UDC 004.056 **doi: 10.26906/SUNZ.2026.2.146**

Mykyta Matvieiev

National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine

PERFORMANCE EVALUATION OF SCENE LOADING OPTIMIZATION IN A WEBAR APPLICATION

Abstract. Relevance. WebAR is rapidly evolving, however, it faces the challenge of high computational loads on devices. The synchronous loading of heavy 3D models during scene initialization often leads to delays and blocks the browser's main execution thread. Applying comprehensive optimization methods (such as transitioning to the GLB format and utilizing Draco geometry compression) is crucial for ensuring stability, yet their implementation requires an objective quantitative evaluation. **The aim of this study** is to evaluate the performance of WebAR application scene loading optimization during the initialization stage**. The object of the research** is the initialization and 3D content loading process within a client WebAR application running on a laptop. **The subject of the research** encompasses the performance metrics of the WebAR application's optimization methods. **Conclusion.** Based on the results of instrumental profiling, the high efficiency of transitioning to the GLB format and employing Draco compression has been confirmed. A 47.7% reduction in the peak allocated JS Heap memory was recorded. The overall scene initialization time was reduced by 30.5%, dropping from 4749 to 3298 ms. Furthermore, the main thread idle time was significantly decreased by 45%, and the parsing phase duration by 66.2%, which successfully eliminated critical bottlenecks in the 3D asset preparation pipeline.

Keywords: web, augmented reality, webAr, performance evaluation, optimization, Draco, Angular.

Introduction

Relevance. Digital technologies have become an integral part of modern society, transforming the ways we communicate, learn, and consume information. One of the most dynamic technologies is augmented reality (AR), which combines virtual digital content with the physical environment in real-time, providing a new level of interactivity [1, 2]. WebAR technology, which implements augmented reality functionality directly through a web browser, has experienced significant development. This approach lowers barriers to entry for users and simplifies the integration of 3D content into digital services, shaping a trend where web resources with AR capabilities are becoming the new standard of interaction [3–5].

Despite a number of significant advantages, WebAR is characterized by an increased computational load on the client device. The fundamental contradiction lies in the fact that underlying web technologies were primarily created to handle 2D content, whereas WebAR requires the continuous execution of resource-intensive operations: 3D graphics processing and spatial tracking. Implementing these processes through standard browser mechanisms leads to a decrease in performance, manifested by a drop in frame rates, blocking of the main execution thread, and an overall inefficient use of hardware resources [6–9]. Significant rendering delays mostly arise from the use of excessively large 3D models and synchronous data exchange mechanisms. In this mode, the client is forced to wait for the transmission of the entire model to complete before its initialization, which, combined with the instability of mobile networks, can lead to critical freezing of the application interface.

The solution to this problem is the optimization of 3D content and loading architecture. Utilizing compression technologies significantly reduces file sizes, improving the efficiency of their transmission through the browser. From the perspective of rendering performance, the glTF format and its binary variant, GLB, ensure the most efficient scene loading. To maximize results, these formats are combined with the Draco algorithm, which employs quantization and delta encoding for the spatial compression of geometry vertex attributes [10, 11]. Furthermore, transitioning to an asynchronous data transmission method eliminates network congestion, allowing information to be processed in chunks and avoiding blocking [12].

However, the theoretical justification and practical implementation of such optimization solutions require an objective quantitative evaluation of their actual effectiveness. Traditionally, analyzing the performance of a web application relies on examining key web metrics: total loading time, interface responsiveness, and the efficient utilization of hardware resources. For WebAR applications, the most resource-intensive stage is the direct initialization of the scene, during which the browser simultaneously processes large volumes of geometric data and configures the graphics pipeline [13–16]. It is at this very stage that the highest risk of blocking the main execution thread arises. Consequently, a systematic verification of the extent to which integrated optimization methods can reduce this load becomes particularly relevant as a primary condition for ensuring the stability of modern WebAR applications.

In view of the above, **the aim of this work** is the comprehensive evaluation of optimization performance and the identification of computational load distribution patterns during the initialization of WebAR applications, utilizing deep browser profiling tools.

The scientific novelty of the obtained results lies in the improvement of the 3D content preparation and initialization pipeline for WebAR applications. Unlike existing approaches, a comprehensive combination of a monolithic binary format (GLB) and spatial compression algorithms (Draco) has been applied, which made it possible to alter the load profile on the browser engine. The method for evaluating the performance of web-based AR systems has been further developed: through deep instrumental profiling, it has been quantitatively proven that the proposed optimization not only reduces the overall loading time but also radically eliminates computational bottlenecks, preventing the main execution thread from blocking.

The object of the research is the initialization and 3D content loading process within a client-optimized WebAR application running on a laptop.

Implementing scene loading optimization

The operational efficiency of WebAR applications directly depends on the speed of 3D scene initialization and the minimization of network latency, particularly when utilizing mobile internet connections. The process of loading and processing heavy 3D assets frequently becomes a bottleneck, leading to significant user wait times or the blocking of the browser's main execution thread. To address this issue, a comprehensive approach has been developed and implemented, combining a change in the data transmission format with the application of geometric compression algorithms on the server side. The technical implementation of the proposed optimization encompasses two key levels: structural and algorithmic.

At the structural level, the binary GLB format was selected instead of the standard multi-file glTF format. This solution enables the encapsulation of all model components, including geometry, textures, and material descriptions, into a single monolithic file. The primary advantage of this approach lies in the reduction of HTTP requests to a single one, which radically decreases the total server response wait time, especially under the conditions of unstable mobile networks. Furthermore, GLB format binary data is processed and parsed by the browser significantly faster, as it is transmitted in a format that is maximally close to its native representation in the GPU memory.

The algorithmic level of optimization is aimed at minimizing the payload volume during data transmission over the network. To achieve this, a 3D geometry compression mechanism has been implemented on the backend utilizing the Draco library. This method allows for the significant compression of topological data, such as vertex coordinates, indices, and normals, without any loss in the visual quality of the objects. The decompression process of the received files on the client side is executed using WebAssembly modules, ensuring high-speed operations directly within the web environment.

The applied combination of structural and algorithmic solutions ensures a faster initialization speed for the WebAR application, even when handling large 3D models.

Methodology

The performance evaluation of the optimized WebAR application, developed using Angular 19 and the A-Frame 1.7.1 library, was conducted using Chrome DevTools. A Location AR scene featuring a reference 3D model was selected for testing. The initial size of the test asset in the glTF format was 16.4 MB. To test the optimized version of the WebAR application, this same model was preliminarily converted into the binary GLB format. The file size of the converted GLB asset is 11.8 MB; it is this optimized asset that serves as the target object for subsequent performance profiling.

Given that the maximum load on the application occurs during scene initialization and loading, the research focuses exclusively on this stage. CPU utilization within the context of the web page was selected as the primary metric for performance evaluation. Because the Chrome DevTools instrumentation process inherently introduces additional overhead, each test scenario was executed 10 times. The run with average values was selected for further analysis.

Testing was conducted on a Lenovo Legion S7 laptop. The test device is equipped with an AMD Ryzen 9 6900HX processor (8 cores, 16 threads, 3.3 – 4.9 GHz), 16 GB of RAM, and operates on the Windows 11 operating system.

To ensure the objectivity and reliability of the experimental performance results for the client application featuring augmented reality elements, the testing process was conducted under conditions that closely approximate a real-world operating environment. For this purpose, the deployment of both the server and client components of the software system was carried out on remote servers. This approach eliminates the specific optimizations inherent to a local development environment and accounts for actual network latencies during the transmission of large volumes of data, such as 3D models and high-resolution textures.

Given that the target platform for augmented reality applications is predominantly mobile devices, which frequently operate under unstable wireless connection conditions, the preparation for testing included the mandatory application of an artificial network bandwidth throttling method. A network bandwidth limit of 6 megabytes per second was applied for both incoming and outgoing traffic. This bandwidth metric reflects the typical operating conditions of mobile devices within fourth-generation (4G) communication networks.

Experiment

To minimize the impact of external factors on the measurement results, a device preparation procedure was implemented. Prior to testing, a full system reboot was performed to clear the RAM and force the termination of background processes. The display brightness parameter was fixed at 80%. All third-party applications were deactivated; within the web browser, only the tab containing the WebAR application remained active.

Following the preparatory stage, the data collection procedure was implemented, which entailed executing a series of ten consecutive iterations of recording the performance profile via the Performance tab interface in Chrome DevTools. The experimental scenario encompassed the complete cycle of loading and initializing the geolocation-anchored WebAR scene.

Upon the completion of each measurement, trace files were exported, containing detailed metrics – specifically, the time expended on Scripting, System processes, Rendering and Painting. In addition, the total scenario recording time and JavaScript heap allocation dynamics were recorded.

To select the most representative sample for subsequent analysis, a comparison of the peak JS Heap memory usage values and the durations of key processing stages was conducted. It is important to note that the Total metric also accounts for system idle periods. The obtained quantitative data have been systematized and are presented in Table 1.

Table 1 – **Test result of the optimized version on the Lenovo Legion S7 device**

Test	JS Heap (peak), mb	Scripting, ms	System, ms	Rendering, ms	Painting, ms	Total, ms
1	20,6	1285	967	7	0	5839
2	21,3	1487	945	29	29	5779
3	24	1495	920	29	29	5749
4	24,8	1458	947	31	30	5915
5	31,1	1339	795	31	30	5735
6	24	1432	837	32	30	5773
7	35,2	1366	834	31	31	5737
8	23,9	1343	773	28	27	5752
9	27,1	1294	839	30	29	5819
10	24	1326	783	29	29	5794

Selection of a representative recording

To identify the most representative record among the ten obtained results, t*he normalized deviation method* was employed. This approach objectively determines which test run most closely aligns with the average execution conditions. For each test, the normalized deviation was calculated using the following formula:

$$Score_i = \sum_j \frac{|X_{ij} - Me_j|}{IQR_j}, \quad (1)$$

where X_{ij} is the value of the jth metric for the ith test; Me_j - the median of the corresponding metric; IQR_j is the interquartile range of this metric.

The interquartile range is calculated by the formula:

$$IQR = Q3 - Q1, \quad (2)$$

where Q1 is is the 25th percentile (lower quartile); Q3 is is the 75th percentile (upper quartile).

For a sample size of $n = 10$, the calculation was performed using Tukey's method: the ordered dataset was divided into two equal halves of 5 elements each. In this case, is determined as the median of the first half of the dataset – the 3rd element in the ordered list – while Q3 is determined as the median of the second half, the 8th element. Based on Tukey's method, basic statistical parameters were determined for each metric. The results of identifying the median, Q1, Q3 and calculating the IQR (2) for each column are presented in Table 2. The result of calculating the normalized deviation (1) for each test is given in Table 3.

Table 2 – **Indicator values based on Tukey's method**

Metric	Median	Q1 (3rd element)	Q3 (8th element)	IQR
JS Heap	24	23,9	27,1	3,2
Scripting	1354,5	1326	1458	132
System	838	795	945	150
Rendering	29,5	29	31	2
Painting	29	29	30	1
Total	5776	5749	5819	70

Table 3 – **Normalized deviation of tests on Lenovo Legion S7**

Test	Score	Test	Score
1	43,5990	6	2,8866
2	2,8537	7	6,9209
3	2,2468	8	3,6446
4	5,4965	9	2,2980
5	4,9586	10	1,0897

The obtained values showed that Test No. 10 has the lowest normalized indicator, indicating a minimal deviation from the median characteristics. Accordingly, this record was chosen as representative for further comparison between devices.

Analysis of a representative recording

The representative trace, obtained in the Performance tab of the Chrome DevTools suite, is shown in Fig. 1.

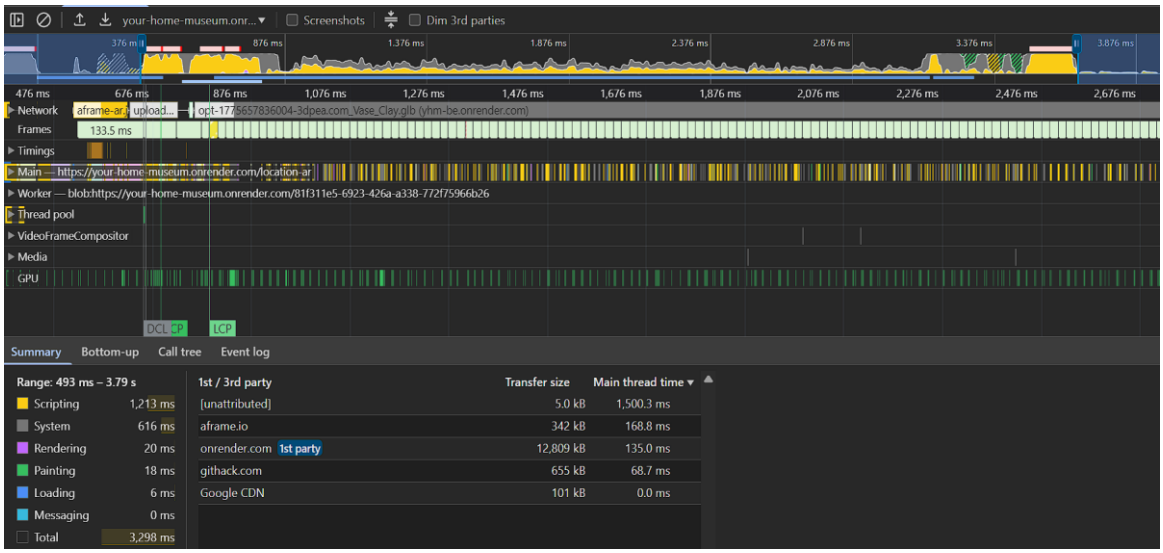

Fig. 1. Representative recording

For the analysis, the application initialization period was considered; therefore, the intervals at the beginning and end of the record were excluded. The main visual metrics include: the CPU activity graph at the top of the figure, the Main thread where function calls are recorded, and the Summary pane.

The Summary pane indicates that the total recording time is 3.29 s, of which 1.21 s is allocated to Scripting, 1.42 s to Idle, 0.61 s to System overhead, 0.02 s to Rendering, and 0.01 s to Painting. These data are presented as a diagram in Fig. 2. Loading and Painting are not considered in the detailed breakdown, as they account for less than 1% of the total time. Thus, the largest share is occupied by system downtime (Idle) with a metric of 43%, followed by the Scripting process (37%).

The CPU graph is conditionally divided into four intervals (Fig. 3), two of which demonstrate a high load on the processor, as evidenced by the fully filled areas. The unhighlighted intervals were excluded, as they do not directly relate to the initialization of the application.

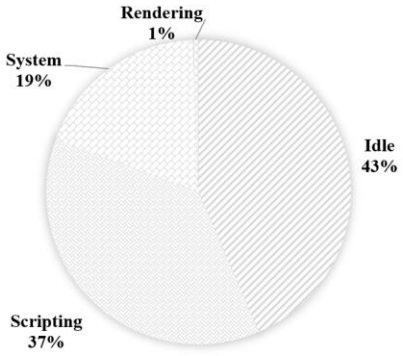

Fig. 2. Process diagram when loading an application

The initial profiling stage, spanning the time interval from 0.491 to 0.924 seconds, is characterized by a high computational load driven by the initialization of the client application components. The majority of this time, specifically 0.354 seconds or over 80% of the interval's duration, is allocated to Scripting.

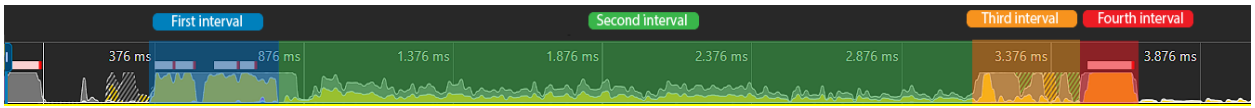

Fig. 3. Dividing a recording into intervals

The longest duration, at 135 milliseconds, was recorded in the Unattributed category. According to the analysis, this corresponds to the execution of native code by the JavaScript engine; however, due to the specific optimization features of the Angular framework, the profiler is unable to identify these calls in greater detail. The second most significant operation is the initialization of the A-Frame spatial library, which lasts 119 milliseconds.

The subsequent stage lasts from 0.924 to 3.25 seconds and is distinguished by moderate computational intensity. The main thread remains in an idle state for 1.302 seconds, accounting for 56% of the interval's time. Such a low processor load is explained by the wait for the network transmission of the 3D model from the server. Computational activity during this phase is limited to non-blocking operations for reading the input stream and executing background processes.

The third phase, spanning from 3.25 to 3.6 seconds, is dedicated to processing the received model data. The load on the main thread increases, with Scripting occupying nearly 39% of the time. The parsing of binary data and its translation into internal scene structures take place during this period. Simultaneously, the garbage collector is activated, optimizing RAM usage down to 8 megabytes. Background thread pools are also initialized, facilitating the parallel decoding of resources and effectively offloading the main thread.

At the final stage, within the interval from 3.6 to 3.78 seconds, the actual rendering process occurs. A peak CPU load is recorded here, with 87% of the time allocated to Scripting. Basic rendering methods, shader program compilation, the uploading of textures to video memory, and the final visualization of the object are identified within this phase.

Thus, the entire loading process is decomposed into four sequential stages: client environment initialization, lasting 0.433 seconds; a network wait period, lasting 2.326 seconds with predominant system idle time; data parsing and structuring, taking 0.350 seconds and utilizing multi-threaded optimization; and the final phase of graphics preparation and rendering, lasting 0.180 seconds with a peak load on the main thread.

Load estimation

To determine the most demanding stage of WebAR application initialization, four intervals were chosen: initialization, loading, parsing, and rendering. The load index for each interval was calculated using the formula:

$$Load = \frac{T_{busy}}{T_{total}} * 100\%, \quad (3)$$

where T_{busy} is total time the main thread performed any tasks (Scripting, Rendering, Painting, System); T_{total} is total interval duration.

The calculation results are given in Table 4.

Table 4 – **Processor performance characteristics at key initialization stages**

№	Stage	Interval, s	T_{total}, ms	T_{busy}, ms	Idle, ms	Load, %
1	Initialization	0.491 – 0.924	434	425	9	97,9%
2	Loading	0.924 – 3.25	2328	1025	1303	44,0%
3	Parsing	3.25 с – 3.6	354	242	112	68,4%
4	Rendering	3.6 – 3.78	188	184	4	97,9%

The highest density of computational operations within the optimized profile is exhibited by the initialization and rendering stages, where peak CPU utilization values are recorded at 97.9% (Fig. 4). Despite the brief overall duration of these phases, amounting to 434 and 188 ms respectively, the processor has virtually no idle time. These stages demonstrate identical utilization metrics but differ in the nature of their operations: during the initialization phase, the processor is primarily engaged in JIT compilation and the execution of library JavaScript code, whereas during rendering, the primary load is attributed to interacting with the graphics engine and frame preparation.

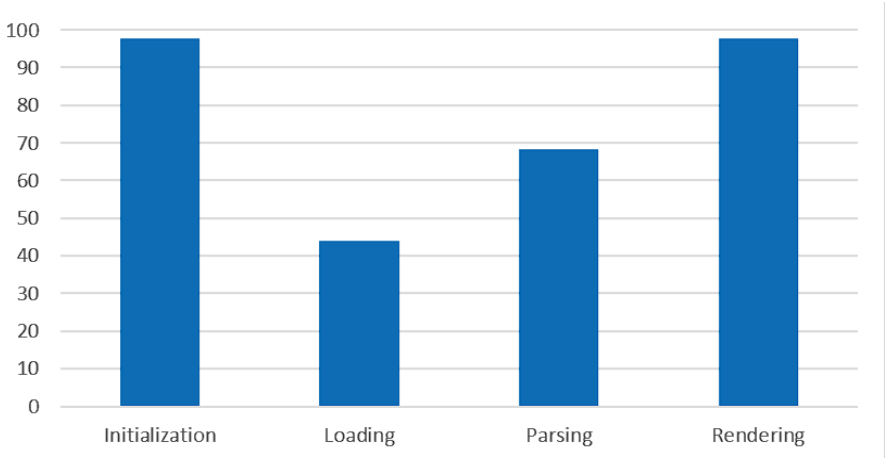

Fig. 4. Percentage load graph of intervals

The third interval, which corresponds to the parsing stage, is characterized by a utilization level of 68.4% over a total duration of 354 ms. The presence of 112 ms of idle time during this stage indicates the successful optimization of the 3D model's binary data deserialization and in-memory object processing. This facilitated the elimination of the severe main thread blocking inherent in the baseline version of the application.

As expected, the Loading stage remains the least resource-intensive, with a utilization metric of 44.0%. This distribution is due to the fact that network bandwidth acts as the limiting factor during this interval; consequently, the processor's main thread spends more than half of the phase's total duration – specifically, 1303 ms – in an idle state, waiting for data.

Thus, in the optimized version, the highest computational load is observed during the fourth interval, Rendering, whereas the Loading stage is the least demanding and exhibits the highest idle time.

Performance benchmarking

To objectively evaluate the effectiveness of the proposed approaches for optimizing WebAR scene loading, it is advisable to conduct a comparative analysis of the obtained experimental data. The performance metrics of the baseline application version were selected as the benchmark for comparison. This enables a quantitative assessment of the performance gain and the reduction in the load on the device's hardware resources.

The primary criteria for comparison are the peak allocated memory and the distribution of processor time among native code execution, system calls, and main thread idle time. The summarized instrumental profiling results for both architectural solutions are presented in Table 5.

The conducted comparative analysis of the performance profiles of the baseline and optimized application versions demonstrates a significant increase in the overall efficiency of WebAR scene initialization. Overall, the implemented architectural changes ensured a more efficient utilization of computational resources and eliminated blocking factors within the 3D asset preparation pipeline. A major achievement is the substantial reduction in the load on the device's RAM. The peak allocated memory decreased by nearly half – by 47.7%. This confirms the high efficiency of the applied data compression methods, which significantly reduced the volume of objects generated in memory during the decompression of the scene's geometry.

Table 5 – **Comparison of performance indicators of the basic and optimized versions of the WebAR application**

Metric	Basic version	Optimized version
JS Heap (peak), mb	45,9	24
Scripting, ms	1462	1213
System, ms	586	616
Idle, ms	2593	1425
Total, ms	4749	3298

The total scene initialization time was reduced by 30.5% – dropping from 4749 to 3298 ms. The transition to the monolithic binary GLB format radically accelerated the network stage of asset retrieval. According to the detailed profiling metrics, the primary driver of this overall acceleration is a 45% reduction in wait time and main thread idle time – decreasing from 2593 to 1425 ms. The obtained results convincingly demonstrate the effectiveness of the architectural solution in eliminating network bottlenecks and optimizing data processing through the synergy of the binary container and geometry compression.

The most significant time reduction was recorded during the Parsing phase (Fig. 5). Its duration decreased by 66.2%, dropping from 1048 to 354 ms. This indicates

the successful elimination of the primary computational bottleneck, achieved through the optimization of the 3D geometry decompression and processing pipeline. The Loading stage, which traditionally occupies the largest absolute share of the total time, accelerated by 20.6%, reducing its metric from 2933 to 2328 ms. The Initialization phase demonstrated a 22.6% acceleration, decreasing from 561 to 434 ms. The smallest relative and absolute performance gain is observed during the Rendering stage, where the execution time decreased by 13.4% – from 217 to 188 ms. Given that the rendering stage was not initially a defining blocking factor in the baseline architecture, such an improvement is a logical and proportional consequence of the overall offloading of the main thread and the optimization of the preceding phases.

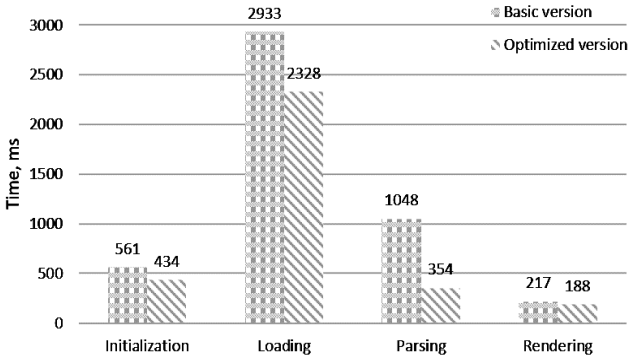

Fig. 5. Comparison of T_{total} at each phase of the base and optimized versions

A 17% reduction in Scripting duration was recorded, dropping from 1462 to 1213 ms, which further attests to the successful optimization of the parsing stage. Concurrently, a slight 5.1% increase in System overhead, reaching 616 ms, is a logical and acceptable trade-off caused by the overhead associated with managing background threads.

The primary consequence of the implemented architectural changes was the targeted redistribution of the computational load among the data preparation phases (Fig. 6).

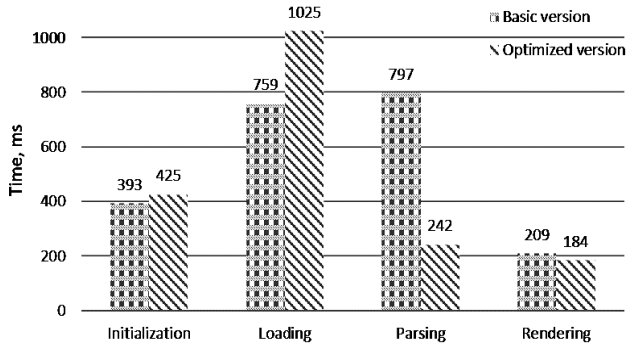

Fig. 6. Comparison of T_{busy} at each phase of the base and optimized versions

The most significant reduction in the volume of active computations was recorded during the Parsing phase, where T_{busy} decreased by 69.6%, dropping sharply from 797 to 242 ms. Such a radical offloading of the main thread confirms the successful elimination of the critical bottleneck, achieved through the application of compression and the optimization of scene geometry processing algorithms.

In contrast, during the Loading stage, a logical increase in active processor time by 35.0% is observed, specifically from 759 to 1025 ms.

This increase is not indicative of performance degradation but rather serves as a direct result of implementing the optimization strategy. Instead of passively waiting for the network download, the system preemptively executes preparatory computational tasks, effectively utilizing processor time.

During the Initialization phase, a minor increase in active load of 8.1% was recorded, specifically from 393 to 425 ms. This can be logically attributed to the initial overhead associated with preparing the background thread infrastructure and initializing the decompression modules. Conversely, the Rendering stage demonstrated a 12.0% decrease in processor time consumption, dropping from 209 to 184 ms, which is a direct consequence of operating on an already optimized and pre-processed 3D model structure.

As a result, the recorded shift in the profile of T_{busy} illustrates a qualitative transition from a sequential and blocking execution of tasks to a balanced and parallel data processing pipeline.

Conclusions

In this work, a comprehensive performance evaluation of WebAR application scene loading optimization was conducted. The results of instrumental profiling confirmed the high efficiency of the applied approach, which is based on transitioning to the binary GLB format and utilizing Draco geometric compression. The implemented architectural changes successfully eliminated critical bottlenecks in the 3D asset preparation pipeline.

A significant reduction in the load on hardware resources was recorded: the peak allocated JS Heap memory decreased by 47.7%. The overall scene initialization time was reduced by 30.5%, dropping from 4749 to 3298 ms.

The primary driver of this acceleration was a 45% reduction in main thread idle time and a radical 66.2% decrease in the duration of the parsing phase.

It has been proven that the integration of a monolithic format and client-side decompression leads to a natural and more balanced redistribution of computational resources, allowing for the efficient utilization of processor time during the loading stage. This guarantees improved stability and performance for modern WebAR applications.

Conflicts of interest

The author declare that he has no conflicts of interest in relation to the current study, including financial, personal, authorship, or any other, that could affect the study, as well as the results reported in this paper.

Use of artificial intelligence

The author confirm that they did not use artificial intelligence technologies when creating the current work.

REFERENCES

1. Nadeem M., Lal M., Cen J., Sharsheer M. AR4FSM: Mobile Augmented Reality Application in Engineering Education for Finite-State Machine Understanding // Education Sciences. 2022. Vol. 12. No. 8. P. 555. DOI: https://doi.org/10.3390/educsci12080555
2. Parekh P., Patel S., Patel N., Shah M. Systematic Review and Meta-Analysis of Augmented Reality in Medicine, Retail, and Games // Visual Comput. Ind. Biomed. Art. 2020. Vol. 3. No. 1. Art. 21. DOI: https://doi.org/10.1186/s42492-020-00057-7
3. Butt A., Ahmad H., Muzaffar A., Ali F., Shafique N. WOW, the Make-Up AR App is Impressive: A Comparative Study Between China and South Korea // Journal of Services Marketing. 2022. Vol. 36. No. 1. P. 73–88. DOI: https://doi.org/10.1108/JSM-12-2020-0508
4. Yin C. Z. Y. et al. Mobile Augmented Reality Heritage Applications: Meeting the Needs of Heritage Tourists // Sustainability. 2021. Vol. 13. No. 5. Article 2523. DOI: https://doi.org/10.3390/su13052523
5. Divya Udayan J. et al. Augmented Reality in Brand Building and Marketing – Valves Industry // 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). 2020. P. 1–6. DOI: https://doi.org/10.1109/ic-ETITE47903.2020.425
6. Ghattas M. M. Performance Evaluation of Websites Using Machine Learning // EIMJ. 2020. Vol. 51. P. 36–41.URL: https://www.researchgate.net/publication/345975296_Performance_Evaluation_of_Websites_Using_Machine_Learning
7. Matvieiev M. I. Analysis of Optimization Methods for Augmented Reality Web Applications // Control, Navigation and Communication Systems. 2024. No. 4(78). P. 106–108. DOI: https://doi.org/10.26906/SUNZ.2024.4.106
8. Alsulami M. H. et al. Development of an Approach to Evaluate Website Effectiveness // Sustainability. 2021. Vol. 13. No. 23. Article 13304. DOI: https://doi.org/10.3390/su132313304
9. Boutsi A.-M., Ioannidis C., Verykokou S. Multi-Resolution 3D Rendering for High-Performance Web AR // Sensors. 2023. Vol. 23. Article 6885. DOI: https://doi.org/10.3390/s23156885
10. Maclntyre B., Smith T.F. Thoughts on the Future of WebXR and the Immersive Web // Proceedings of the 2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct. 16–20 October 2018. P. 338–342. DOI: https://doi.org/10.1109/ISMAR-Adjunct.2018.00099
11. gLTF Runtime 3D Asset Delivery. Available online: https://www.khronos.org/gltf/
12. Li L., Qiao X., Lu Q., Ren P., Lin R. Rendering Optimization for Mobile Web 3D Based on Animation Data Separation and On-Demand Loading // IEEE Access. 2020. Vol. 8. P. 88474-88486. DOI: https://doi.org/10.1109/ACCESS.2020.2993613
13. Ghattas M., Mora A. M., Odeh S. A Novel Approach for Evaluating Web Page Performance Based on Machine Learning Algorithms and Optimization Algorithms // AI. 2025. Vol. 6. No. 2. P. 19. DOI: https://doi.org/10.3390/ai6020019
14. Kumar A., Arora A. A Filter-Wrapper based Feature Selection for Optimized Website Quality Prediction // Proceedings 2019 Amity International Conference on Artificial Intelligence (AICAI). 2019. P. 284–291. DOI: https://doi.org/10.1109/ai-cai.2019.8701362
15. Allison R. et al. A Comprehensive Framework to Evaluate Websites: Literature Review and Development of GoodWeb // JMIR Formative Research. 2019. Vol. 3. P. e14372. DOI: https://doi.org/10.2196/14372
16. Morales-Vargas A., Pedraza-Jimenez R., Codina L. Website quality in digital media: literature review on general evaluation methods and indicators and reliability attributes // Revista Latina de Comunicacion Social. 2022. Vol. 80. P. 39–63. DOI: https://doi.org/10.4185/RLCS-2022-1515

Received (Надійшла) 11.02.2026
Accepted for publication (Прийнята до друку) 29.04.2026
Publication date (Дата публікації) 22.05.2026

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Матвєєв Микита Іванович – аспірант кафедри комп'ютерної інженерії та програмування, Національний технічний університет «Харківський політехнічний інститут», Харків, Україна;
Mykyta Matvieiev – PhD student, Department of Computer Engineering and Programming, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: Mykyta.Matvieiev@cit.khpi.edu.ua; ORCID Author ID: https://orcid.org/0009-0000-8773-6640;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=60141774400.

Оцінювання продуктивності оптимізації завантаження сцени у вебAR-додатку

М. І. Матвєєв

Анотація. Актуальність. WebAR стрімко розвивається, проте стикається з проблемою високого обчислювального навантаження на пристрої. Синхронне завантаження важких 3D-моделей під час ініціалізації сцени часто призводить до затримок і блокує основний потік виконання браузера. Застосування комплексних методів оптимізації (таких як перехід на формат GLB і використання стиснення геометрії Draco) є критично важливим для забезпечення стабільності, однак їх впровадження потребує об'єктивної кількісної оцінки. **Мета дослідження** – оцінити продуктивність оптимізації завантаження сцени WebAR-додатку на етапі ініціалізації. **Об'єкт дослідження** – процес ініціалізації та завантаження 3D-контенту у клієнтському WebAR-додатку, що працює на ноутбуці. **Предмет дослідження** – показники продуктивності методів оптимізації WebAR-додатку. **Висновки.** За результатами інструментального профілювання підтверджено високу ефективність переходу на формат GLB та використання стиснення Draco. Зафіксовано зменшення пікового обсягу виділеної пам'яті JS Heap на 47,7%. Загальний час ініціалізації сцени скоротився на 30,5% – з 4749 до 3298 мс. Крім того, час простою основного потоку суттєво зменшився на 45%, а тривалість фази парсингу – на 66,2%, що дозволило усунути критичні вузькі місця у конвеєрі підготовки 3D-ресурсів.

Ключові слова: веб, доповнена реальність, WebAR, оцінювання продуктивності, оптимізація, Draco, Angular.