RENDERING PERFORMANCE ANALYSIS OF SMARTPHONE LIDAR 3D MODELS IN AN OPEN-SOURCE WEBGIS

Ketut Tomy Suhari*[1], Yustina Cheline J Owa[1], Septa Erik Prabawa[2]
[1]Program Studi Teknik Geodesi, Institut Teknologi Nasional Malang, Malang, Indonesia
[2]Program Studi Teknik Geomatika, Universitas Dr. Soetomo, Surabaya, Indonesia
Penulis korespondesi: Ketut Tomy Suhari (ksuhari@lecturer.itn.ac.id)

This study evaluates the rendering performance of smartphone LiDAR-derived 3D models in a web-based CesiumJS environment using a mural dataset. Three formats were compared: raw OBJ, monolithic glTF, and 3D Tiles. The geometry comprises 801,949 vertices and 1,372,979 triangles. Asset-level measurements show that transferable payload decreases from 77.82 MB (OBJ) to 67.15 MB (glTF) and 13.67 MB (3D Tiles), demonstrating that tiled delivery offers superior storage and transfer efficiency. The 3D Tiles package includes 89 GLB tiles with a dominant structural concentration at traversal depth 3. Tile-level statistics reveal payload variability between 5.54 KB and 481.88 KB. To evaluate runtime performance, the study proposes a regression-oriented benchmark framework using loading time, frame rate, tile-load intensity, browser type, and device class. Analysis suggests that loaded-tile intensity influences frame-rate variation more significantly than loading time. Findings indicate that 3D Tiles is the most suitable representation for CesiumJS when prioritizing progressive refinement and scalable access. Furthermore, the results confirm that low-cost smartphone LiDAR combined with open-source web technologies is effective for the online visualization and dissemination of geospatial digital twins.

Keywords — LiDAR, Cesiumjs, Rendering Performance, GLTF, 3D Tiles.

I. INTRODUCTION

Smartphone LiDAR has expanded access to dense three-dimensional data capture for small-scale documentation, making 3D acquisition more affordable and practical than conventional high-cost survey systems. This development is particularly important for objects such as murals, façades, and other localized structures, where rapid documentation and online dissemination are often more relevant than large-area mapping [1], [2].

At the same time, the publication of 3D data on the web introduces a different set of challenges. Previous studies in web-based geospatial visualization have shown that effective dissemination depends not only on geometric quality, but also on data structure, browser-side processing, and delivery strategy [3], [4]. In this context, browser-based 3D visualization has increasingly relied on tiled streaming and efficient representation schemes to improve transfer efficiency and interactive performance [5], [6].

Open-source web environments have further expanded the practical use of online 3D visualization by enabling heterogeneous 3D content to be integrated, explored, and disseminated without reliance on proprietary visualization platforms [7], [8]. Recent comparative studies also indicate that display performance may vary substantially depending on the web-rendering framework used, especially for extensive 3D geospatial datasets [9]. Beyond simple file delivery, web-based 3D systems are also shaped by broader data-exchange and scene-graph concepts that affect how heterogeneous assets can be structured and interpreted in runtime environments [10].

Against this background, the present study focuses on the rendering performance of smartphone LiDAR-derived 3D models in a CesiumJS-based web environment. Using a mural dataset as the experimental case, three alternative representations derived from the same geometry are evaluated: raw OBJ mesh, monolithic glTF, and 3D Tiles. The study combines asset-level measurement, tile-structure analysis, and exploratory runtime observation in order to examine how representation strategy influences web delivery and rendering behavior. In doing so, the paper positions itself as a comparative rendering-performance investigation rather than as a system-development article.

Accordingly, this research addresses four main questions: (1) how much transferable payload reduction is achieved by each representation format; (2) how the tiled payload is distributed across traversal depths in the 3D Tiles hierarchy; (3) how loading time, frame rate, and tile-load behavior vary in a CesiumJS runtime environment; and (4) to what extent exploratory regression-oriented analysis can reveal the relationship among representation structure, browser execution, and rendering performance.

II. MATERIALS AND METHODS

A. Dataset and Representation Preparation

The research utilizes a mural situated on the exterior of a curved building facade as the experimental case for evaluating web-based rendering performance. This object

was specifically selected because its composition integrates elongated planar segments, a curved terminal section, and a continuous distribution of mural panels. Such spatial characteristics are analytically relevant for assessing how alternative 3D representations influence browser-side processing and visualization behavior.

The physical properties of the documented object are established in Figures 1(a) and 1(b). Figure 1(a) illustrates the primary facade and the ordered mural sequence, while Figure 1(b) captures the curvature of the terminal structure. These field photographs provide the necessary reference for interpreting the fidelity of the web-rendered model, confirming that the dataset encompasses both planar and non-planar geometries, which directly impact representation efficiency and rendering complexity

Figure 1. (a) field photo of the mural main facade; (b) field photo of the curved facade

The mural dataset was formulated into four distinct representations. The primary source archive consists of a LAS point cloud, capturing the original spatial distribution as discrete points; however, this format was excluded from the comparative rendering benchmark. The second representation is a raw OBJ mesh, which serves as the baseline surface model. The third is a monolithic glTF, designed to store the reconstructed mesh and associated textures within a compact, web-optimized asset structure. Finally, the fourth representation is a 3D Tiles package, comprised of multiple GLB payloads structured for hierarchical streaming. These conditions were selected to evaluate alternative web-delivery strategies grounded in established concepts of efficient browser-based transmission and progressive 3D content dissemination [11].

Furthermore, the evaluation of glTF and 3D Tiles within a unified experimental context is motivated by the need for interoperability and structured geospatial visualization. This focus aligns with contemporary research emphasizing interoperable web-based 3D frameworks that facilitate the integration of heterogeneous geospatial assets in a common delivery environment [12]. In this broader informatics domain, emergent Web3D application models indicate that the structured interaction and exchange of 3D content increasingly rely on extensible web-oriented APIs and interoperable scene logic [13].

The operational workflow for the web-based visualization environment is delineated in Figure 2. In this pipeline, the initial smartphone LiDAR acquisition is processed into three distinct representations: raw OBJ mesh, monolithic glTF, and 3D Tiles. These diverse assets are subsequently integrated and deployed within a development framework leveraging HTML5, JavaScript, and CesiumJS, which facilitates the multi-platform dissemination of the mural model across desktop and mobile execution contexts.

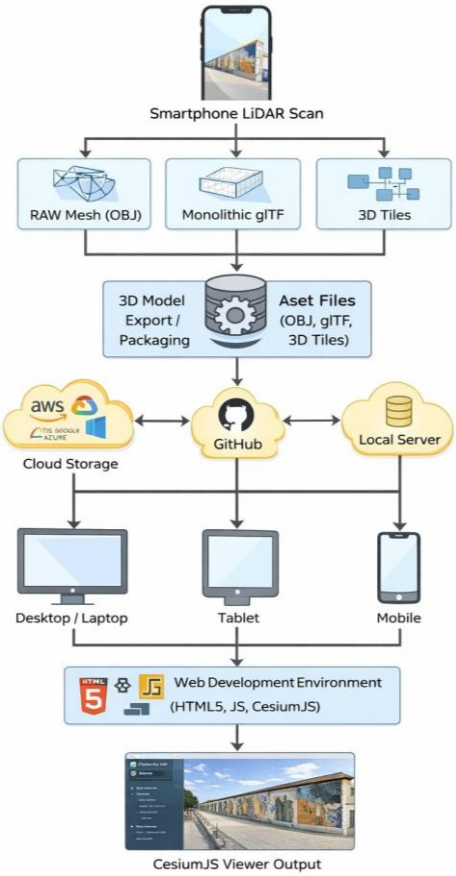

Figure 2. CesiumJS workflow and viewer setup

Figure 3 illustrates the successful integration and visualization of the smartphone LiDAR-derived mural model within a CesiumJS-based web environment. The rendered output preserves the fundamental geometric

integrity of the documented structure, accurately reproducing the curved facade and the systematic mural sequence established by the field photographs.

These results confirm that the synthesis of low-cost acquisition workflows with open-source web technologies provides a technically viable and visually coherent three-dimensional representation for online dissemination. Consequently, the comparative rendering performance investigation is grounded in a functional deployment context rather than a simulated visualization scenario.

Figure 3. CesiumJS-based web visualization of the mural model.

To ensure comparability, all evaluated surface representations were derived from the same reconstructed mural geometry. The raw OBJ baseline was obtained by exporting the geometry contained in the monolithic glTF model into Wavefront OBJ format. This procedure preserves the same surface content while removing the compact packaging and progressive streaming advantages associated with glTF and 3D Tiles. Consequently, the measured differences reflect representation strategy rather than differences in geometric reconstruction.

The dataset contains 801,949 vertices and 1,372,979 triangles. The percentage of size reduction between representations is calculated as:

$$R_s = (1 - \frac{S_f}{S_r}) \times 100\% \quad (1)$$

Where R_s denotes the percentage of size reduction, S_f is the file size of the reference representation, and S_r is the file size of the evaluated representation. Equation (1) is applied to compare raw OBJ with glTF, raw OBJ with 3D Tiles, and glTF with 3D Tiles.

The structural properties of the three representations are summarized in Table 1. These measured differences are analytically important because they directly affect network transfer size, browser parsing overhead, memory consumption, and rendering scalability in CesiumJS [3], [5], [8].

Table 1. Structural summary of the three dataset representations

Representation	Primary unit	Size (MB)	Structural note
Raw OBJ mesh	Single file	77.82	801,949 vertices; 1,372,979 triangles
Monolithic glTF	Single file	67.1	Embedded buffers; one textured mesh
3D Tiles	89 GLB tiles + 1 subtree	13.7	Hierarchical streaming package

Applying Equation (1), the measured size reduction from raw OBJ to monolithic glTF is 13.72%, while the reduction from raw OBJ to 3D Tiles reaches 82.44%, and from monolithic glTF to 3D Tiles is 79.65%, as presented in Table 3. These values demonstrate that the most substantial performance gain is achieved not merely by adopting a web-oriented asset format, but by reorganizing the content into a tiled streaming structure, which in turn influences download latency, parsing time, and the responsiveness of browser-based rendering [5], [6], [9].

Figure 4 further highlights that the choice of representation strategy has a direct influence on transferable *payload* and the potential startup cost within browser environments. Although all three formats preserve the identical mural geometry, their storage behavior differs substantially. The raw OBJ representation is the least efficient as it retains the highest transfer burden, while glTF provides moderate optimization through compact serialization. The 3D Tiles representation yields superior efficiency because the dataset is subdivided into hierarchical units that support selective loading and progressive refinement. This indicates that efficient web rendering depends not only on mesh complexity, but also on data partitioning and *streaming* architecture.

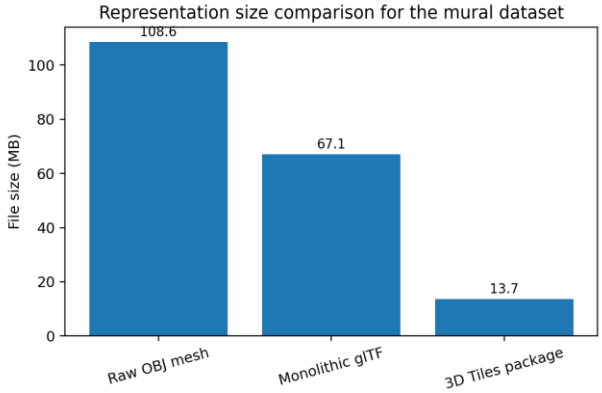

Figure 4. Representation size comparison for the mural dataset

B. CesiumJS Benchmark Scenario

Runtime performance is evaluated in a CesiumJS environment using a controlled and repeatable navigation sequence. From a standards perspective, the two main delivery formats examined in this study are directly grounded in the 3D Tiles and glTF specifications, which define the structural basis for hierarchical streaming and compact 3D asset representation in web-based environments [14], [15]. Each representation is tested under the same camera presets and interaction sequence, consisting of initial load, zoom-in, orbit, pan, and return to overview.

To reduce stochastic variation caused by network jitter, browser background processes, and caching effects, each benchmark configuration should be repeated at least 30 times. Repeated measurements provide a more stable estimate of average performance and support subsequent statistical modeling. The main runtime indicators considered in this study are loading time, average frame rate, request count, and tile traversal structure.

Initial loading time is defined as the time interval between the start of the asset request and the first visually usable frame:

$$LT = t_{usable} - t_{start} \quad (2)$$

Where LT is the initial loading time, t_{start} is the request start time, and t_{usable} is the timestamp at which the first visually usable frame becomes available.

Average frame rate is computed over a fixed observation window during active navigation:

$$FPS_{avg} = \left(\frac{1}{n}\right) \Sigma_{j=1}^{n} FPS_j \quad (3)$$

Where FPS_{avg} is the average frame rate, FPS_j is the instantaneous frame rate at sample j, and n is the number of frame-rate samples collected during the observation interval.

For the tiled representation, traversal structure is quantified by the proportion of tiles observed at each hierarchy depth:

$$P_k = \frac{N_k}{N_{tot}} \quad (4)$$

where P_k is the proportion of tiles at traversal depth k, N_k is the number of tiles observed at depth k, and N_{tot} is the total number of observed tiles.

To complement tile count analysis, payload concentration at each depth can also be measured as:

$$Q_k = \frac{B_k}{B_{tot}} \quad (5)$$

where Q_k is the payload proportion at depth k, B_k is the cumulative payload size at that depth, and B_{tot} is the total payload size across all tiles. This metric is useful because rendering efficiency in 3D Tiles is not determined solely by tile count but also by how much payload is concentrated at different refinement levels [5], [9].

C. Device and Browser Matrix

To ensure that the runtime analysis remains relevant to browser-based informatics applications, the benchmark was organized across browser and device classes rather than being limited to a single execution condition. In the current dataset, runtime observations were available from Chrome on desktop and Safari on mobile. These two conditions were sufficient to provide an initial comparison between desktop and mobile execution contexts, which is important because rendering performance in CesiumJS depends not only on data representation, but also on browser engine behavior and device capability [4], [5], [8].

Accordingly, browser type and device class were retained as explanatory variables in the analytical framework. Although the broader benchmark matrix originally envisioned for this study included additional browsers, the present paper reports only the configurations that were actually executed and analyzed.

Table 2. Device conditions used in the present study for browser execution

Class	Browser	Representative metric	Purpose
Desktop	Chrome	LT, FPS_{avg}, loaded tiles	Main reference configuration
Mobile	Safari/iOS	LT, FPS_{avg}, loaded tiles	Apple browser behavior

D. Regression Framework

A central contribution of this study is the use of regression analysis to separate structural effects from runtime effects in browser-based rendering. Rather than relying only on descriptive comparison, the proposed framework allows the contribution of representation type, request behavior, and device/browser condition to be estimated quantitatively. This is especially relevant when different representations may trade off transfer size against request overhead and refinement complexity [4], [5], [8], [9].

The first model explains loading time as a function of payload size, request behavior, tile count, device class, and browser category:

$$LT_i = \beta_0 + \beta_1 S_i + \beta_2 Req_i + \beta_3 Tile_i + \beta_4 Mob_i + \Sigma_{m=1}^{M-1} \gamma_m B_{im} + \varepsilon_i \quad (6)$$

Where LT_i is the loading time for benchmark run i, S_i is the transferred payload size, Req_i is the total number of network requests, $Tile_i$ is the number of loaded tiles, Mob_i is a binary variable indicating mobile device class, B_{im} represents browser dummy variables, β_0 is the intercept, β_1, ..., β_4 are regression coefficients, γ_m are browser-effect coefficients, and ε_i is the residual error term.

The second model explains frame rate as a function of visible geometric load, request behavior, device class, and representation format:

$$FPS_i = \alpha_0 + \alpha_1 Tri_i + \alpha_2 Req_i + \alpha_3 Mob_i + \Sigma_{r=1}^{R-1} \delta_r F_{ir} + u_i \quad (7)$$

Where FPS_i is the average frame rate for benchmark run i, Tri_i is the number of visible triangles in the rendered view, Req_i is the total number of network requests, Mob_i indicates mobile device class, F_{ir} represents format dummy variables for raw mesh, glTF, and 3D Tiles, α_0 is the intercept, α_1, ..., α_3 are regression coefficients, δ_r are format-effect coefficients, and u_i is the residual error term.

If repeated observations are collected from the same device, the analysis may be extended into a mixed-effects formulation by using device identity as a random intercept. Such an extension is beneficial when device-specific variability is expected to influence rendering outcomes beyond the fixed effects explicitly modeled above.

III. RESULTS AND DISCUSSION

A. Asset-Level Rendering Indicators

The measured dataset properties reveal a significant divergence among the evaluated representations even before runtime execution is initiated. The raw OBJ baseline requires a 77.82 MB payload, whereas the monolithic glTF and 3D Tiles package occupy 67.15 MB and 13.67 MB, respectively. Given that all three formats preserve the identical mural geometry, the variation in storage size is directly attributable to the representation strategy and delivery structure rather than geometric simplification, as summarized in Table 3.

Applying Equation (1), the measured size reduction from raw OBJ to monolithic glTF is 13.72%, while the reduction from raw OBJ to 3D Tiles reaches 82.44%, and from monolithic glTF to 3D Tiles is 79.65%, as presented in Table 3. These values demonstrate that the most substantial performance gain is achieved not merely by adopting a web-oriented asset format, but by reorganizing the content into a tiled streaming structure.

Table 3 further highlights that the choice of representation strategy has a direct influence on transferable payload and the potential startup cost within browser environments. While the conversion from OBJ to glTF provides moderate optimization through compact serialization, the transition to 3D Tiles yields superior efficiency because the dataset is subdivided into hierarchical units that support selective loading and progressive refinement.

Table 3. Measured representation-size reduction

Comparison	Reference Size (MB)	Evaluated Size (MB)	Reduction (%)
Raw OBJ → glTF	77.82	67.15	13.72
Raw OBJ → 3D Tiles	77.82	13.67	82.44
glTF → 3D Tiles	67.15	13.67	79.65

From a rendering perspective, this result is scientifically important because browser-based deployment depends strongly on the volume of data that must be transferred and parsed before the scene becomes visually usable. The raw OBJ representation is the least efficient as it retains the highest transfer burden while lacking a web-optimized delivery structure. Although the monolithic glTF improves compactness, it still concentrates the majority of transfer and parsing costs at the beginning of the session. By contrast, the 3D Tiles representation drastically reduces the transferable package and shifts the delivery logic toward progressive refinement, making it the most structurally suitable format for scalable CesiumJS visualization.

B. Relation Between Field Object and Web Representation

The field photographs in Figures 1 and 2 establish the physical characteristics that the rendered model is expected to preserve. The object consists of a long mural-bearing wall and a curved terminal facade under a large roof structure. This composition makes the case suitable for web-based rendering analysis because it combines repeated mural panels, curved geometry, and spatial continuity along a facade.

In the web environment, the rendered model should reproduce the same mural order, facade curvature, and wall extent observed in the field photographs. Therefore, the quality of the web representation is not assessed only in terms of visual appearance, but also in terms of structural fidelity and delivery efficiency. In practical terms, a representation that preserves the curved wall and mural layout while minimizing transfer cost is more suitable for online dissemination.

This relationship between field reference and rendered representation is especially important for CesiumJS deployment. If the object seen in the viewer corresponds clearly to the physical object documented in the field, the browser experiment becomes meaningful not only as a graphics test, but also as a documentation and dissemination case.

C. Tile Traversal Structure

The 3D Tiles package includes 89 GLB payloads, a single subtree file, and a *tileset.json* descriptor. Traversal depth analysis reveals a significant structural concentration at deeper hierarchy levels; specifically, depth 3 contains 41 tiles and depth 4 contains 32 tiles. Collectively, these two levels represent 82.02% of the entire tile set (73 of 89 tiles), as illustrated in Figure 5.

Regarding cumulative payload, depth 3 contributes 7.917 MB (57.51% of the total GLB payload), while depth 4 contributes 3.504 MB (25.45%). In contrast, upper levels remain lightweight, with depth 0 containing only one tile and depth 1 containing four tiles. This confirms that the model is organized for progressive refinement rather than as a shallow hierarchy. As shown in Figure 5, the structural design ensures that lightweight upper levels provide an initial scene overview, while the majority of geometric detail is deferred to deeper refinement levels.

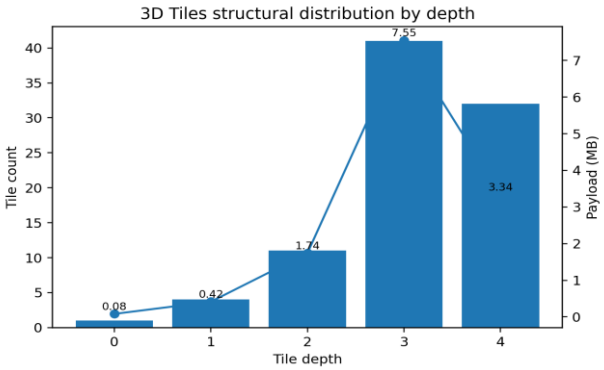

Figure 5. Count and payload distribution across traversal depths

This structure is consistent with a streaming strategy in which coarse scene context is loaded first and detailed content is requested only when camera proximity requires higher resolution. For the mural object, such a hierarchy is well suited to CesiumJS because the browser can first display the overall facade and subsequently refine mural details during closer interaction.

The size of individual GLB payloads is highly variable. Across the 89 payload tiles, the minimum size is 5.54 KB, the maximum size is 481.88 KB, the median is 122.52 KB, and the mean is 151.05 KB, as summarized in Table 4. These descriptive statistics indicate a strongly

heterogeneous payload structure rather than a uniform subdivision scheme.

Table 4. Descriptive statistics of individual GLB tile payloads

No	Statistic	Value
1	Number of GLB tiles	89
2	Minimum payload size	5.54 KB
3	Maximum payload size	481.88 KB
4	Median payload size	122.52 KB
5	Mean payload size	151.05 KB

The distribution of tile payload sizes is further visualized in Figure 6, which highlights that most tiles are concentrated in the lower-to-middle payload range, while a smaller subset occupies substantially larger sizes. This right-skewed pattern suggests that only specific segments of the model contain relatively heavy geometric or texture content.

Additional insight is provided by Figure 7, where tile payload variability is examined across traversal depths. The figure demonstrates that depth alone does not fully explain tile cost, as tiles located at similar hierarchy levels may still differ considerably in payload size. Such variability likely reflects local geometric complexity, texture density, or the specific subdivision strategy applied during tiling generation.

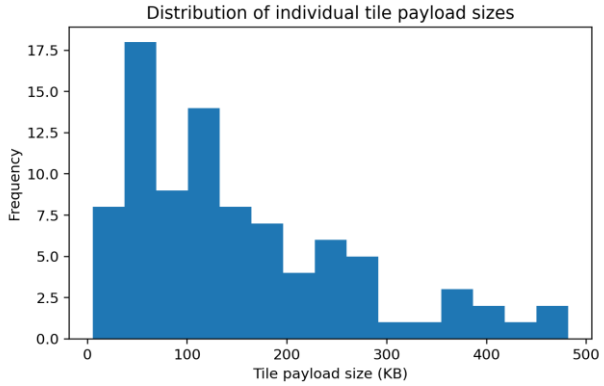

Figure 6. Distribution of individual tile payload sizes

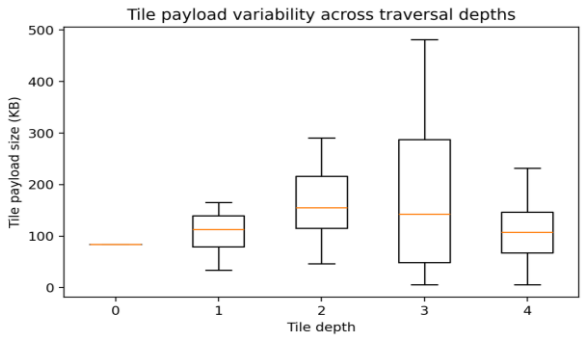

Figure 7. Variability of tile payload size across traversal depths

This finding is important because browser rendering cost depends not only on the number of requested tiles, but also on the volume of data carried by those requests. For the mural case, runtime behavior in CesiumJS should therefore be interpreted using both tile-count and payload metrics. A depth-rich hierarchy can remain efficient if upper-level tiles are lightweight and most detail is delayed, although performance may still fluctuate when certain deeper tiles are disproportionately heavy.

D. Implications for CesiumJS Rendering Performance

Although the present section is based on measured asset-level evidence rather than complete runtime logs, the rendering implications are already clear. The raw OBJ representation is the least favorable for web use because it combines the largest payload with a non-streaming structure. The monolithic glTF performs better in terms of file size and graphics compatibility, yet it still requires a substantial upfront transfer before meaningful interaction begins. The 3D Tiles representation is the most structurally appropriate for CesiumJS because it distributes the dataset into small, view-dependent payloads that can be requested incrementally.

In this case, the tiled representation does not merely reduce size. It also changes the rendering logic from front-loaded delivery to hierarchical streaming. For the mural facade, this is advantageous because users typically do not need the highest detail everywhere at the initial view. Instead, they first need a usable overview and then gradual refinement while approaching the wall or inspecting individual mural panels.

This distinction is central to the scientific contribution of the paper. The observed differences are not only visual or descriptive; they directly reflect alternative computational strategies for browser-based 3D delivery. Therefore, representation should be understood as a rendering-performance factor rather than as a neutral storage choice.

E. Exploratory Runtime Regression Analysis

To extend the asset-level interpretation into browser-execution behavior, the recorded runtime observations were further examined using scatter plots with simple linear regression lines. This step was designed as an exploratory analysis rather than a final inferential model, because the current benchmark dataset is still limited in size and several runtime indicators, particularly request-related variables, require further refinement in the logging procedure. Even so, the available observations are sufficient to reveal preliminary tendencies in the relationships among loading time, average frame rate, and loaded-tile count.

The exploratory analysis was carried out using three scatter plots. The first plot relates loading time to average frame rate, the second relates loaded-tile count to average frame rate, and the third relates loaded-tile count to loading time. In each case, the linear regression line serves as a compact summary of the dominant trend in the current observations. This approach is useful because it allows the runtime behavior of glTF and 3D Tiles to be examined not only descriptively, but also in terms of directional association among variables.

F. Loading Time versus Average Frame Rate

Figure 8 presents the scatter plot of loading time against average frame rate. After removing unrealistic frame-rate outliers, 20 observations were retained for this analysis. The recorded loading times range from 1.712 s to 47.304 s, while the average frame rate ranges from 42.640 FPS to 70.212 FPS. The fitted regression line shows a weak negative tendency, with a coefficient of determination of $R^2 = 0.045$. This value indicates that loading time explains only a very small proportion of the variation in average frame rate.

This result suggests that startup cost and rendering smoothness should not be interpreted as the same

performance dimension. A representation may require longer initialization while still maintaining acceptable interactive smoothness, whereas another may load more rapidly but perform less consistently during camera movement. In practical terms, the plot shows that loading time alone is not sufficient to explain runtime frame-rate behavior in the current CesiumJS benchmark. Additional factors, including device class, browser engine, and representation structure, must also be considered.

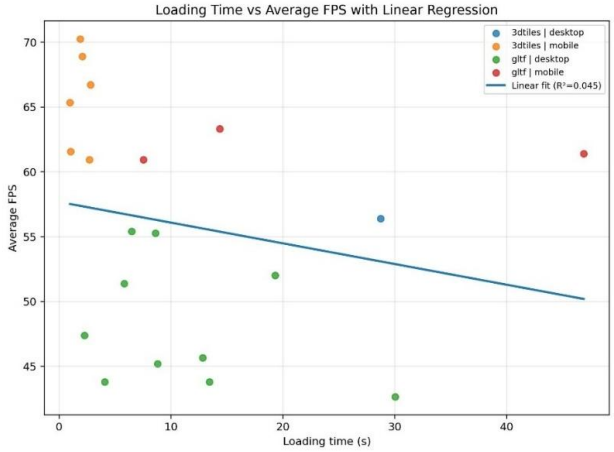

Figure 8. Loading Time vs Average FPS with Linear Regression

G. *Loaded Tiles versus Average Frame Rate*

Figure 9 shows the relationship between loaded-tile count and average frame rate. A total of 20 observations were included after filtering unrealistic frame-rate values. The loaded-tile count ranges from 22 to 54, while the average frame rate ranges from 42.640 FPS to 70.212 FPS. Compared with the first plot, this figure shows a much clearer tendency. The fitted regression line indicates a negative relationship between loaded tiles and average frame rate, with $R^2 = 0.757$.

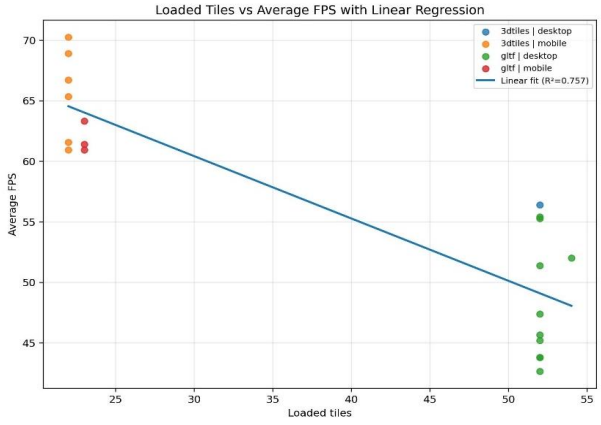

Figure 9. Loaded Tiles vs Average FPS with Linear Regression

This relatively high coefficient of determination suggests that, within the present dataset, loaded-tile count is more strongly associated with frame-rate variation than loading time alone. However, this result should still be interpreted cautiously. The observations are visibly clustered by format and device class, particularly between mobile and desktop conditions, which means that the apparent strength of the relationship may partly reflect broader execution differences rather than a single direct causal mechanism. Even so, the figure is informative because it indicates that tile-load intensity is an important variable in understanding CesiumJS runtime performance.

H. *Loaded Tiles versus Loading Time*

Figure 10 presents the scatter plot of loaded tiles versus loading time using 21 observations. The loaded-tile counts vary from 22 to 54, whereas loading time ranges from 1.712 s to 47.304 s. In this case, the fitted regression line shows only a very weak positive tendency, with $R^2 = 0.037$. This result indicates that the number of loaded tiles alone explains very little of the observed variation in startup duration.

This finding is important because it shows that loading time cannot be reduced to a simple tile-count effect. In a tiled streaming system, startup cost also depends on payload size, request scheduling, refinement logic, browser execution, and hardware capability. Therefore, even though the tiled structure is central to the rendering strategy, the number of loaded tiles does not by itself provide a complete explanation of startup performance. The mural dataset illustrates this clearly, since its hierarchy contains variable payload sizes across depths and not all tiles impose equal transfer or parsing cost.

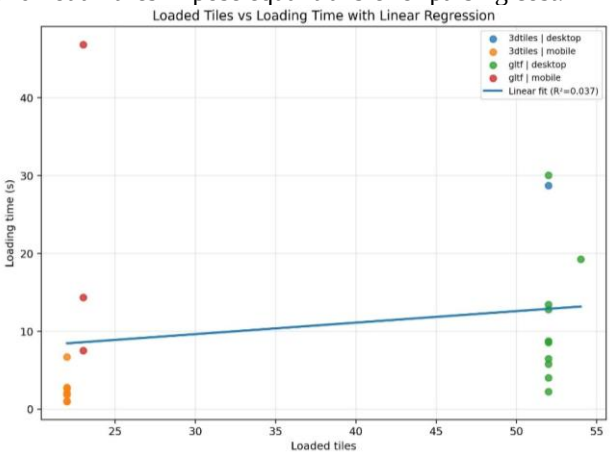

Figure 10. Loaded Tiles vs Loading Time with Linear Regression

I. *General Interpretation of the Regression Oriented Results*

Taken together, Figures 8 to 10 show that runtime behavior in the current CesiumJS benchmark is influenced by multiple interacting variables. The weak regression results for loading time versus average frame rate and for loaded tiles versus loading time suggest that these relationships are not strongly linear in the current dataset. By contrast, the stronger negative tendency observed between loaded tiles and average frame rate indicates that tile-load intensity may be an important runtime factor, although its effect is likely entangled with browser and device differences.

From the perspective of this study, these results reinforce the argument that rendering performance in web-based CesiumJS environments should be understood as a representation and delivery problem rather than merely as a storage problem. The runtime outcome depends not only on file size reduction, but also on how content is subdivided, requested, refined, and processed across heterogeneous execution environments. The exploratory regression analysis therefore does not replace the asset-level findings, but complements them by showing that representation structure has measurable runtime implications.

At the present stage, the regression results should be interpreted as preliminary rather than definitive. A stronger inferential analysis will require a larger number of repeated runs per representation, cleaner request and payload logging, and stricter separation between format-specific runtime indicators. Nevertheless, the current exploratory plots are already useful for demonstrating that the mural dataset exhibits measurable runtime variation across formats and device/browser contexts, which supports the central framing of this paper as a comparative rendering-performance study.

IV. CONCLUSIONS AND SUGGESTIONS

The present study reformulates the mural dataset into a performance-oriented investigation focused on CesiumJS-based web delivery strategies. By utilizing the same underlying geometry, three alternative representations—raw OBJ mesh, monolithic glTF, and 3D Tiles—were evaluated to determine their relative efficiency. Asset-level measurements reveal a significant reduction in transferable payload, which decreases from 77.82 MB for the raw OBJ baseline to 67.15 MB for the monolithic glTF and 13.67 MB for the 3D Tiles package. The tiled archive comprises 89 GLB payload tiles, a single subtree file, and a tileset.json descriptor, with a dominant structural concentration observed at traversal depth 3. These findings confirm that 3D Tiles offers the most delivery-efficient representation for CesiumJS, particularly when prioritizing progressive refinement, scalable access, and initial scene usability.

Furthermore, the study indicates that rendering performance cannot be reduced to storage size alone. Analysis of the tile hierarchy, payload distribution, and exploratory runtime observations suggests that rendering behavior is shaped by complex interactions among representation format, browser environment, and device class. The regression-oriented analysis further suggests that loaded-tile intensity exhibits a more significant relationship with frame-rate variation than loading time alone, demonstrating that rendering efficiency depends not only on transfer volume but also on the strategy for content subdivision and request scheduling during visualization. Consequently, the paper contributes a practical benchmark framework for future runtime modeling incorporating loading time, frame rate, request behavior, browser type, and device class.

More broadly, the results demonstrate that a low-cost smartphone LiDAR workflow, combined with open-source web technologies, is capable of producing effective three-dimensional visualizations for online dissemination. Even in the absence of proprietary platforms, the integration of smartphone-based acquisition with glTF or 3D Tiles packaging provides a technically viable and scientifically meaningful approach for geospatial publication. This finding indicates that accessible hardware and open web standards can support practical 3D documentation workflows suitable for research-oriented deployment. Therefore, this research positions itself as a scientific informatics contribution emphasizing rendering performance, delivery efficiency, and the potential of open-source solutions for web-based 3D visualization.

Future research should aim to expand the sample size of runtime observations and diversify interaction scenarios to enhance the statistical robustness and generalizability of the proposed regression framework

REFERENCES

[1] F. Di Stefano, S. Chiappini, A. Gorreja, M. Balestra, and R. Pierdicca, "Mobile 3D scan LiDAR: A literature review," Geomatics, Natural Hazards and Risk, vol. 12, no. 1, pp. 2387–2429, 2021.

[2] D. Costantino, G. Vozza, M. Pepe, and V. S. Alfio, "Smartphone LiDAR technologies for surveying and reality modelling in urban scenarios: Evaluation methods, performance and challenges," Appl. Syst. Innov., vol. 5, no. 4, p. 63, 2022.

[3] M. Kulawiak, M. Kulawiak, and Z. Lubniewski, "Integration, processing and dissemination of LiDAR data in a 3D Web-GIS," ISPRS Int. J. Geo-Inf., vol. 8, no. 3, p. 144, 2019.

[4] M. Buyukdemircioglu and S. Kocaman, "Reconstruction and efficient visualization of heterogeneous 3D city models," Remote Sens., vol. 12, no. 13, p. 2128, 2020.

[5] W. Zhan, Y. Chen, and J. Chen, "3D Tiles-based high-efficiency visualization method for complex BIM models on the Web," ISPRS Int. J. Geo-Inf., vol. 10, no. 7, p. 476, 2021.

[6] A. Schilling, J. Bolling, and C. Nagel, "Using glTF for streaming CityGML 3D city models," in Proc. 21st Int. Conf. Web3D Technol., 2016, pp. 109–116.

[7] M. La Guardia and M. Koeva, "Towards digital twinning on the web: Heterogeneous 3D data fusion based on open-source structure," Remote Sens., vol. 15, no. 3, p. 721, 2023.

[8] M. Auer and A. Zipf, "3D WebGIS: From visualization to analysis. An efficient browser-based 3D line-of-sight analysis," ISPRS Int. J. Geo-Inf., vol. 7, no. 7, p. 279, 2018.

[9] T. Seto, Y. Shiwaku, T. Miyauchi, D. Yoshida, and Y. Nishimura, "Assessment of display performance and comparative evaluation of web map libraries for extensive 3D geospatial data," arXiv preprint arXiv:2602.23660, 2026.

[10] J. S. Dhanjan and A. Steed, "Revisiting the scene-graph-as-bus concept: Inter-networking heterogeneous applications using glTF fragments," in Proc. IEEE Conf. Virtual Reality and 3D User Interfaces Workshops (VRW), 2021, pp. 342–346.

[11] W. Lemoine and M. Wijnants, "Progressive network streaming of textured meshes in the binary glTF 2.0 format," in Proc. 28th Int. ACM Conf. 3D Web Technol., 2023, pp. 1–11.

[12] A. Havele, A. Plesch, and M. McCann, "Conceptualizing interoperable 3D geospatial data visualization with X3D and OGC 3D Tiles," in Proc. 29th Int. ACM Conf. 3D Web Technol., 2024, pp. 1–6.

[13] N. Narra, A. Marisetty, N. Polys, and B. Sandbrook, "A generalized Web3D API for metaverse bookmarks," in Proc. 30th Int. ACM Conf. 3D Web Technol., 2025, pp. 1–8.

[14] Open Geospatial Consortium, "3D Tiles Standard," OGC Std. 18-053r2, 2023.

[15] Khronos Group, "glTF 2.0 Specification," Oct. 11, 2021.