. https://github.com/aframevr/aframe. [accessed on 09/24/2023].
[9] 2015. Binaryen. https://github.com/WebAssembly/binaryen. [accessed on 09/24/2023].
[10] 2017. AR.js. https://github.com/AR-js-org/AR.js. [accessed on 09/24/2023].
[11] 2017. OpenCV. http://opencv.org/. [accessed on 09/24/2023].
[12] 2019. Microsoft HoloLens 2. https://www.microsoft.com/en-us/hololens. [accessed on 09/24/2023].
[13] 2020. Blazor WebAssembly. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor. [accessed on 09/24/2023].
[14] 2020. Oculus Quest 2. https://www.meta.com/quest/products/quest-2/. [accessed on 09/24/2023].
[15] 2020. WebXR Device API. https://www.w3.org/TR/webxr/. [accessed on 09/24/2023].
[16] 2020. Wonderland Engine. https://wonderlandengine.com/. [accessed on 09/24/2023].
[17] 2021. Safari Browser Install | Can You Install the Safari Browser on Android? https://www.lifewire.com/install-safari-browser-on-android-5078911. [accessed on 09/24/2023].
[18] 2021. Sketchfab Spiderman 3D Model. https://sketchfab.com/3d-models/spiderman-2002-movie-version-sam-raimi-6b6511850e2148b68ac33607af91574d. [accessed on 09/24/2023].
[19] 2022. Apple vs Android RAM management: Who does it better? https://www.androidauthority.com/apple-vs-android-ram-management-3100032/. [accessed on 09/24/2023].
[20] 2022. Apple WebXR | Apple is a massive force in AR. It's also been holding the technology back. https://www.protocol.com/entertainment/apple-webxr-ar-ios-iphone. [accessed on 09/24/2023].
[21] 2022. Magic Leap 2. https://www.magicleap.com/magic-leap-2. [accessed on 09/24/2023].
[22] 2022. Sketchfab Spider Gwen 3D Model. https://sketchfab.com/3d-models/spider-gwen-animated-b5af8af881744fa5aa419c25c96fe2ba. [accessed on 09/24/2023].
[23] 2023. "WebAssembly | Can I use ...? https://caniuse.com/wasm. [accessed on 09/24/2023].
[24] 2023. "WebXR Device API | Can I use ...? https://caniuse.com/webxr. [accessed on 09/24/2023].
[25] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, and Amit K. Roy-Chowdhury. 2019. Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality. In *Proceedings of ACM SenSys*. https://doi.org/10.1145/3356250.3360044
[26] Weichen Bi, Yun Ma, Deyu Tian, Qi Yang, Mingtao Zhang, and Xiang Jing. 2023. Demystifying Mobile Extended Reality in Web Browsers: How Far Can We Go?. In *Proceedings of ACM WWW*. https://doi.org/10.1145/3543507.3583329
[27] Federico Biggio. 2020. Protocols of Immersive Web: WebXR APIs and the AR Cloud. In *Proceedings of ACM Conference on Web Studies*. https://doi.org/10.1145/3423958.3423965
[28] Kevin Boos, David Chu, and Eduardo Cuervo. 2016. FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization. In *Proceedings of ACM MobiSys*. https://doi.org/10.1145/2906388.2906418
[29] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. Shapenet: An Information-Rich 3D Model Repository. https://arxiv.org/abs/1512.03012
[30] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. 2018. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Proceedings of ACM SenSys*. https://doi.org/10.1145/3274783.3274834
[31] Ruizhi Cheng, Nan Wu, Songqing Chen, and Bo Han. 2022. Reality Check of Metaverse: A First Look at Commercial Social Virtual Reality Platforms. In *Proceedings of the IEEE Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*.
[32] Ruizhi Cheng, Nan Wu, Matteo Varvello, Songqing Chen, and Bo Han. 2022. Are We Ready for Metaverse? A Measurement Study of Social Virtual Reality Platforms. In *Proceedings of the ACM IMC*. https://dl.acm.org/doi/pdf/10.1145/3517745.3561417
[33] Eduardo Cuervo, Krishna Chintalapudi, and Manikanta Kotaru. 2018. Creating the Perfect Illusion: What will it take to Create Life-Like Virtual Reality Headsets?. In *Proceedings of ACM HotMobile*. https://doi.org/10.1145/3177102.3177115
[34] Fabian Göttl, Philipp Gagel, and Jens Grubert. 2018. Efficient Pose Tracking from Natural Features in Standard Web Browsers. In *Proceedings of ACM Conference on 3D Web Technology*. https://doi.org/10.1145/3208806.3208815
[35] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2022. DeepMix: Mobility-aware, Lightweight, and Hybrid 3D Object Detection for Headsets. In *Proceedings of ACM MobiSys*. https://doi.org/10.1145/3498361.3538945
[36] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. 2017. Bringing the Web up to Speed with WebAssembly. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/3062341.3062363
[37] Aaron Hilbig, Daniel Lehmann, and Michael Pradel. 2021. An Empirical Study of Real-World WebAssembly Binaries: Security, Languages, Use Cases. In *Proceedings of ACM WWW*. https://doi.org/10.1145/3442381.3450138
[38] Tanner Hobson, Jeremiah Duncan, Mohammad Raji, Aidong Lu, and Jian Huang. 2020. Alpaca: AR Graphics Extensions for Web Applications. In *Proceedings of IEEE VR*. https://doi.org/10.1109/VR46266.2020.00036
[39] Hugues Hoppe. 1996. Progressive Meshes. In *Proceedings of ACM SIGGRAPH*. https://doi.org/10.1145/237170.237216
[40] Hugues Hoppe. 1997. View-Dependent Refinement of Progressive Meshes. In *Proceedings of ACM SIGGRAPH*. https://doi.org/10.1145/258734.258843
[41] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. 2022. ILLIXR: An Open Testbed to Enable Extended Reality Systems Research. *IEEE Micro* 42, 4 (2022), 97–106. https://doi.org/10.1109/MM.2022.3161018
[42] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. 2015. OverLay: Practical Mobile Augmented Reality. In *Proceedings of ACM MobiSys*. https://doi.org/10.1145/2742647.2742666
[43] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. 2019. Not so Fast: Analyzing the Performance of Webassembly vs. Native Code. In *Proceedings of USENIX ATC*. https://dl.acm.org/doi/10.5555/3358807.3358817
[44] Benjamin Kenwright. 2022. Introduction to the WebGPU API. In *Proceedings of ACM SIGGRAPH*. https://doi.org/10.1145/3532720.3535625
[45] Bohdan B Khomtchouk. 2021. WebAssembly Enables Low Latency Interoperable Augmented and Virtual Reality Software. https://doi.org/10.48550/arXiv.2110.07128
[46] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. 2017. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. In *Proceedings of ACM MobiCom*. https://doi.org/10.1145/3117811.3117815
[47] Royson Lee, Stylianos I Venieris, Lukasz Dudziak, Sourav Bhattacharya, and Nicholas D Lane. 2019. MobiSR: Efficient On-Device Super-Resolution through Heterogeneous Mobile Processors. In *Proceedings of ACM MobiCom*. https://doi.org/10.1145/3300061.3345455
[48] Daniel Lehmann, Johannes Kinder, and Michael Pradel. 2020. Everything Old is New Again: Binary Security of WebAssembly. In *Proceedings of USENIX Conference on Security Symposium*. https://dl.acm.org/doi/10.5555/3489212.3489225
[49] Daniel Lehmann and Michael Pradel. 2019. Wasabi: A Framework for Dynamically Analyzing WebAssembly. In *Proceedings of ACM ASPLOS*. https://doi.org/10.1145/3297858.3304068
[50] Borui Li, Hongchang Fan, Yi Gao, and Wei Dong. 2022. Bringing WebAssembly to Resource-constrained IoT Devices for Seamless Device-cloud Integration. In *Proceedings of ACM MobiSys*. https://doi.org/10.1145/3498361.3538922
[51] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering. In *Proceedings of ACM MobiSys*. https://doi.org/10.1145/3210240.3210313
[52] Bruce D Lucas and Takeo Kanade. 1981. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the Joint Conference on Artificial Intelligence*.
[53] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *Journal of Computer Vision* 60, 2 (2004), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94
[54] Blair MacIntyre and Trevor F Smith. 2018. Thoughts on the Future of WebXR and the Immersive Web. In *Proceedings of IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. https://doi.org/10.1109/ISMAR-Adjunct.2018.00099
[55] Nitika Nitika, Tanuja Kumari Sharma, Saumya Rajvanshi, and Keshav Kishore. 2021. A Study of Augmented Reality Performance in Web Browsers (WebAR). In *Proceedings of International Conference on Computational Methods in Science & Technology (ICCMST)*. https://doi.org/10.1109/ICCMST54943.2021.00065
[56] Michael Oduor and Timo Perälä. 2021. Interactive Urban Play to Encourage Active Mobility: Usability Study of a Web-Based Augmented Reality Application. *Frontiers in Computer Science* 3 (2021), 67. https://doi.org/10.3389/fcomp.2021.706162
[57] Nuno Pereira, Anthony Rowe, Michael W Farb, Ivan Liang, Edward Lu, and Eric Riebling. 2021. ARENA: The Augmented Reality Edge Networking Architecture. In *Proceedings of IEEE International Symposium on Mixed and Augmented Reality*

(ISMAR). https://doi.org/10.1109/ISMAR52148.2021.00065
[58] Xiuquan Qiao, Pei Ren, Schahram Dustdar, Ling Liu, Huadong Ma, and Junliang Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. *Proceedings of the IEEE* 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105
[59] Alec Jacobson Qingnan Zhou. 2016. Thingi10k: A Dataset of 10,000 3D-Printing Models. https://arxiv.org/abs/1605.04797
[60] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An Efficient Alternative to SIFT or SURF. In *Proceedings of ICCV.* https://doi.org/10.1109/ICCV.2011.6126544
[61] Maximilian Speicher, Brian D Hall, and Michael Nebeling. 2019. What is Mixed Reality?. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI).* https://doi.org/10.1145/3290605.3300767
[62] Wenxin Sun, Mengjie Huang, Rui Yang, Jingjing Zhang, Liu Wang, Ji Han, and Yong Yue. 2020. Workload, Presence and Task Performance of Virtual Object Manipulation on WebVR. In *Proceedings of IEEE Conference on Artificial Intelligence and Virtual Reality (AIVR).* https://doi.org/10.1109/AIVR50618.2020.00073
[63] Minna Vasarainen, Sami Paavola, and Liubov Vetoshkina. 2021. A Systematic Literature Review on Extended Reality: Virtual, Augmented and Mixed Reality in Working Life. *International Journal of Virtual Reality* 21, 2 (2021), 1–28. https://doi.org/10.20870/IJVR.2021.21.2.4620
[64] Chao Wang, Shuanq Lianq, and Jinyuan Jia. 2018. Immersing Web3D Furniture into Real Interior Images. In *Proceedings of IEEE VR.* https://doi.org/10.1109/VR.2018.8446341
[65] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. 2013. Demystifying Page Load Performance with WProf.. In *Proceedings of USENIX NSDI.* https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/wang_xiao
[66] Nannan Xi, Juan Chen, Filipe Gama, Marc Riar, and Juho Hamari. 2022. The Challenges of Entering the Metaverse: An Experiment on the Effect of Extended Reality on Workload. *Information Systems Frontiers* (2022), 1–22. https://doi.org/10.1007/s10796-022-10244-x
[67] Chenhao Xie, Xie Li, Yang Hu, Huwan Peng, Michael Taylor, and Shuaiwen Leon Song. 2021. Q-VR: System-Level Design for Future Mobile Collaborative Virtual Reality. In *Proceedings of ACM ASPLOS.* https://doi.org/10.1145/3445814.3446715
[68] Jingao Xu, Guoxuan Chi, Zheng Yang, Danyang Li, Qian Zhang, Qiang Ma, and Xin Miao. 2021. FollowUpAR: Enabling Follow-up Effects in Mobile AR Applications. In *Proceedings of ACM MobiSys.* https://doi.org/10.1145/3458864.3467675
[69] Yutian Yan, Tengfei Tu, Lijian Zhao, Yuchen Zhou, and Weihang Wang. 2021. Understanding the Performance of WebAssembly Applications. In *Proceedings of ACM IMC.* https://doi.org/10.1145/3487552.3487827
[70] Chih-Yuan Yang, Jia-Bin Huang, and Ming-Hsuan Yang. 2010. Exploiting Self-Similarities for Single Frame Super-Resolution. In *Proceedings of the 10th Asian Conference on Computer Vision.*
[71] Yang Yang, Lei Feng, Xiaoyu Que, Fanqin Zhou, and Wenjing Li. 2022. Energy-and Quality-Aware Task Offloading for WebVR Service in Terminal-Aided Mobile Edge Network. *IEEE Transactions on Vehicular Technology* 71, 8 (2022), 8825–8838. https://doi.org/10.1109/TVT.2022.3173709
[72] Juheon Yi and Youngki Lee. 2020. Heimdall: Mobile GPU Coordination Platform for Augmented Reality Applications. In *Proceedings of ACM MobiCom.* https://doi.org/10.1145/3372224.3419192
[73] Wenxiao Zhang, Bo Han, and Pan Hui. 2018. Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking. In *Proceedings of ACM MM.* https://doi.org/10.1145/3240508.3240561
[74] Wenxiao Zhang, Bo Han, and Pan Hui. 2022. SEAR: Scaling Experiences in Multi-user Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics* 28, 5 (2022), 1982–1992. https://doi.org/10.1109/TVCG.2022.3150467

APPENDICES

A ETHICS

This work does not raise any ethical issues.

B CPU AND GPU USAGE

Figure 7 shows the comparison of CPU and GPU usage for JS and Wasm across the common browsers on three devices: Chrome on Samsung Galaxy S21 (Android phone), the Oculus browser on Oculus Quest 2, and Edge on Microsoft Hololens 2. Although we can load 3D content on Safari and accelerate it with Wasm, we are unable to render content as the WebXR API is not supported on Safari [24]. Based on the results revealed in this figure, we have the following findings:

(1) *Wasm does not lead to much difference in CPU and GPU utilization compared to JS.* Although CPU and GPU usage patterns vary drastically between stages, Wasm can maintain almost the same usage patterns as JS at different stages. The reason is that the CPU usage is either close to 100% during content decoding or low when rendering content (which is taken care of by GPUs). However, we have not yet explored how to optimize and accelerate GPU operations with Wasm.

(2) *The GPU utilization still has a large room for improving the decoding process.* From Figure 7, we can see that during content decoding, the CPU utilization is maximized across all devices. Hololens 2 makes extensive use of GPU resources, while Galaxy S21 and Quest 2 utilize only a small amount of their GPU capabilities. Hence, there exists a considerable opportunity to speed up the decoding process by improving the utilization of GPUs on these devices with Wasm.

(3) *The GPU utilization is usually high during the rendering process.* Figure 7 shows that Galaxy S21 and Hololens 2 have higher utilization of GPUs than that of CPUs during content rendering, which is GPU-intensive and has not been accelerated by today's Wasm. For Quest 2, both CPU and GPU utilization are low, which could be due to the low framerate (*e.g.,* 7 frames per second for eight objects) caused by the complex models.

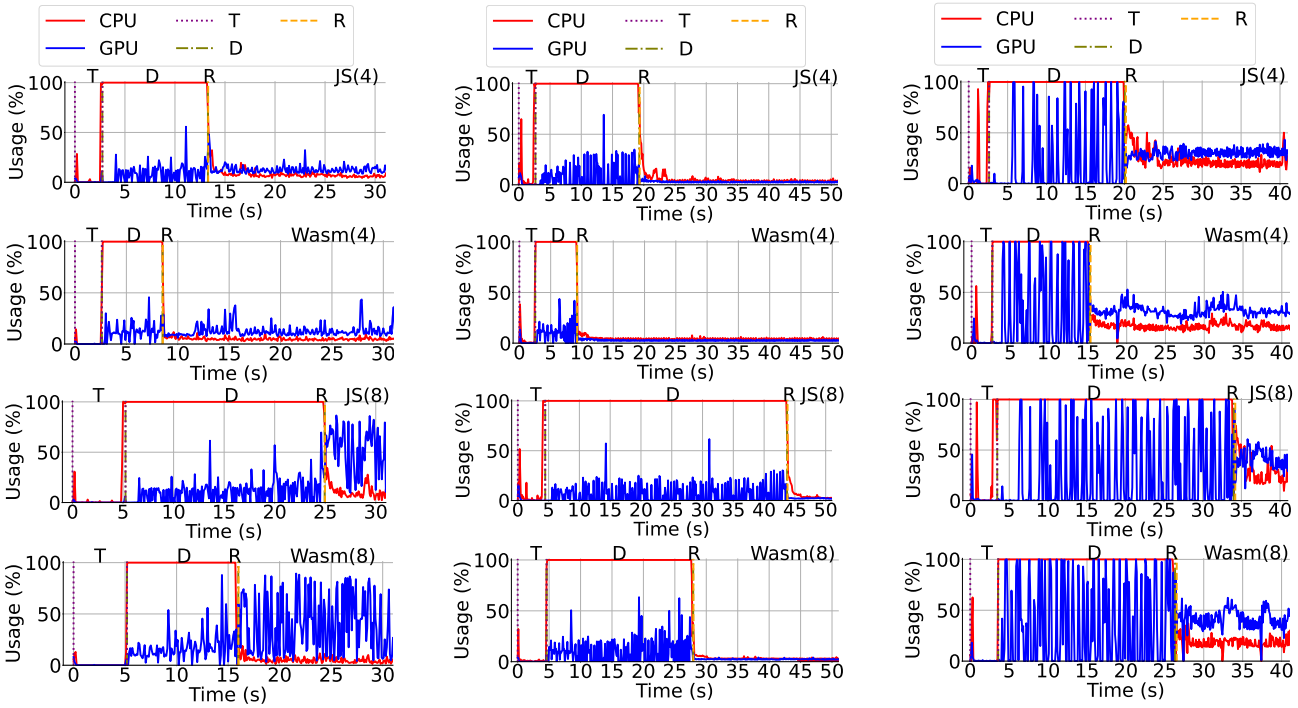

Figure 7: Comparison of CPU and GPU usages for JavaScript (JS) and WebAssembly (Wasm): (left) Chrome on Galaxy S21 (Android), (middle) Oculus browser on Quest 2, and (right) Edge on Hololens 2. The four rows are JS with four objects, Wasm with four objects, JS with eight objects, and Wasm with eight objects. T: Transmission, D: Decoding, and R: Rendering.