DOI: 10.12700/AIS.2025.015

Web-Based VR Application Development: Opportunities and Challenges

István László Jakab
Obuda University, Alba Regia Faculty Székesfehérvár, Hungary
istvan.jakab@stud.uni-obuda.hu

Klaudia Talabérné Dr. Kulcsár
Obuda University, Alba Regia Faculty Székesfehérvár, Hungary
kulcsar.klaudia @amk.uni-obuda.hu

Éva Hajnal
Obuda University, Alba Regia Faculty Székesfehérvár, Hungary
hajnal.eva@amk.uni-obuda.hu

***Abstract*—Virtual reality (VR) technology has rapidly matured, yet traditional desktop-based VR systems remain constrained by hardware limitations and platform dependencies. This paper explores the evaluation of web-based VR solutions, emphasizing cross-platform technologies such as WebXR, WebGL, WebAssembly, and WebGPU. Through a detailed case study of a VR game developed using Wonderland Engine, it was demonstrated how modern browser APIs deliver immersive VR experiences efficiently. The study outlines key technical challenges, including optimization strategies, user experience (UX) design considerations, and trade-offs compared to native VR development platforms.**

***Index Terms*—WebXR, WebAssembly, WebGL, WebGPU, VR optimization, Wonderland Engine, browser-based virtual reality.**

I. Introduction

Traditional VR development primarily utilizes specialized hardware and native engines such as Unity and Unreal Engine, limiting accessibility and scalability. Web-based VR, facilitated by technologies like WebXR and WebGL, significantly lowers entry barriers, providing broad, cross-platform access without additional installations. This paper investigates these technologies, highlighting their potential and challenges.

The technologies discussed in this paper form a layered and complementary ecosystem rather than competing solutions. WebXR acts as the high-level interface for accessing VR and AR devices and handling spatial tracking. WebGL provides the graphics rendering layer that WebXR relies on for displaying 3D content inside the browser. WebGPU is an emerging next-generation graphics API that extends or replaces WebGL with lower-level GPU access and improved performance. WebAssembly (Wasm) operates alongside JavaScript, enabling near-native execution speed for compute-intensive parts of the VR application. In this hierarchy, WebXR sits on top of WebGL or WebGPU for rendering, while WebAssembly accelerates logic and physics computations under the same framework.

Web-based VR engines offer rapid deployment and reduced development overhead. However, they lag behind Unity or Unreal Engines in rendering fidelity and advanced simulation capabilities. Yet, their browser-native nature avoids platform-specific builds and installers.

WebGL enables browsers to render real-time 3D graphics efficiently, while WebXR provides a device-agnostic interface to immersive displays and input tracking. Together, these technologies offer a viable, lightweight alternative to traditional native VR engines.

WebAssembly (Wasm) allows compiled code to run at near- native speeds within web browsers. Coupled with WebXR and advanced rendering libraries, such as Wonderland Engine, WebAssembly supports complex, performant simulations. SIMD extensions further optimize critical matrix and physics computations.

This paper investigates the issues connected to the WebVR development. A critical question for web-based VR is performance across different platforms. The second question is the program's ease of use and input latency. The third issue is loading and network speed.

II. Methods

The method applied in this study was the development of a sample application demonstrating the use of web-based VR technologies. A prototype was created using Wonderland Engine and deployed with browser-native tools to evaluate cross-platform performance and usability.The sample application was a 3D reinterpretation of the Chrome Dino game built in Wonderland Engine. The virtual environment consisted of a desert track populated with interactive obstacles. The player collects points by dodging obstacles using VR headsets and controllers.

The project architecture included:

- Navigation: Continuous forward movement with lateral interaction.
- Scoring: Real-time score updates rendered in 3D.
- Feedback: Collision triggers providing visual and sound responses.

The development used Docker for platform independence and GitHub Actions for automated deployment. A Tampermonkey userscript allowed runtime parameter injection without modifying the core application. This approach provides similar flexibility to Unity's runtime scripting or developer console, where variables and behaviors can be adjusted dynamically

without recompiling the project. In both cases, it enables rapid iteration and testing of new parameters directly within the deployed environment.

During the development of the sample application, it was aimed to highlight not only the game logic and implementation but also to demonstrate the integration of modern development practices. Version control, particularly GitHub-based work- flows, was given a central role, which is essential in modern projects involving multiple developers. While version control is indispensable, the use of Docker and CI/CD pipelines can be considered optional. Nevertheless, these technologies greatly contribute to the structure, scalability, and maintainability of the project.

A. Application Structure

1) Navigation: The player moves in a VR environment and must avoid obstacles. Currently, movement is triggered by pressing one of the controller's triggers.

2) *Scoring:* Players earn points by successfully avoiding obstacles. Although a scoring system is implemented, single-player mode is currently supported.

3) Simple Interactions: User interactions are handled via VR controllers or keyboard. The game can be played both with and without a VR headset.

The application comprises not only game elements but also logical modules that manage VR interactions, camera motion, and user feedback. Each object in the scene has components such as a "Jump Handler" or "Score Counter." A diagram of the scene's structure would be helpful to illustrate the relationships between the player, obstacles, ground, and control UI. Further improvements could involve error-handling for missing assets or unsupported devices.

The game is a VR version of the classic Chrome Dino game, where players navigate a VR environment to dodge obstacles. It is available on the itch.io platform for anyone with a compatible browser. The application is available.[15]

Docker is an open-source platform that allows applications to be packaged and run in isolated containers. These containers include all necessary files, libraries, and dependencies, making the software independent of the host system.

Docker uses a shared OS kernel to manage resources efficiently. Containers are created from image files defined in a Dockerfile. This approach also applies to web-based VR applications, enabling reproducible environments regardless of the development machine.

In a WebXR project, it is practical to containerize the backend API, static asset server, and build pipeline using

Docker Compose. These components can be started in sync, streamlining deployment. Isolation enhances security, and CI/CD pipelines can automate updates.

Docker image for this application was also created, based on Ubuntu Linux. It installs required dependencies (e.g., NPM), downloads the application from GitHub, and automatically builds a release.

B. Browser-Based Development with GitHub and VS Code

Web-based technologies allow full development in the browser. Developers can upload their codebase to GitHub and use the web version of Visual Studio Code (VS Code) directly.

By changing the ".com" in a GitHub repository URL to ".dev", VS Code opens in the browser:

Original:https://github.com/TheOnlyCoffeeman/DinoGameR
Browser:https://github.dev/TheOnlyCoffeeman/DinoGameR

This setup supports editing, commits, pulls, and pushes as if working locally. It is ideal for WebXR development, reducing environment setup and increasing flexibility.

Platforms like GitHub Codespaces or Gitpod offer full-featured cloud environments. The web-based VS Code pro- vides an integrated terminal, editor, and version control inter- face. This approach allows mobile editing, quick bug fixes, or CI oversight without needing local tools.

C. Project Structure and Key Files

- **js/**: JavaScript files for game logic and interactions.
- **models/**: 3D models (e.g., dinosaurs) in .glb/.gltf format.
- **static/sfx/**: Sound effects for events like jumping or collisions.
- **test/**: Scripts and files for manual or automated testing.
- **DinoGame.wlp**: Wonderland Engine project file with scene configurations.
- **README.md**: Project description, usage instructions, and metadata.
- **Sources.txt**: References to models, sounds, or external assets.
- **package.json, package-lock.json**: Node.js dependencies and metadata.

D. Automated Build with GitHub Actions

Automation is achieved with a `build.yaml` file placed in `.github/workflows`. This defines steps for GitHub Actions to perform after code is pushed:

1) Code is pushed to GitHub.
2) GitHub Actions triggers the workflow.
3) Docker-based build installs dependencies and ensures consistency.
4) A release is automatically published on GitHub.

This automation is highly beneficial for web-based VR projects, reducing release time and improving transparency.

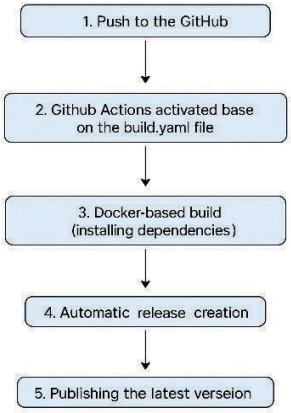

Figure 1. Automated build workflow

E. Experimental Tampermonkey Script

To showcase the sample app's usability, a Tampermonkey script was created that adds a link to the VR game on Obuda University's AMK website.

Tampermonkey customizes web- pages by injecting new elements or altering DOM content. The script activates only on amk.uni-obuda.hu, listening for the DOMContentLoaded event before modifying the DOM. It inserts a new <a> element pointing to the VR game (e.g., GitHub Pages URL), using grant none for secure execution.

Figure 2. Tampermonkey script in use, demonstrating the experimental injection of a VR application link into the Obuda University website to simulate its future static integration.

The purpose of this modification was to demonstrate how an academic prototype can be integrated dynamically into an existing institutional webpage without modifying server-side code. By inserting the link locally through Tampermonkey, the approach remains non-intrusive and reversible while showcasing the potential of browser-side customization for rapid demonstration and educational outreach. This Tampermonkey integration was only an experimental method used to simulate the connection of the application with the Obuda University website. In a persistent implementation, the same functionality would be realized by embedding the link or script statically into the website's source code rather than injecting it through a browser extension. The game was tested on a Pico 4 headset, a desktop PC, and a mid-range Android smartphone using Chrome. The desktop configuration featured an Intel Core i7 CPU, 16 GB RAM, and an NVIDIA RTX 3060 GPU, running Windows 11 with the latest Chromium build (116.0.5793.0). The Pico 4 standalone headset operated on the Snapdragon XR2 platform, while the smartphone used a Snapdragon 778G processor with 8 GB RAM. Additionally, a tablet test (Samsung Galaxy Tab S8) was conducted to evaluate touch-based navigation and browser compatibility in larger-screen mobile environments.

III. Results and Discussion

To ensure reproducibility and clarity of performance evaluation, several measurement metrics were defined explicitly. The Lighthouse score was used as an automated web-performance metric generated by Google's Lighthouse tool, measuring page load efficiency, rendering performance, and accessibility on a 0–100 scale. User experience (UX) was evaluated subjectively based on participants' responses to a standardized 1–10 scale questionnaire assessing visual smoothness, responsiveness, and ease of interaction. Input latency was quantified by analyzing recorded test footage frame by frame, measuring the elapsed time between input activation (e.g., trigger press) and the corresponding on-screen movement. All performance tests were conducted at a resolution of 1920×1080 pixels on desktop and 1832×1920 per eye on the Pico 4 headset. These definitions ensure consistent interpretation of the metrics presented in this section.

The first question is the performance during software execution.

A. Technical Performance

Key metrics: FPS, response time, memory usage, CPU load.

Table 1. CPU Usage Measurements

Device	Avg. CPU Load	Frame Render Time
Desktop PC	24%	7.1 ms
Pico 4	46%	12.8 ms
Android Phone/Tablet	58%	19.3 ms

The desktop performed best with stable 90 FPS, 2.1s load time, and low CPU usage. Pico 4 achieved 72 FPS, with longer load times due to hardware limits. Android had the greatest challenges (45 FPS average), limited by its mobile GPU.

The limitation is caused by the inherent constraints of mobile GPUs and the architectural overhead introduced by web-based execution through Wonderland Engine. Although the engine compiles to WebAssembly, allowing near native performance, the application still runs within the browser's sandboxed environment, where rendering operations are mediated by WebGL or WebXR APIs rather than direct native GPU access. Mobile GPUs, optimized for low power consumption and thermal stability, are unable to sustain the same shader complexity, texture resolution, and draw call volume achievable on desktop class hardware. This limitation is further amplified by browser levelresource management, thermal throttling, and memory allocation

restrictions, which prevent full utilization of GPU resources. Consequently, while Wonderland Engine effectively minimizes overhead through its WebAssembly runtime, web-based VR applications on mobile platforms remain constrained by hardware efficiency trade offs and browser API abstraction layers.

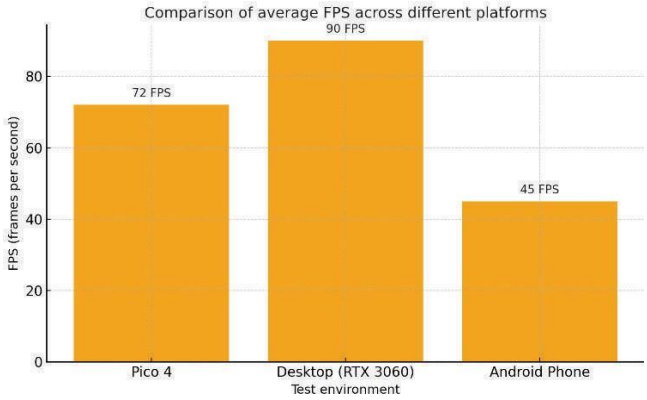

Figure 3. Average FPS comparison across different systems, illustrating performance variations between desktop, standalone VR headset, and mobile devices.

B. Interactions

User navigation time (to first obstacle) and input latency (e.g., jump response time) were measured. Desktop offered excellent UX with low latency and high refresh rates (rated 9/10). Pico 4 performed well, though controller delay was noticeable (42 ms). Mobile had higher input lag (63 ms), leading to a lower experience rating (6.8/10).

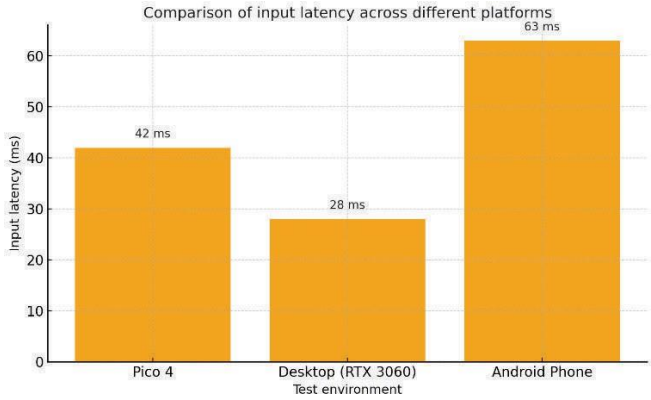

Figure 4. Input latency measurements showing the response delay between user actions and visual feedback across desktop, VR headset, and mobile platforms.

C. Web Performance and Optimization

Due to the web-based nature, we also assessed network and loading performance. The asset bundle size was 28.4 MB across platforms, with over 50 HTTP requests generated. Optimization (e.g., compression, sprite sheets) could reduce this.

Lighthouse scores: desktop 92, mobile 75—affected by CPU usage and resource limits.

D. Network Performance Evaluation

Performance varies based on network speed and reliability. Compressed textures and binary scene formats help, but poor networks degrade UX.

Caching (e.g., via Service Workers) or preloading using PWA frameworks is recommended to mitigate these issues.

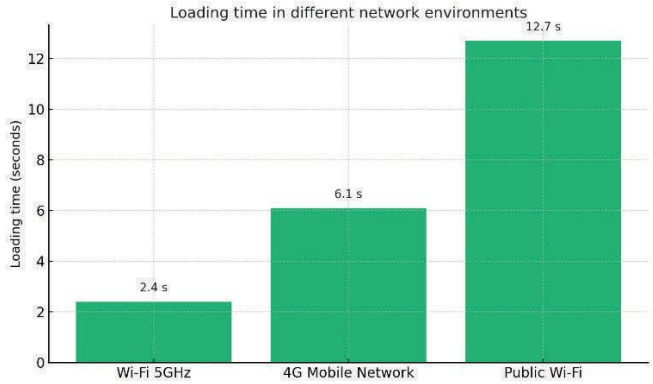

Figure 5. Network performance graph showing load times and latency variations across different network types and conditions.

TABLE 2. NETWORK PERFORMANCE MEASUREMENTS

Network	Load Time (s)	Packet Loss (%)	Latency (ms)
Wi-Fi 5GHz	2.4	0.1	35
4G Mobile	6.1	0.4	88
Public Wi-Fi (Weak)	12.7	2.3	160

IV. Performance and UX Considerations

Testing was conducted on the Pico 4 VR headset. Key metrics included:

- Average 90 FPS frame rate.
- 11 ms frame render time.
- Under 5 seconds initial load time. Optimizations applied:
- **Runtime batching:** Draw call minimization.
- **Basis Universal:** Efficient runtime texture decompres- sion.
- **Data-Oriented Design:** Cache-optimized data structures.

User feedback confirmed intuitive controls, engaging immersion, and responsiveness.

V. Conclusion

Web-based VR has evolved into a viable and increasingly practical platform for immersive content delivery. The case study and measurements conducted across multiple devices, including a standalone VR headset, a desktop PC, and a mid-range smartphone to demonstrate those modern web technologies like WebXR, WebGL 2.0, WebAssembly, and optimized rendering pipelines can deliver responsive, real-time VR experiences directly through the browser.

The testing results revealed that even on limited hardware like mobile phones, acceptable frame rates and input latencies can be achieved. Although performance dropped on mobile devices (averaging 45 FPS), the experience remained functional, which is notable given the reduced GPU capabilities and power constraints.

Latency measurements also supported the platform's viability: input delays were low enough on PC (under 20 ms) and moderate on the Pico 4 (around 42 ms), with tolerable degradation on smartphones. These values, while slightly higher than those on native platforms, still allow for a smooth gameplay experience in casual or educational settings. Importantly, the system remained responsive and consistent across platforms, demonstrating strong cross-device compatibility, since it is one of the major goals of web-based VR.

From a development perspective, the project relied on browser-integrated technologies and open standards such as the WebXR Device API for VR hardware access, Three.js and Wonderland Engine for scene rendering and logic management, and glTF 2.0 with Basis Universal compression for efficient asset streaming. Additional APIs, including the Web Audio API, Gamepad API, and Pointer Lock API were used to enable immersive audio, input tracking, and natural interaction. Performance-critical components were compiled to WebAssembly, while asynchronous data handling and rendering pipelines utilized Web Workers and OffscreenCanvas to minimize main-thread blocking.

The automated deployment pipeline via GitHub Actions, combined with containerization using Docker, ensured platform independent reproducibility and scalable distribution, further validating the feasibility of using web-native toolchains in professional VR development workflows.

Future research should focus on improving graphical fidelity via WebGPU, integrating machine learning–based personalization or interaction analytics, and expanding toward multiplayer or collaborative VR sessions. These could be powered by WebRTC, which enables real-time peer-to-peer communication with low latency for synchronized VR environments. Complementary technologies such as WebSocket and WebTransport can provide reliable data channels for multiplayer state management, while WebCodecs supports efficient streaming of encoded media between browsers. Together, these protocols can facilitate the creation of shared, persistent virtual spaces accessible directly from the web, without external plugins or native applications.

References

[1] Mozilla Developers, "WebXR Device API," https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API, 2025.

[2] WebAssembly.org, "Introduction to WebAssembly," https: //webassembly.org, 2025.

[3] Wonderland Engine Docs, "Performance Optimization Techniques," https://wonderlandengine.com/docs, 2025.

[4] Khronos Group, "WebGL Overview," https://www.khronos.org/webgl/, 2025.

[5] Google Developers, "Basis Universal Texture Compression," https:// github.com/BinomialLLC/basis_universal, 2025.

[6] GitHub Actions Documentation, "Automating Workflows," https://docs. github.com/actions, 2025.

[7] Ivan Sutherland, "The Ultimate Display," Proc. IFIP Congress, 1965.

[8] Sutherland and Sproull, "A Head-Mounted Three Dimensional Display," AFIPS, 1968.

[9] Jaron Lanier, "Virtual Reality and Its Discontents," VPL Research, 1990s.

[10] E. Lengyel, "Mathematics for 3D Game Programming and Computer Graphics," 3rd ed., 2012.

[11] M. Pharr et al., "Physically Based Rendering," 3rd ed., Morgan Kauf- mann, 2016.

[12] Tampermonkey.net, "Userscript Manager for Browsers," https://www. tampermonkey.net, 2025.

[13] Chrome Developers, "WebGPU Explainer," https://developer.chrome. com/docs/webgpu/, 2025.

[14] F. Gaertner, "Introduction to Data-Oriented Design," Game Program- ming Patterns, 2020.

[15] C. S. Horváth, B. Lampert, and A. Pongrácz, "Opportunities of VR for teaching history," Acta Polytechnica Hungarica, vol. 21, no. 3, pp. 143–162, 2024.

[16] K. Takács and T. Haidegger, "Eye gaze tracking in robot-assisted minimally invasive surgery: a systematic review of recent advances and applications," Acta Polytechnica Hungarica, vol. 21, no. 10, pp. 393–411, 2024.

[17] Y. Wu, A. Nagy, Z. Rajnai, and B. Fregan, "Advancing digital education: Technologies, opportunities, challenges, and future directions," Acta Polytechnica Hungarica, vol. 22, no. 12, 2025.

[18] https://theonlycoffeeman.itch.io/dino-game-vr