Visualization of Urban Digital Twins on the web with attribute-driven adaptive tiling

Usta, Ziya; Akın, Alper Tunga; Ohori, Ken Arroyo; Stoter, Jantien

DOI
10.1016/j.envsoft.2026.106863

Publication date
2026

Document Version
Final published version

Published in
Environmental Modelling and Software

Citation (APA)
Usta, Z., Akın, A. T., Ohori, K. A., & Stoter, J. (2026). Visualization of Urban Digital Twins on the web with attribute-driven adaptive tiling. *Environmental Modelling and Software*, *197*, Article 106863. https://doi.org/10.1016/j.envsoft.2026.106863

Environmental Modelling and Software 197 (2026) 106863

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft

Visualization of Urban Digital Twins on the web with attribute-driven adaptive tiling

Ziya Usta [a,b,*], Alper Tunga Akın [a,c], Ken Arroyo Ohori [a], Jantien Stoter [a]

[a] 3D Geoinformation, Urban Data Science, Delft University of Technology, Delft, 2628 BL, The Netherlands
[b] Department of Geomatics Engineering, Faculty of Engineering, Artvin Çoruh University, Artvin, 08100, Türkiye
[c] Department of Geomatics Engineering, Faculty of Engineering, Karadeniz Technical University, Trabzon, 61080, Türkiye

ARTICLE INFO

Keywords:
Urban Digital Twins
3D city models
CityJSON
3D visualization
Web
Geospatial
WebGPU

ABSTRACT

Despite growing use of 3D city models (3DCMs) and urban digital twins (UDTs), web tools for their processing and visualization remain scarce. We present an interoperable, high-performance web application composed of a 3D tiler and a WebGPU viewer that enables scalable conversion, streaming, and rendering of urban datasets in compliance with open standards. The proposed system allows users to explore large-scale 3DCMs interactively without local installations. A showcase visualizing quality-validation results for a 3DCM demonstrates practical value. Experiments confirm that 3D Tiles 1.1 standard enables scalable data management and richer interaction, whereas WebGPU offers up to 7x better rendering performance on modern hardware. By presenting this solution and usage example, we aim to foster development of next-generation web-based 3D geospatial, digital-twin, and metaverse solutions.

Software and data availability

- Developers: Ziya Usta
- Contact:
- Date first available: June 2, 2025
- Software required: Anaconda, NodeJS
- Program language: Python, JavaScript, HTML
- Source code at: https://osf.io/rhfxn/?view_only=a2fca00c4b144a0f90cb8ea9488d94df
- Test data at: https://osf.io/y43hr/?view_only=0fe0c56a8c8f4a46b3423e4d4455c5c2
- Documentation: Detailed documentation is in the Wiki section of the shared repository.

1. Introduction

For sustainable planning and design, cities are increasingly using three-dimensional city models (3DCMs) or Urban Digital Twins (UDTs) as the primary source of data. As a result, an increasing number of cities are producing 3DCMs and sharing them as open data, such as Berlin, Helsinki, and New York City (Labetski et al., 2023). While many cities share their 3D models through open data repositories, they rarely have a user-friendly way to explore these data. Leveraging Web 3D technologies offers a promising approach to visualize 3DCMs (Gaillard et al., 2020). Various software tools are employed to generate 3DCMs and develop applications based on them. However, having a well-performing viewer stands out as crucial among these tools (Arroyo Ohori, 2020). With rapid developments in web technologies such as HTML5 and WebGL, WebGIS applications have started to replace desktop applications for 3D GIS (Usta, 2021). One of these applications is visualizing 3DCMs via browsers. With the increasing popularity of web-based applications, the utilization of 3D city models would benefit from the availability of web applications that can handle 3D city models (Vitalis et al., 2020). Access to 3DCMs from internet browsers enables the use of 3DCMs by a large mass of professionals who are not experts in spatial information but who can benefit from 3DCMs in their own studies (Prandi et al., 2015). Beyond their technical capacity, digital twin technologies rely heavily on clarity and precision in how information is modeled and conveyed. This communicative accuracy ensures that the digital representation faithfully reflects the physical system, thus improving decision-making and collaboration (Khan and Ahmad, 2025)

However, 3DCMs or UDTs can be huge in size, while browsers are essentially document viewers with restricted memory and only limited ability to render large-scale UDTs. Existing solutions to address these issues, such as OGC 3D Tiles 1.0 and WebGL, suffer from their own specific architectural constraints. In 3D Tiles 1.0, models are batched into .b3dm files using a batch table for identification. However, this

* Corresponding author at: 3D Geoinformation, Urban Data Science, Delft University of Technology, Delft, 2628 BL, The Netherlands.
E-mail address: z.usta@tudelft.nl (Z. Usta).

https://doi.org/10.1016/j.envsoft.2026.106863
Received 7 June 2025; Received in revised form 1 December 2025; Accepted 5 January 2026
Available online 6 January 2026

table supports only feature-level IDs, meaning complex semantic hierarchies are lost and individual sub-components, such as roofs or walls, cannot be distinguished. Furthermore, 1.0 requires explicit definitions for bounding volumes and metadata, resulting in bloated tileset.json files that hinder client performance. Similarly, WebGL, based on the legacy OpenGL ES standard, restricts access to the low-level features of modern GPUs.

This research bridges the gap between these legacy standards and their successors, OGC 3D Tiles 1.1 and WebGPU. We present a novel tiler and viewer designed to convert large-scale CityJSON files for high-performance rendering. A key contribution of our solution is its "attribute-aware" methodology, which proves that leveraging data context significantly optimizes both the tiling and visualization processes. The remainder of the article is organized as follows: Sections 2 and 3 review relevant technologies and related work; Sections 4 and 5 present our methodology and use case scenarios; and Sections 6 and 7 offer a discussion and conclusion.

2. Background

2.1. Streaming standards for 3D geospatial data

To enable the efficient dissemination of large amounts of 3D data, the data needs to be decomposed into smaller chunks. For this purpose there are two OGC community standards, OGC 3D Tiles and I3S. OGC 3D Tiles, originally developed by Cesium, is an open standard for streaming and rendering massive, heterogeneous 3D geospatial datasets. OGC 3D Tiles builds on the glTF 3D format and was adopted as an OGC community standard in 2019 (Open Geospatial Consortium, 2019). I3S, introduced by ESRI, is an open standard for streaming and storing large-scale 3D geospatial data. I3S was approved as an OGC community standard in 2017 (Belayneh, 2022). Both standards are based on the same fundamental concepts and organize 3D geospatial data (e.g., point clouds, meshes, buildings) into a hierarchical spatial structure of tiles, using a tree-like organization. Each tile contains a subset of the data at varying levels of detail (LOD). Thus, only the tiles needed for the current view (based on the user's location, zoom level, and viewport) are streamed, reducing bandwidth usage and enabling smooth rendering of massive datasets progressively, even on low-end devices.

I3S is more prevalent in ESRI-centric environments, while 3D Tiles benefits from a broader open-source adoption in diverse applications. 3D tiles integrates with many open-source platforms such as Cesium, NVIDIA Omniverse, Unreal Engine, Unity, Godot, and QGIS.

2.2. Attribute-driven adaptive tiling of 3D geospatial data

The concept of attribute-driven tiling, particularly in 3D datasets, is a useful method for effectively managing and analyzing complex environments that reflect non-geometric attributes alongside geospatial factors. This multifaceted approach aims to enhance how spatial data is utilized, extracting more detailed insights from 3D objects by incorporating the contextual non-geometric information that the data has. Such partitions, or tilings, are not just a function of spatial coordinates but also consider environmental, contextual, and functional attributes. The integration of these elements can create more informative models that accurately reflect the complexities of the environments they represent. This is particularly evident in frameworks that focus on dynamic visualization, as discussed by Hairuddin et al. (2019) and Beil et al. (2022) Furthermore, the hierarchical structure of 3D Tiles supports the tiling of both geometric and semantic layers, enabling detailed urban analyses, such as modeling energy consumption, as demonstrated by Mao et al. (2020).

2.3. 3D graphics on the web

Before WebGL's introduction in 2011, web-based 3D graphics relied on proprietary plugins like Adobe Flash and Microsoft Silverlight, which posed performance, compatibility, and security challenges due to the lack of a standardized, hardware-accelerated API.

To overcome aforementioned issues, WebGL, a graphics API based on OpenGL that enables hardware-accelerated 3D rendering without plug-ins was introduced in 2011. Using WebGL, 3D functionality can be realized directly on the browser utilizing the client's graphic hardware without any plug-in installation (Chaturvedi et al., 2015). Numerous studies have successfully visualized 3D city models (3DCMs) and digital twins directly in web browsers, eliminating the need for additional software or plug-ins. Gesquière and Manin (2012) utilized WebGL to render CityGML data, Jaillot et al. (2020) incorporated time-dynamic data into 3DCM visualizations, and Gaillard et al. (2020) proposed a method for visualizing 3DCMs at multiple levels of detail.

WebGL is based on OpenGL, which itself was originally developed in 1992, and has started to show its age (Usta, 2024). Modern GPUs, which prioritize parallelism and low-level control, are slightly different from GPUs 30 years ago. As GPU hardware evolves, WebGL's reliance on OpenGL ES becomes a liability. Unlike WebGPU, which supports general-purpose GPU (GPGPU) computing through dedicated compute shaders, WebGL lacks native compute shader support and relies on cumbersome workarounds using vertex and fragment shaders, limiting its efficiency for advanced rendering techniques like ray tracing and GPGPU tasks critical for modern applications.

As a response to the limitations of WebGL, a new modern graphics API called WebGPU has been emerged. WebGPU is built on next-generation GPU APIs like Vulkan, Metal and Direct3D 12 which are designed for modern GPU hardware and offers lower overhead, better parallelism, and advanced features. WebGPU's low-level API reduces driver overhead, allowing more efficient GPU utilization. WebGPU supports multi-threaded rendering and compute tasks, leveraging modern multi-core CPUs and GPUs. WebGPU supports hardware-accelerated ray tracing (where available), enabling photorealistic lighting, shadows, and reflections. WebGPU had been in development since 2011, but it gained significance in 2023 when Chrome started to support it. While WebGPU has made significant strides since its initial release in Chrome in 2023, it is not yet as universally reliable or mature as WebGL due to incomplete browser support, evolving ecosystem, and debugging challenges. The WebGPU specification, maintained by the W3C GPU for the Web Community Group, is stable for its core features, such as rendering, compute shaders, and resource management, but developers should monitor for minor breaking changes in browser updates.

In summary, the background outlined above demonstrates that efficient web-based visualization of large-scale 3D geospatial data depends on standardized streaming formats and modern GPU-based rendering. Although these components have been advanced individually, they are seldom integrated within a single, fully web-based framework. The following section reviews recent studies that have addressed these aspects to contextualize the contribution of this work.

3. Related work

In recent years, a growing body of research and software development has focused on the efficient visualization and delivery of large-scale 3D geospatial data, particularly through the use of OGC 3D Tiles and related web technologies. While several academic studies have explored various aspects of 3D Tiles integration—from API-based delivery methods to real-time simulations and BIM visualization—existing tools often exhibit significant limitations. Most solutions either lack full web-based functionality, do not support emerging standards such as OGC 3D Tiles 1.1 or WebGPU, or are incompatible with lightweight formats like CityJSON. The following section reviews both academic efforts

Table 1
Summary of the existing software components.

Software	Viewer	Generator	3DTiles1.0	3DTiles1.1	Platform	WebGPU	CityJSON
NINJA	✓	×	✓	×	Web	×	✓
3DBAG Viewer	✓	×	✓	×	Web	×	✓
iTowns	✓	×	✓	×	Web	×	×
3DTilesRendererJS	✓	×	✓	×	Web	×	×
Azul	✓	×	×	×	Desktop	×	✓
Py3DTilers	×	✓	✓	×	Desktop	×	×
obj2tiles	×	✓	✓	×	Desktop	×	×
citygml-to-3dtiles	×	✓	✓	×	Desktop	×	×
cesium.js	✓	×	✓	✓	Web	×	×
giro3d	✓	×	✓	×	Desktop	×	×
3dtiles	×	✓	✓	×	Desktop	×	×
mago 3DTiler	×	✓	✓	×	Desktop	×	×
Usta (2024b)	✓	✓	✓	×	Web	×	×
Proposed solution	✓	✓	✓	✓	Web	✓	✓

and software tools, highlighting their capabilities and shortcomings in relation to modern 3D geoinformation needs.

In their work, La Guardia et al. (2024) proposed an open-source solution for developing UDTs accessible via web using 3D Tiles. In their work, Santhanavanich et al. (2022) focuses on interoperable 3D data delivery on the web by using OGC 3D GeoVolumes API and OGC 3D Tiles. They generated standardized URL paths to access 3D Tiles datasets in a server by implementing the OGC 3D GeoVolumes API. Würstle et al. (2022) has tested OGC 3D Tiles and Esri Indexed 3D Scene Layer in a game engine environment and showed that 3D Tiles can be used directly in game engines. Mao et al. (2020) proposed a dynamic online 3D visualization framework based on 3D Tiles for displaying real-time energy simulation results. Zhan et al. (2021) proposed a 3D Tiles-based visualization method on the web for complex BIM models. Ilgar et al. (2024) developed a Web-based 3D cadastre prototype for visualization of condominiums. Colin et al. (2024) used 3D Tiles to visualize 3D geospatial urban entities. obben (2024) Integrated 3D functionality into Tailormap using 3D Tiles. Coors and Padsala (2024) visualized UDTs enriched with energy-related data on the web using 3D Tiles and GeoVolumes standards for public participation. Chen et al. (2024) developed a platform for coastal resistance planning using 3D Tiles. Yu et al. (2025) used 3D Tiles for streaming 3D spatial data on the web.

13 different software components have been examined regarding their type tiler or viewer, whether they support the OGC 3D Tiles specifications, CityJSON, or WebGPU. The results are listed in Table 1. Most of the software in the table either views or generates a 3D Tileset, but cannot do both. None of them supports WebGPU. Only Cesium.js (CesiumJS, 2025), a product of the company behind the 3D Tiles specification, fully supports OGC 3D Tiles 1.1. For viewers, only NINJA (Vitalis et al., 2020), 3DBAG Viewer (3DBag Viewer, 2025), and Azul (Arroyo Ohori, 2020) support CityJSON. Azul and NINJA do not implement OGC 3D Tiles and view 3DCMs all at once. This is not a problem for Azul, which is developed for macOS by Arroyo Ohori (2020); however, NINJA is a web-based product developed by TU Delft 3D Geoinformation Group and has memory limitations imposed by browsers. For example, in 32-bit Chrome, memory cannot be allocated more than 2 GB, whereas in 64-bit Chrome, there is a 4 GB limit. Hence, if the CityJSON file exceeds 4 GB in size, Chrome will crash, and the 3DCM will not be able to be visualized by NINJA. iTowns (iTowns, 2025) and 3DTilesRendererJS (3DTilesRendererJS, 2025) are two JavaScript libraries for viewing OGC 3D Tilesets based on well-known open-source 3D render library THREE.js. They do not support OGC 3D Tiles 1.1 and WebGPU. 3DBAG Viewer is developed by TU Delft 3D Geoinformation Group and their spin-off 3DGI. It can view 11 million buildings on the web and uses 3DTilesRendererJS under the hood. For the generation of 3D tilesets from 3DCMs, web-based solutions are respectively rare. There is desktop software, Py3DTilers (Marnat et al. 2022). For 3D tiling, there are no open-source web-based solutions available. ‘obj23dtiles’ (obj23tiles, 2025) is an open source Node.js module designed to convert 3D data to 3D Tiles. obj23dtiles has a significant limitation: it does not construct a hierarchy; thus, it does not implement a tiling method. It merely converts the entire 3DCM in the OBJ format to a single B3DM file. Similarly, the open-source Node.js component citygml-to-3dtiles (citygml-to-3dtiles, 2025) has the same limitation; it only converts CityGML data to the b3dm format without a tiling method. The most advanced open-source component is Py3DTilers, which converts CityGML data to 3D Tiles by tiling the data and constructing a hierarchical tileset. However, Py3DTilers is not web-based and must be installed as standalone software, and it is not based on OGC 3D Tiles 1.1 When academic studies and existing software components are examined, it is evident that many are not web-based, do not support the CityJSON format, do not support OGC 3D Tiles 1.1, and do not support WebGPU.

4. Methodology

4.1. Implementation of the 3D tiler according to OGC 3D Tiles 1.1

The implementation of OGC 3D Tiles 1.1 comprises two primary components: a 3D Tiler and a 3D Viewer. The overall system architecture is illustrated in Fig. 1.

Client-side operations facilitate data ingestion, where users provide datasets in CityJSON format. The client is also responsible for visualization, utilizing 3DTilesRendererJS with support for both WebGPU and WebGL backends. This dual-backend capability enables a comparative performance analysis across diverse hardware configurations.

Server-side operations focus on the parsing of geometries and semantics to execute tileset generation. This process incorporates user-defined attribute prioritization, the mechanics of which are detailed in Section 5. Upon completion, the generated dataset is streamed to the client in OGC 3D Tiles 1.1 format for rendering.

4.1.1. Pre-processing

The pre-processing step begins with parsing the CityJSON file. For this purpose, the open-source Python library ‘cjio’ has been used. Using cjio’s classes and methods, city objects are traversed, and their geometries and attributes are extracted. At this stage, two specific situations need to be handled. First, the CityJSON file does not include vertex normals, they must be computed and added to the data. This is because some viewers calculate vertex normals on the fly when they are missing, while others do not. Since there is no standard behavior among viewers in this regard, it is essential to compute vertex normals at this stage if they are not already present in the CityJSON file. Second, both WebGL and WebGPU support only points, lines, and triangles as geometric primitives. Hence, to display geometries on the browsers, polygonal surfaces in CityJSON need to be triangulated. In this regard, an ear-clipping algorithm has been implemented to convert polygonal surfaces into triangles.

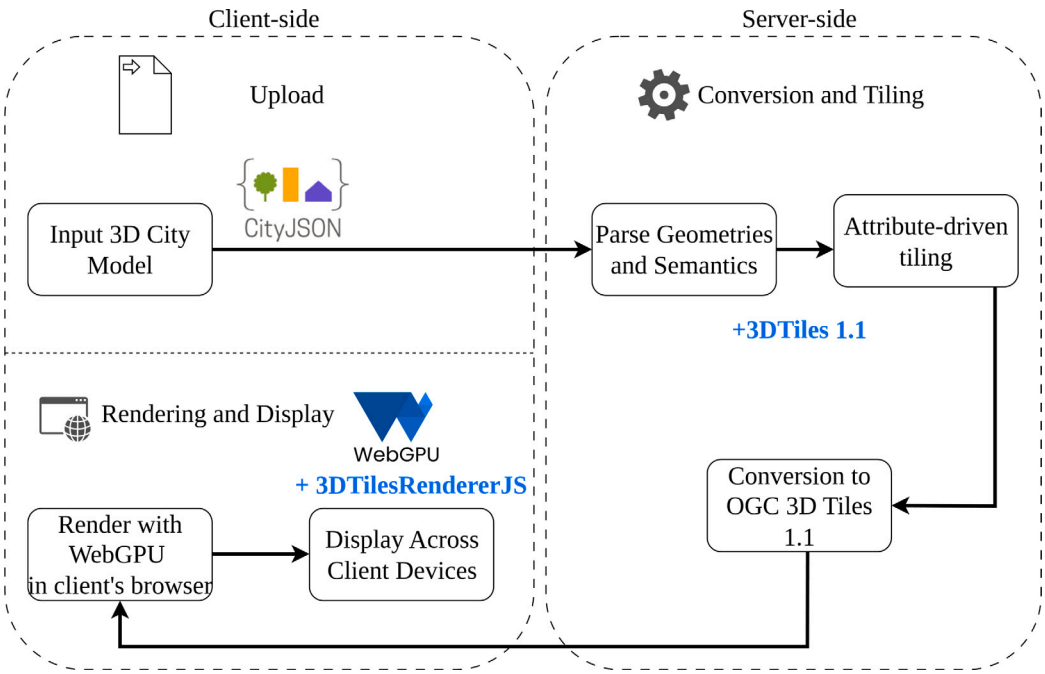

Fig. 1. General system overview.

4.1.2. Spatial partitioning of 3DCM

For subdivision of 3DCM into tiles, a density-based QuadTree data structure has been implemented. Any data structure can be used along with OGC 3D Tiles as long as it is a tree. However a density based QuadTree has some advantages. First, if it is density-based, each tile is subdivided into child tiles only if the tile size is exceeded. Thus, the data sizes across the tiles are approximately equal, which results in the same fetching and loading times while displaying the tiles.

Another advantage is that using regular subdivision schemes such as QuadTree or Octree make it possible to implement 'implicit tiling', which is a new feature of OGC 3D Tiles 1.1. Unlike explicit tiling in 3D Tiles 1.0, where each tile's bounding volume, content URI, and metadata are explicitly specified in a tileset.json file, implicit tiling uses a predefined, regular pattern to implicitly determine the structure and availability of tiles. By eliminating the need to explicitly define each tile's bounding volume and URI in tileset.json, implicit tiling significantly reduces the size of tileset metadata, especially for large datasets. A tileset with millions of tiles no longer requires a massive JSON file, as tile locations are computed algorithmically.

4.1.3. Implementation of OGC 3D Tiles 1.1

In 3D Tiles, hierarchical information and metadata about tiles are encoded to a JSON file called "tileset.json". Thus, the tileset.json file is consumed by the client at runtime, and the scene graph is derived for the spatial queries and optimizations in the visualization pipeline (Usta et al., 2024). Hierarchical information is extracted from the QuadTree and encoded into the tileset.json file. For 3D model format, OGC 3D Tiles 1.1 uses glTF. Hence, for each tile, the data of the tile is converted to glTF 2.0 format and encoded as binary .glb files. To avoid losing semantic information and object hierarchy in the CityJSON file, two extensions of glTF specification 'EXT_mesh_features' and 'EXT_structural_metadata', have been implemented.

The EXT_mesh_features extension allows developers to assign unique identifiers (feature IDs) to specific subcomponents of a glTF asset's geometry, which can be a building, a part of a building, etc. These IDs enable the association of attributes with individual "features" without requiring separate meshes or nodes, preserving rendering efficiency.

The EXT_structural_metadata extension provides a framework for storing structured attributes associated with glTF assets in a compact, binary format. It defines schemas which are templates for attribute classes and storage mechanisms such as property tables, attributes, or textures to link attributes to features identified by EXT_mesh_features. This enables complex, typed attributes such as strings, floats, vectors, or arrays of these types to be associated with geometry at various levels of granularity, from entire nodes to individual vertices. Thus this extension enables queries or visualizations based on attributes, such as coloring the buildings based on the height attribute.

4.2. Implementation of the 3D viewer based on WebGPU and OGC 3D Tiles 1.1

Designing a web-based viewer using OGC 3D Tiles 1.1 and WebGPU involves creating a high-performance, interactive 3D visualization platform capable of streaming and rendering large-scale geospatial datasets in a browser. This viewer will leverage the advanced features of 3D Tiles 1.1 and WebGPU's modern GPU capabilities.

4.2.1. 3D Tiles 1.1 support

For 3D Tiles 1.1 Support, the viewer fetches and parses the tileset.json file to extract the tile structure and properties of the tiles. To support streaming of the tiles, a tile loader that fetches relevant tiles based on the information extracted from the tileset.json file has been implemented.

4.2.2. Streaming of tiles

To enable progressive visualization of the tiles based on the position of the camera, a tile loader has been developed. The tile loader utilizes error-based metrics such as geometric error (GE) and screen space error (SSE) to decide which tiles should be loaded and unloaded. GE is a per-tile property defined in the tileset.json file. SE is a calculated metric that measures the visual error of a tile's geometry in screen pixels when projected onto the viewer's screen. It can be calculated as in the following formula (1). In the formula (1) "FOV" stands for vertical field of view angle of the camera measured in radians.

$$SSE = \frac{GE \cdot screenHeight}{2 \cdot distance \cdot tan(FOV/2)} \quad (1)$$

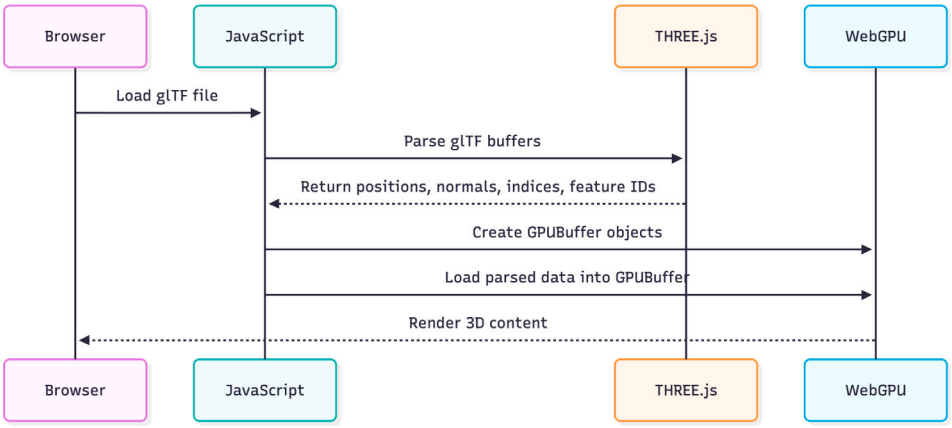

Fig. 2. Sequence diagram of the rendering pipeline.

Each tile in a 3D Tiles tileset has a predefined GE, set during tileset creation. This value is static and reflects the level of simplification of the tile. The SSE is dynamically computed by the renderer based on the GE, camera distance, and viewport properties. The renderer compares the SSE to a maximum SSE threshold to decide whether to render the current tile, load child tiles or skip rendering. If the SSE is below the threshold, the tile's detail is sufficient and it is rendered. If the SSE exceeds the threshold, more detailed child tiles which have lower GE are fetched from the server to reduce the visual error. If the SSE is too high and no children are available, the tile must be culled from the scene to keep the number of tiles manageable for rendering. Thus, the rendering process is to be prevented from clashes by keeping browser memory sufficient.

To optimize streaming performance, a LRU (Least Recently Used) based caching mechanism has been implemented to optimize repeated tile access while zooming in and out. Thus, instead of fetching the relevant tiles from the server each time, frequently used tiles are stored in the browser's cache and can be loaded much faster when needed. This caching mechanism reduces the networking overhead between the server and the client, improving the overall performance of the viewer.

4.2.3. Rendering pipeline

A rendering pipeline is a series of GPU operations that process input data, such as 3D geometry, textures, and metadata, to produce a rendered image to display on the screen. For a 3D Tiles 1.1 viewer, the pipeline must efficiently handle loading glTF content from tiles based on the URI of the tiles and processing geometry, materials, and metadata for visualization. The pipeline is designed to leverage the low-overhead API and modern features of WebGPU to achieve better performance in a Web browser. WebGPU enables high-performance 3D graphics in web browsers but lacks direct file input capabilities, requiring integration with other languages and libraries for data handling. To address this, we use the THREE.js library to parse glTF buffers, extracting positions, normals, indices, and feature IDs, which are then loaded into WebGPU GPUBuffer objects for rendering. JavaScript is employed to process vertex attributes, including feature IDs from the EXT_mesh_features extension, to ensure proper attribute linkage for WebGPU's rendering pipeline (Fig. 2).

4.2.4. Interactivity

A viewer must enable rich, dynamic, and semantically meaningful interactions with 3D geospatial models in a web browser. Interactivity refers to the ability of a user to actively engage with a 3D model in a viewer, such as feature picking, dynamic styling, and attribute queries. Feature picking involves clicking or hovering over a 3D object to select it and retrieve associated information.

EXT_mesh_features and EXT_structural_metadata provide the data infrastructure to enable interactivity in 3D Tiles 1.1. Feature IDs allow the viewer to pinpoint specific objects within a 3D model, forming the basis for picking and styling. When a user clicks on a model, the viewer renders feature IDs to an offscreen buffer to determine which feature was selected. Feature IDs are passed through the rendering pipeline to associate geometry with attributes. Feature IDs from EXT_mesh_features act as indices into property tables defined by EXT_structural_metadata. The viewer retrieves attributes (e.g., height, owner) for the selected feature and uses them for styling or UI display. The class diagram of EXT_mesh_features and EXT_structural_metadata is given in Fig. 3. Furthermore, the visualization system can fetch additional metadata from an external API using feature IDs as keys, enabling efficient attribute-based queries in GIS applications, such as retrieving building properties for spatial analysis.

5. Use cases

In the following use cases, attribute-driven tiling is demonstrated through two representative categorical attributes: "roofType" and "isWatertight". These attributes were selected based on a systematic rationale aimed at ensuring transparency, reproducibility, and methodological clarity. The roofType attribute is commonly present in many CityJSON datasets and is a suitable example of an attribute that can be broadly used in tiling scenarios across datasets. In contrast, isWatertight is a dataset-specific attribute produced by a validation workflow from another study of the author group; although not commonly found in publicly available datasets, it offers a direct link between semantic validation results and the tiling logic. Together, these two attributes illustrate the flexibility of the approach: while the glTF writer can transfer all attributes present in the data, making tiling technically possible with any attribute, the chosen examples demonstrate both a widely available real-world attribute and a specialized, analysis-derived attribute. This combination highlights how the method can accommodate diverse user requirements and remain fully reproducible across different contexts.

5.1. Tiling according to validation results

This implementation adopts visualizing the results of a CityJSON model validator. The objective of this validator is to highlight invalid objects identified by the validator by distinguishing them from valid ones in the visualization interface. The validator detects various types of geometric and semantic errors in CityJSON models, and these identified invalidities can be leveraged in an attribute-driven tiling process. In particular, users can select one specific type of invalidity, recorded as an attribute in the CityJSON data, via the interface to give it priority during tiling. As a result, the tiling algorithm stores objects with the selected invalidity and their specified value into the upper tiles. This ensures that such invalid objects are more prominently displayed during

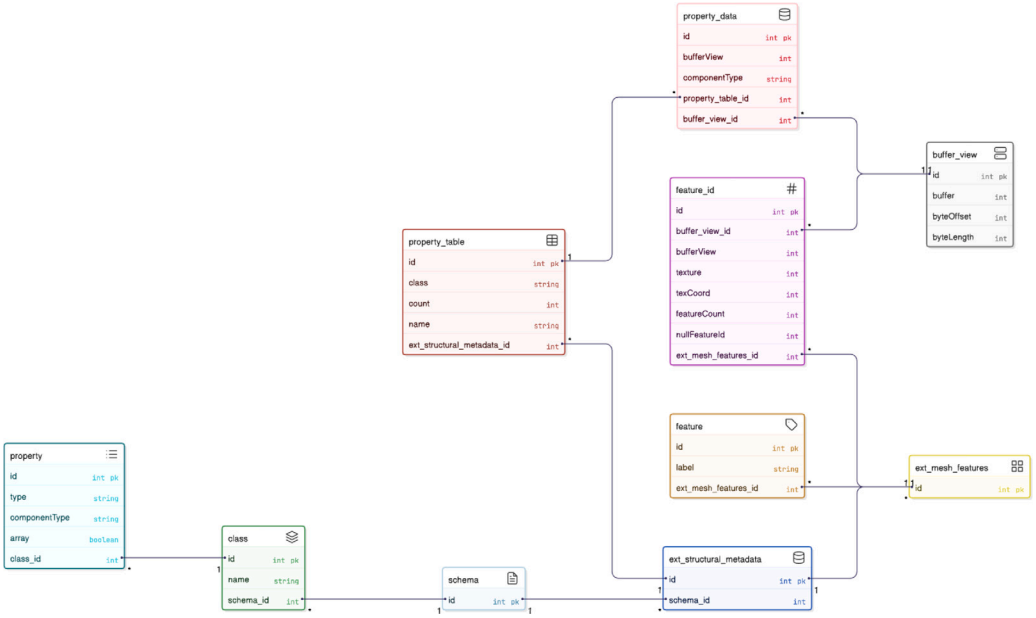

Fig. 3. Class diagram of the EXT_mesh_features and EXT_structural_metadata.

zoom operations. In the current implementation, attribute prioritization follows a user-driven criterion: the selected attribute is "isWatertight", determines which features are aggregated in higher-level tiles. This design ensures transparency and reproducibility, as the prioritization directly reflects user-defined semantics rather than implicit heuristic weighting. Such explicit criteria for prioritization align with adaptive data-handling principles discussed in Rangarajan and Al-Quraishi (2023), where transparent decision logic is emphasized as a prerequisite for context-aware data processing. This integration of validation result data into the tiling logic illustrates the potential of attribute-driven tiling, where spatial subdivision is guided not only by geometry or view-dependent factors, but also by the attributes of the dataset. As a result, the viewer becomes more informative and responsive to the user's needs, supporting targeted scenarios. The context in which attributes are defined or users' needs may vary across different application domains, allowing this approach to be tailored to diverse scenarios and requirements. Fig. 4 demonstrates the flexibility described. The images in the first row include snapshots during different zoom levels, while watertight ("isWatertight == 1") objects are in the upper tiles. The bottom row includes the snapshots of the scenario in which non-watertight objects ("isWatertight == 0") are in the upper tiles. The green and red circles mark the locations where the difference in tiling between the two scenarios is visible.

5.2. Tiling according to roof type

This implementation uses the roofType attribute to prioritize a specific roof category within the CityJSON data. The test dataset employed for this use case is the DenHaag dataset provided by CityJSON.org (2025). The table below lists the roofType codes, their corresponding meanings as defined in the CityGML code list (CityGML Code Lists, 2025), and their occurrence counts within the dataset. Among the 1991 Building and BuildingPart objects, 411 are classified as Gable roofs (code 1030). In our implementation, Gable roofs are given priority during the tiling process, allowing users to see these objects first in the upper tiles, at initial zoom levels. The roof type to be prioritized can later be changed by the user, offering flexibility for different exploration scenarios. Fig. 5 shows the scenario, the images at the first row show the tiling results without an attribute priority, and the second row includes the roofType-driven tiling results (see Table 2).

6. Results and discussion

6.1. Performance and interoperability of the 3D tiler

3D tiler was tested using 4 different CityJSON data sets, namely Rotterdam, Montreal, New York, and Zurich. These datasets are open datasets and can be found at CityJSON.org (2025). Information about the data sets and times to generate 3D tilesets from them can be found in Table 3.

For the first three datasets, namely Rotterdam, Montreal, and New York, the processing time per tile increases almost linearly with the total number of elements. A constant time cost of approximately 20 ms per million elements (vertices + indices) is observed. For Zurich, the processing time per million elements is around 23.3 ms. Although this is still a good performance, it shows a slight deviation compared to the earlier datasets. This difference may be due to approaching memory bandwidth limitations. The ability to process millions of vertices and indices in just milliseconds indicates that our program is highly efficient. 3D Tiler achieves near-linear time complexity on large-scale 3D geospatial datasets by generating nearly equally-sized tiles based on data density, ensuring response times remain consistent across diverse building types and complex geometries, such as intricate landscapes with high vertex and index counts.

For interoperability, the datasets generated by 3D Tiler should be able to display with different OGC 3D Tiles compatible viewers without any issues. To show this, the 3D Tiles datasets produced were rendered in the browser environment via Cesium.js and 3DTilesRenderer.js, both of which are compatible viewers for OGC 3D Tiles (Fig. 6).

6.2. Challenges to implement OGC 3D Tiles 1.1

OGC 3D Tiles 1.1 modernizes the standard by leveraging glTF for improved efficiency and attribute flexibility. By utilizing the `EXT_mesh_features` and `EXT_structural_metadata` extensions, attributes can be assigned at varying granularity, from whole features to individual vertices. This allows for multiple semantic layers within a single mesh (Fig. 7), enabling complex highlighting and interaction. However, this flexibility introduces storage overhead and processing constraints. Feature IDs must be pre-calculated prior to tiling to ensure balanced tile sizes. Furthermore, variable-length string attributes complicate parallelization; the reliance on sequential string

Table 2
CityGML roof type classification and occurrence counts.

Code	English translation	Description	Occurrence count
1000	Flat roof	A roof that is horizontal or nearly level.	1238
1010	Shed roof	A roof with a single slope (Lean-to Roof).	110
1030	Gable roof	A roof with two sloping sides that meet at a ridge.	411
1040	Hip roof	A roof where all sides slope downward to the walls.	40
1060	Mansard roof	A four-sided roof with a double slope on each side (lower slope is steeper).	5
1070	Half-Hip roof	A hip roof where the hip is cut short, leaving a small gable section at the top.	13
1120	Turret roof/Spire roof	A steep, conical or pyramidal roof, typically found on towers.	34
1130	Arch roof/Barrel roof	A roof with a continuous, curved shape.	138
9999	Other/Unknown	A classification for roof types that do not fit into the other defined categories.	1

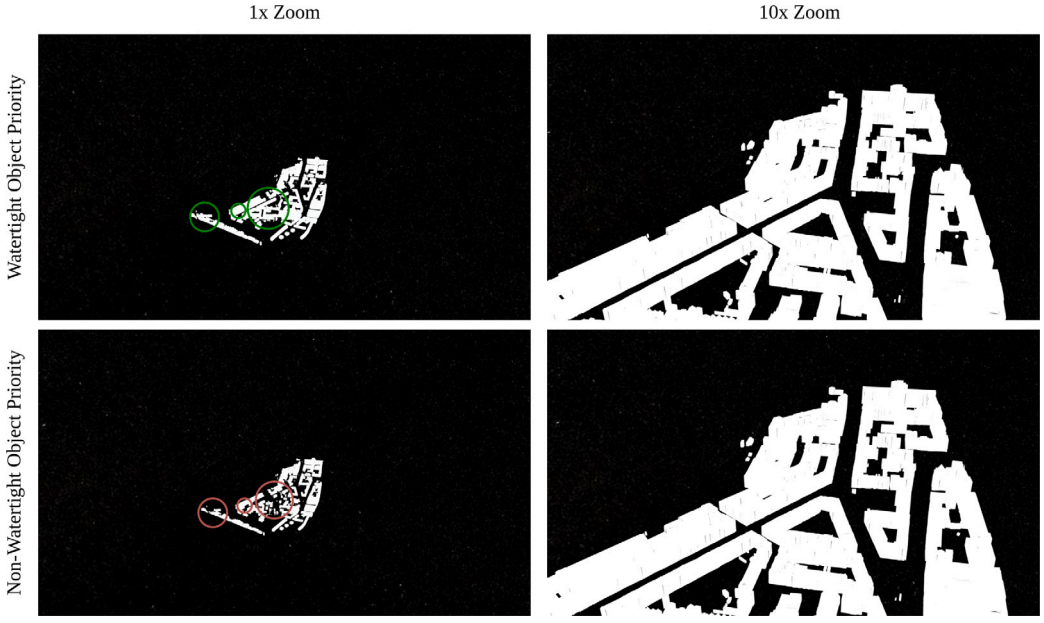

Fig. 4. Demonstration of the validation results (isWatertight = 0 or 1 priority) scenario.

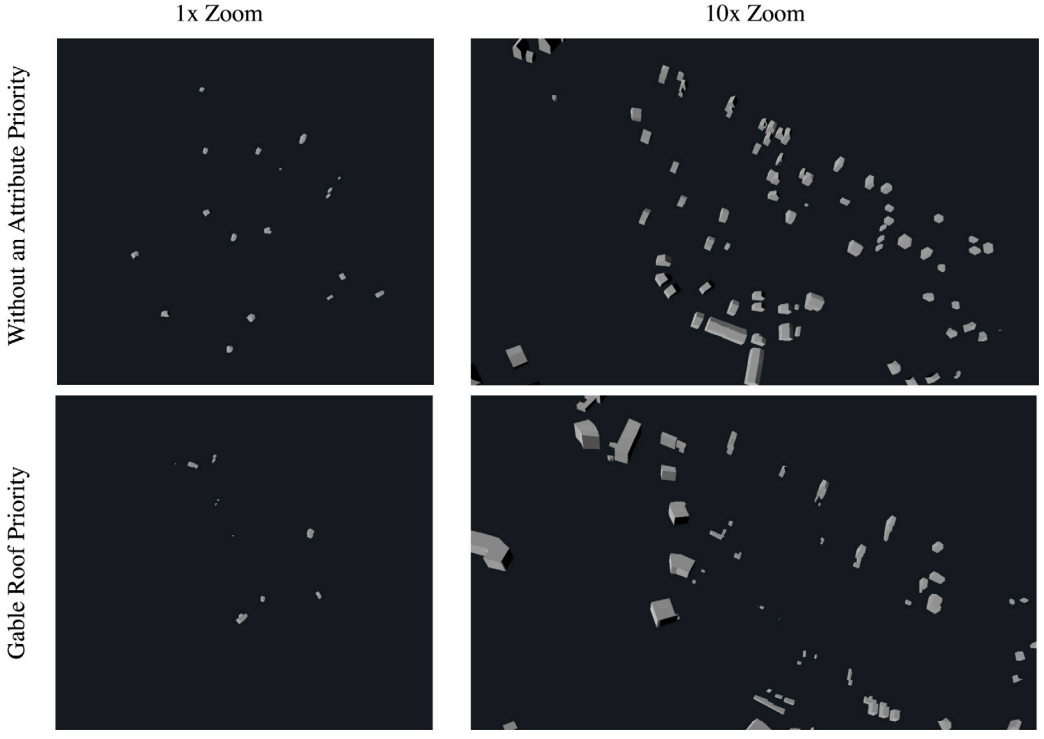

Fig. 5. Demonstration of roofType = 1030 (Gabled roof) priority scenario.

Table 3
Datasets and 3D tiles generation times.

City	Vertices	Triangles	Time (ms)	Tot. Ele. (V + I)	Time for each 1M Ele. (ms)
Rotterdam	85,104	37,478	2.51	122,582	20.48
Montreal	253,060	62,987	6.34	316,047	20.05
New York	3,757,910	1,930,067	113.65	5,687,977	19.97
Zurich	10,917,773	4,928,826	369.60	15,846,599	23.32

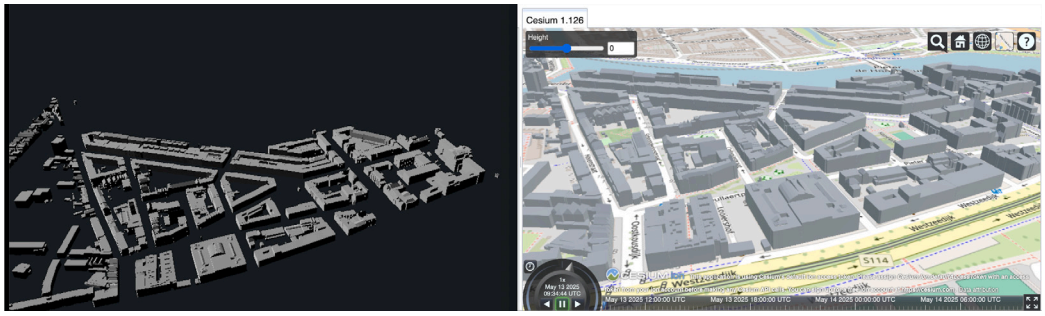

Fig. 6. Same 3D tileset rendered using 3DTilesRenderer.js (left) and Cesium.js (right).

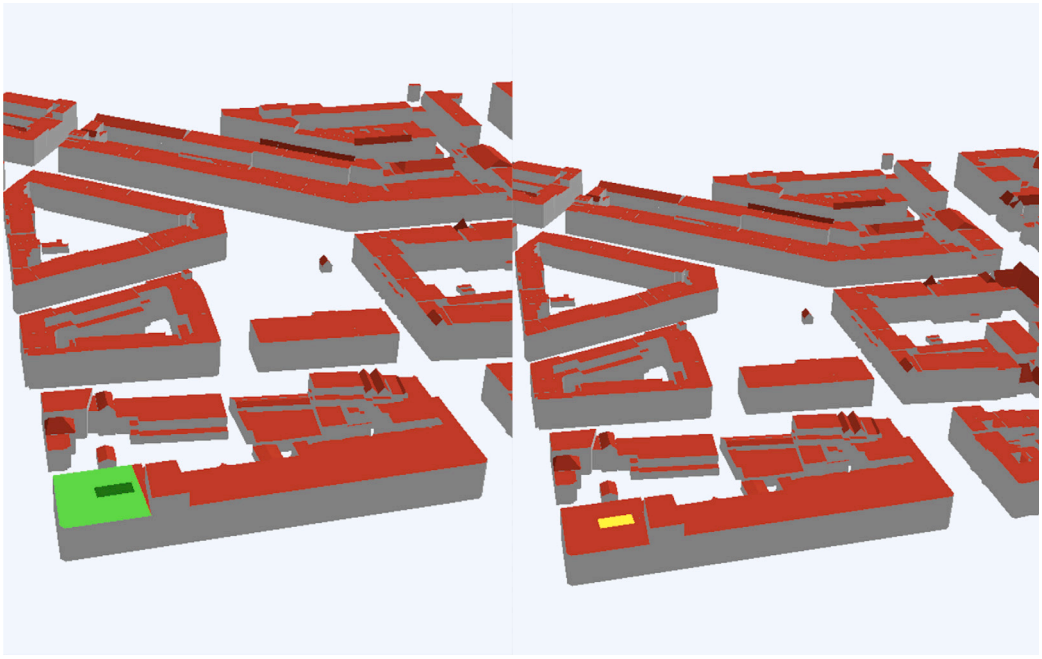

Fig. 7. The Feature_ID hierarchy can be used for different coloring for semantic surfaces(both), selecting and highlighting subcomponents of features (left) or highlighting invalid surface of a feature (right).

offsets creates dependencies between features, making multiprocessing difficult.

6.3. Performance and interoperability of the web viewer

The viewer's performance was tested using one mobile phone and two computers — one with a modern high-end graphics card and the other with a low-end, old one. All the machines screens refresh rates were set to 60 Hz to eliminate the impact of different screen refresh rates. Table 4 shows the system specifications of the machines.

Using these devices, the rendering performance of the viewer has been compared using both WebGL and WebGPU. Table 5 shows the results of the Computer 1, Table 6 shows the results of the Computer 2, and Table 7 shows the results of the Mobile. "10×" stands for that each dataset contains ten times more features than normal.

Additionally, render visuals from different devices have been compared (Fig. 8). The best visual is from Computer 2, and the worst visual is from a mobile device. This is mainly because of the maximum screen resolutions the devices have.

It can be seen in both Tables 5 and 6 that WebGPU render times are slower on both computers. This is because processes such as memory allocation for low-level memory management in WebGPU take more time. Surprisingly, WebGL performs better than WebGPU on Computer 1. This indicates that WebGPU drivers or optimizations are immature or inefficient on older hardware. Poor WebGPU performance is likely due to weak support on older NVIDIA GPUs, immature WebGPU drivers for legacy systems, and a lack of optimized GPU pipelines for older hardware. However, Table 5 shows exactly the opposite results. WebGPU consistently outperforms WebGL in all models. This indicates that WebGPU aligns very well with modern hardware and outperforms WebGL drastically, especially in heavy scenes. For instance, Zurich 10x

Table 4
System specifications.

Computers	CPU	RAM	GPU	OS	Browser
Computer 1	Intel Core i7-7700HQ 2.80 GHz	32 GB DDR4	4 GB GTX1050	Ubuntu 22.04.5 LTS	Chrome 136.0.7103.113
Computer 2	Apple M3 Max	36 GB LPDDR5	Up to 128 GB Apple M3 Max GPU	MacOS Sequoia	Chrome 136.0.7103.113
Mobile	Snapdragon 732G	8 GB LPDDR4X	Adreno (TM) 618	Android 13 Tiramisu	Chrome 136.0.7103.125

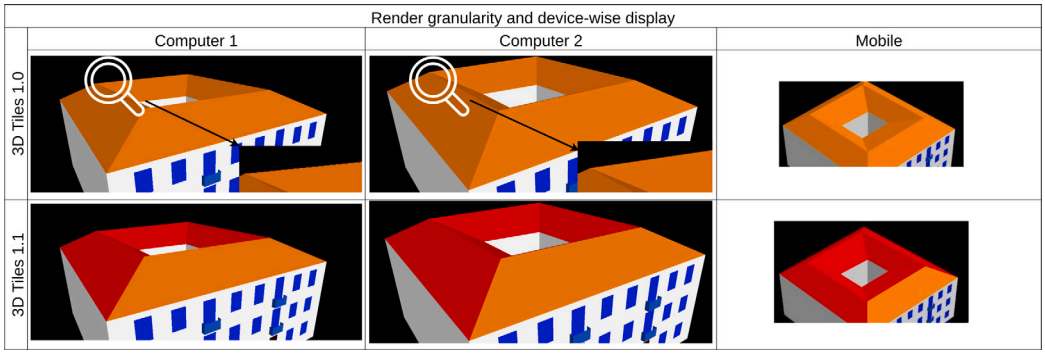

Fig. 8. Different renders from different devices.

Table 5
Computer 1 WebGL vs. WebGPU performance.

Model	WebGL render time (ms)	WebGL FPS	WebGPU render time (ms)	WebGPU FPS
Rotterdam	0.12	60.00	0.16	40.00
Montreal	0.15	60.00	0.17	33.00
New York	0.16	25.00	0.19	3.20
Zurich	0.18	20.00	0.23	1.30
Rotterdam 10x	0.20	46.00	0.15	10.00
Montreal 10x	0.18	48.00	0.19	7.00
New York 10x	0.14	3.00	0.18	0.3
Zurich 10x	0.20	2.00	0.21	0.1

Table 6
Computer 2 WebGL vs. WebGPU performance.

Model	WebGL render time (ms)	WebGL FPS	WebGPU render time (ms)	WebGPU FPS
Rotterdam	0.06	120.00	0.10	120.00
Montreal	0.09	120.00	0.12	120.00
New York	0.08	85.00	0.13	120.00
Zurich	0.07	40.00	0.15	120.00
Rotterdam 10x	0.10	120.00	0.13	120.00
Montreal 10x	0.12	120.00	0.12	120.00
New York 10x	0.03	25.00	0.13	120.00
Zurich 10x	0.04	12.00	0.02	90

was rendered as 12 FPS in WebGL, while the same model was rendered as 90 FPS in WebGPU. New York 10x was rendered as 25 FPS in WebGL, while the same model was rendered as 120 FPS in WebGPU. Although WebGL is more stable across both machines, it begins to struggle at scale. WebGPU is clearly the future, offering major gains on modern hardware.

Table 7 shows that the capability in mobile is not yet as eligible for use as in computers for such use, as is foreseeable. The WebGL performance decreases drastically as models grow larger; in fact, it does not respond in Zurich, New York 10x, and Zurich 10x models. The WebGPU setup worked well on mobile, but the rendering process crashed a few seconds after initialization for all models due to the immaturity of the WebGPU engine on mobile browsers.

Table 8 shows WebGPU performance on Mozilla Firefox on different machines. Since WebGPU is not yet enabled on Mozilla Firefox standard releases, the latest nightly version has been used, forcing it WebGPU enabled. Tables 8, 5, and 6 show that on Computer 1, in small models, WebGPU on Firefox performs worse than on Chrome, but in bigger models, Firefox WebGPU performance is better than Chrome. On the other hand, in Computer 2, more consistent results are obtained only with Firefox, just 3 FPS slower than Chrome, only in the biggest model, Zurich 10x. These results show that Mozilla Firefox shows inconsistent results in older machines on WebGPU. In addition, the use of WebGPU required a sort of activation of experimental flags in the Chrome browser compared to desktop browsers. The WebGPU is still immature on desktop browsers, especially for use in legacy hardware, so it needs a longer time to use the mature WebGPU API, considering the downsizing of the improvements to smaller architecture processors as in mobile.

Performance improvement in WebGPU contributes not only to improving viewer performance, but also to improving the performance of the tiler and the overall application. Since WebGPU increases performance on the viewer side, the tile data size can be increased accordingly. This enables the creation of larger tiles without compromising rendering performance, helps reduce the total number of tiles, and decreases the network overhead between the server and the client

6.4. Challenges and limitations

Implementing OGC 3D Tiles 1.1 in both the 3D tiler, the tool or pipeline that generates 3D Tiles tilesets, and the 3D viewer, and also implementing WebGPU on the rendering pipeline of the 3D viewer presents several challenges. While these standards improve scalability, flexibility, and interactivity for geospatial applications, they also introduce complexities in data processing, rendering, and software-developing processes.

3D Tiles 1.1's implicit tiling replaces the explicit listing of tiles in tileset.json with an algorithmic approach using quadtree or octree subdivision, URI templates, and subtree files with availability bitstreams. As a consequence, generating implicit tiling requires defining regular subdivision schemes such as QuadTree or Octree. In some application-specific tasks, such as spatial queries, R-Tree or any other irregular data structure may be more efficient (Usta et al., 2024). However, the implicit tiling feature makes it impossible to implement irregular schemes such as R-Tree. Converting a CityJSON dataset to a quadtree-based implicit tileset requires mapping irregular data to a regular scheme, potentially creating empty tiles. Another disadvantage of this feature is that it increases computational overhead by creating subtree files with bitstreams to indicate tile and content availability that is more complex than 1.0's explicit tiling. Hence, this feature must be selected carefully by the user according to its application. If the application will not sufficiently benefit from this feature, it should not be selected.

Table 7
Mobile WebGL vs. WebGPU performance.

Model	WebGL render time (ms)	WebGL FPS	WebGPU render time (ms)	WebGPU FPS
Rotterdam	0.35	60.00	0.60	60.00
Montreal	0.30	60.00	0.55	60.00
New York	0.45	7.00	1.00	60.00
Zurich	Not responding	Not responding	0.60	30.00
Rotterdam 10x	0.70	25.00	1.00	60.00
Montreal 10x	0.75	24.00	0.93	60.00
New York 10x	Not responding	Not responding	1.00	8.00
Zurich 10x	Not responding	Not responding	1.3	3.00

Table 8
WebGPU performance in Mozilla Firefox on different devices.

Model	Computer 1 render time (ms)	Computer 1 FPS	Computer 2 render time (ms)	Computer 2 FPS
Rotterdam	0.15	20.00	0.15	120.00
Montreal	0.18	20.00	0.15	120.00
New York	0.20	20.00	0.20	120.00
Zurich	0.20	20.00	0.15	120.00
Rotterdam 10x	0.25	23.00	0.15	120.00
Montreal 10x	0.20	20.00	0.18	120.00
New York 10x	0.20	12	0.10	120
Zurich 10x	0.13	0.5	0.08	86

3D Tiles 1.1 replaces 1.0's batchTable with `EXT_mesh_features` and `EXT_structural_metadata` glTF extensions for fine-grained attribute support. Implementing these features correctly requires assigning feature IDs to geometry subcomponents of features and involves mapping CityJSON semantics to feature IDs, which can be complex and may require data restructuring, especially for large-scale datasets. Mapping input data to glTF extensions causes high complexity, increased memory, and processing requirements for generating binary buffers and schemas.

Binary formats are easy for machines to process, but difficult to debug for developers. Storing string attributes in binary buffers is especially difficult. For each city object, attributes and string offsets must be calculated and stored for string types. Calculation of string offset values depends on the previous values, which are calculated sequentially and prevent them from being parallelized using methods such as multiprocessing.

7. Conclusion

This paper explores the new standards, specifically OGC 3D Tiles 1.1 and WebGPU, for the Web-based visualization of UDTs. For this purpose, two web-based software components have been developed to implement these standards.

Key features such as EXT_mesh_features and EXT_structural_metadata significantly enhance the precision of attribute integration and interactivity within 3D geospatial models. These new features enable finer-grained control, allowing users to interact with individual vertices, texels, or specific features in a mesh. The introduction of feature IDs and their linkage to property tables has facilitated more efficient querying and rendering of spatial data, essential for dynamic styling and interaction in various application domains, including GIS and digital twins. Furthermore, the hierarchy of feature IDs enables the storage of semantic attributes of the CityGML data model, which was not possible in 3D Tiles 1.0.

Additionally, the integration of validation results into the tiling process for the visualization use case showcases the potential of attribute-driven tiling, where spatial subdivision is guided not only by geometric or view-dependent factors but also by the specific attributes of the dataset based on the purpose of the application. This ability to prioritize the attributes, such as "isWatertight" and "roofType", allows for a more informed and user-centric visualization experience according to the use case of the application.

The performance evaluation of the 3D Tiler and its interoperability with different viewers has shown promising results. The tiling process scales effectively with larger datasets, and the use of implicit tiling in 3D Tiles 1.1 provides a more compact and efficient approach for managing large-scale datasets.

The comparative analysis between WebGL and WebGPU rendering pipelines demonstrated that WebGPU's performance is highly conditional on the rendering environment and dependent on modern hardware and mature driver support. While WebGPU consistently and drastically outperforms WebGL on modern hardware, demonstrated by rendering gains from 12 FPS to 90 FPS on a Zurich 10x model, its performance is notably slower and often worse than WebGL on older or legacy systems due to the immaturity of its drivers and optimization for legacy hardware. Furthermore, WebGPU's current implementation is highly unstable on mobile browsers, often crashing the application, and exhibits inconsistent performance across different desktop browsers like Chrome and Mozilla Firefox, further highlighting its dependence on hardware maturity and stable driver support. This performance improvement in WebGPU is significant because it not only improves viewer performance but also allows for the creation of larger, denser tiles without compromising rendering speed, thereby helping to reduce the total number of tiles and network overhead.

Beyond its technical novelty, this study contributes to the broader goal of interoperability in urban digital twins. By supporting open standards such as CityJSON and OGC 3D Tiles 1.1, and implementing them in a fully web-based environment, the proposed pipeline promotes transparent data exchange and sustainable reuse of 3D city information across platforms. This aligns with the principles outlined in Khan and Ahmad (2025), where interoperability and standardization are identified as key enablers of scalable and sustainable digital twin ecosystems.

However, challenges remain, particularly in terms of data processing complexities and the implementation of WebGPU across diverse hardware configurations. Despite these challenges, the advancements introduced in 3D Tiles 1.1 pave the way for more scalable, flexible, and interactive geospatial applications. As WebGPU continues to mature and its hardware and software support improve, the potential for even larger and more efficient 3D tilesets will increase. Hence, further contributions of WebGPU to the future development of the digital twin, metaverse, and GIS fields will take a wider place.

Finally, this study demonstrates that the OGC 3D Tiles 1.1 standard enables substantially richer user interactions compared to version 1.0, and WebGPU delivers significantly improved rendering performance over WebGL on modern graphics hardware.

CRediT authorship contribution statement

Ziya Usta: Writing – original draft, Visualization, Methodology. **Alper Tunga Akın:** Writing – original draft, Visualization, Methodology. **Ken Arroyo Ohori:** Writing – review & editing, Supervision, Investigation. **Jantien Stoter:** Writing – review & editing, Supervision, Investigation.

Funding

This work was supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) with application numbers 1059B142301188 and 1059B192402172.

Declaration of competing interest

The authors declare no conflict of interest.

Data availability

The link of used data is shared in the text.

References

3DBag Viewer, 2025. 3Dbagviewer. URL https://3dbag.nl/en/viewer.

3DTilesRendererJS, 2025. 3Dtilesrendererjs. URL https://github.com/NASA-AMMOS/3DTilesRendererJS.

Arroyo Ohori, Ken, 2020. Azul: A fast and efficient 3d city model viewer for macos. Trans. GIS 24, 1165–1184.

Beil, Christof, Kendir, Murat, Ruhdorfer, Roland, Kolbe, Thomas H, 2022. Dynamic and web-based 4d visualization of streetspace activities derived from traffic simulations and semantic 3d city models. ISPRS Ann. Photogramm. Remote. Sens. Spatial Inf. Sci. 10, 29–36.

Belayneh, Tamrat, 2022. Indexed 3d scene layers (i3s)–an efficient encoding and streaming ogc community standard for massive geospatial content. Int. Arch. Photogramm. Remote. Sens. Spatial Inf. Sci. 43, 349–355.

CesiumJS, 2025. Cesiumjs. URL https://cesium.com/platform/cesiumjs/.

Chaturvedi, Kanishk, Yao, Zhihang, Kolbe, Thomas H, 2015. Web-based exploration of and interaction with large and deeply structured semantic 3d city models using html5 and webgl. In: Bridging Scales-Skalenübergreifende Nah- und Fernerkundungsmethoden, 35. Wissenschaftlich-Technische Jahrestagung der DGPF.

Chen, Changjie, Han, Yu, Galinski, Andrea, Calle, Christian, Carney, Jeffrey, Ye, Xinyue, van Westen, Cees, 2024. Integrating urban digital twin with cloud-based geospatial dashboard for coastal resilience planning: A case study in florida. J. Plan. Educ. Res. 0739456X251316185.

CityGML Code Lists, 2025. Citygml code lists. URL https://www.citygmlwiki.org/index.php?title=CityGML_Code_Lists.

citygml-to-3dtiles, 2025. Citygml-to-3dtiles-tiles. URL https://github.com/njam/citygml-to-3dtiles.

CityJSON.org, 2025. Cityjsonorg. URL https://www.cityjson.org/datasets/.

Colin, Clement, Vinasco-Alvarez, Diego, Samuel, John, Servigne, Sylvie, Bortolaso, Christophe, Gesquière, Gilles, 2024. A model-driven methodology for integrating heterogeneous 3d geospatial urban entities. AGILE: GIScience Ser. 5, 3.

Coors, Volker, Padsala, Rushikesh, 2024. Urban digital twins empowering energy transition: citizen-driven sustainable urban transformation towards positive energy districts. Int. Arch. Photogramm. Remote. Sens. Spatial Inf. Sci. 48, 51–56.

Gaillard, Jérémy, Peytavie, Adrien, Gesquiére, Gilles, 2020. Visualisation and personalisation of multi-representations city models. Int. J. Digit. Earth 13, 627–644.

Gesquière, Gilles, Manin, Alexis, 2012. 3D visualization of urban data based on citygml with webgl. Int. J. 3D Inf. Model. (IJ3DIM) 1, 1–15.

Hairuddin, A, Azri, S, Ujang, U, Cuétara, MG, Retortillo, GM, Salleh, S Mohd, 2019. Development of 3d city model using videogrammetry technique. Int. Arch. Photogramm. Remote. Sens. Spatial Inf. Sci. 42, 221–228.

Ilgar, Azer, Kara, Abdullah, Çağdaş, Volkan, 2024. Identifying legal, bim data and visualization requirements to form legal spaces and developing a web-based 3d cadastre prototype: A case study of condominium building. Land 13, 1380.

iTowns, 2025. Itowns. URL https://www.itowns-project.org/.

Jaillot, Vincent, Servigne, Sylvie, Gesquière, Gilles, 2020. Delivering time-evolving 3d city models for web visualization. Int. J. Geogr. Inf. Sci. 34, 2030–2052.

Khan, Muuhammad Nawaz, Ahmad, Imtiaz, 2025. Harnessing digital twins: Advancing virtual replicas for optimized system performance and sustainable innovation. Babylon. J. Mech. Eng. 2025, 18–33.

La Guardia, Marcello, Koeva, Mila, Díaz-Vilariño, Lucia, Nourian, Pirouz, 2024. Open-source solutions for real-time 3d geospatial web integration. In: ISPRS TC IV Mid-Term Symposium "Spatial Information To Empower the Metaverse". Copernicus, pp. 289–295.

Labetski, Anna, Vitalis, Stelios, Biljecki, Filip, Ohori, Ken Arroyo, Stoter, Jantien, 2023. 3D building metrics for urban morphology. Int. J. Geogr. Inf. Sci. 37, 36–67.

Mao, Bo, Ban, Yifang, Laumert, Björn, 2020. Dynamic online 3d visualization framework for real-time energy simulation based on 3d tiles. ISPRS Int. J. Geo Inf. 9, 166.

obben, Stein K, 2024. Integrating 3d Functionality Into a Web Application for Sharing Geo-Information (Msc thesis in geomatics). Delft University of Technology, Master thesis.

obj23tiles, 2025. Objto3d-tiles. URL https://github.com/PrincessGod/objTo3d-tiles.

Open Geospatial Consortium, 2019. OGC 3D Tiles. Technical Report 18–053r2, Open Geospatial Consortium, Internal reference number of this OGC® document.

Prandi, Federico, Devigili, Federico, Soave, Marco, Staso, Umberto Di, Amicis, Raffaele De, 2015. 3D web visualization of huge citygml models. Int. Arch. Photogramm. Remote. Sens. Spatial Inf. Sci. 40, 601–605.

Rangarajan, Sarathkumar, Al-Quraishi, Tahsien, 2023. Navigating the future of the internet of things: emerging trends and transformative applications. Babylon. J. Internet Things 2023, 8–12.

Santhanavanich, Thunyathep, Wuerstle, Patrick, Padsala, Rushikesh, Coors, Volker, 2022. Digital 3d city models towards urban data platform using ogc 3d geovolumes api. In: 42. Wissenschaftlich-Technische Jahrestagung der DGPF, 5.-6. Oktober 2022 in Dresden. Geschäftsstelle der DGPF, pp. 237–242, 30.

Usta, Ziya, 2021. The Design and Development of a Web-Based 3D Geographic Information Management Framework (Doctoral thesis). Karadeniz Technical University, The Graduate School of Natural and Applied Sciences.

Usta, Ziya, 2024. Webgpu: A new graphic api for 3d webgis applications. Int. Arch. Photogramm. Remote. Sens. Spatial Inf. Sci. 48, 377–382.

Usta, Ziya, Cömert, Çetin, Akın, Alper Tunga, 2024. An interoperable web-based application for 3d city modelling and analysis. Earth Sci. Inform. (ISSN: 1865-0473) 17, 163–179. http://dx.doi.org/10.1007/s12145-023-01167-5.

Vitalis, S., Labetski, A., Boersma, F., Dahle, F., Li, X., Ohori, K. Arroyo, Ledoux, H., Stoter, J., 2020. Cityjson+ web=ninja. ISPRS Ann. Photogramm. Remote. Sens. Spatial Inf. Sci. 6, 167–173.

Würstle, Patrick, Padsala, Rushikesh, Santhanavanich, Thunyathep, Coors, Volker, 2022. Viability testing of game engine usage for visualization of 3d geospatial data with ogc standards. ISPRS Ann. Photogramm. Remote. Sens. Spatial Inf. Sci. 10, 281–288.

Yu, Dayu, Yue, Peng, Wu, Binwen, Biljecki, Filip, Chen, Min, Lu, Luancheng, 2025. Towards an integrated approach for managing and streaming 3d spatial data at the component level in spatial data infrastructures. Int. J. Geogr. Inf. Sci. 39, 847–871.

Zhan, Wenxiao, Chen, Yuxuan, Chen, Jing, 2021. 3D tiles-based high-efficiency visualization method for complex bim models on the web. ISPRS Int. J. Geo Inf. 10, 476.