Advances in Computer, Signals and Systems (2026)
Clausius Scientific Press, Canada

DOI: 10.23977/acss.2026.100217
ISSN 2371-8838 Vol. 10 Num. 2

A Web-Based BIM and Virtual Reality System for Lightweight Engineering Training

Hao Lei

School of Architectural Engineering, Science and Technology College of Hubei University of Arts and Science, Xiangyang, 441000, Hubei, China

Keywords: BIM; virtual reality; Web3D; model conversion; lightweight rendering

Abstract: This study developed a lightweight web system for interactive BIM and virtual reality models. The system combines an automated RVT to FBX to GLB conversion pipeline, Draco compression, IFC parsing, and Web Worker based processing. Raycaster, OutlinePass, and Pointer Lock API support component selection, visual feedback, and first-person navigation. Performance tests on a representative engineering model showed fast loading, moderate memory use, and stable real-time rendering across both discrete and integrated graphics. The results indicate that browser-based BIM visualization can support interactive engineering applications while reducing software installation and system maintenance requirements.

1. Introduction

Building information modeling (BIM) models integrate geometric representation with data on materials, component properties, and spatial relations. In a browser-based setting, the practical question is how to preserve the connection between a displayed object and this engineering information without requiring each user to install an authoring system. The present work therefore treats web delivery as a technical problem of model transformation, data access, rendering, and interaction rather than as a replacement for desktop authoring.

They contain geometry as well as information about materials, component characteristics, and spatial relations. Hence, they may also serve design coordination purposes, assist during construction, and provide facilities-related knowledge later on. Some previous research concerning IFC-oriented web solutions indicated that one could rely on an openly defined modeling scheme to carry out conversions from other formats, obtain values of certain features, and present results using standard web technology [1]. Then there is Web3D and VR, which make such data available via interactive environments. However, most current approaches connected with both BIM itself and VR involve the use of programs such as Revit or engines such as Unity or Unreal Engine. In turn, these entail having particular programs installed locally, obtaining the necessary licenses, possessing proper equipment, preparing something specifically for that tool beforehand, and so on; hence, problems arise if someone wants to view very large files or open projects remotely.

It was an investigation into one possible browser-based solution for engineering models of this kind. Four concrete problems had to be solved: conversion of authoring models into efficient forms for running them; reading of IFC data when there is no Revit installation present; avoiding blocking

of the user interface while geometry is being parsed; and achieving sufficient interactivity even when using modest graphics cards. Rendering is done by Three.js and WebGL; GLB files deliver content at runtime, but IFC remains available if people want to exchange their models openly. The evaluation concerned itself with questions about file sizes, load times, memory consumption, frames per second, and browser support.

The technical objective is not simply to place a three-dimensional model in a web page. The system must preserve a stable association between each rendered object and its engineering identity, prevent long parsing operations from freezing the interface, and keep the initial resource request within a practical range. The prototype separates geometry, metadata, and interaction state. GLB is used as the delivery asset, whereas IFC remains available as an exchange-oriented source when open data access is required. This arrangement also keeps the runtime architecture independent of a particular desktop BIM license.

2. System design and implementation

2.1 Architecture and data flow

It is a light-browser type prototype. Revit is involved only when generating models. With a pyRevit script, one can read an RVT model, pick out necessary items from its elements and properties, obtain ID numbers and other information, convert FBX files through Blender in batches, clean up meshes, carry out material baking, export geometry as .glb from command lines, compress with Draco, and so on. In this way, there is hardly any difference compared to some previous Web3D BIM research concerning model reduction, elimination of duplication, and organization of scene information suitable for browsers [2].

During conversion, the element identifier is carried forward with the mesh so that selection in the browser can be traced back to the corresponding engineering record. Mesh cleanup removes unnecessary duplicate data and normalizes material information before export. The delivered GLB file is therefore intended to contain only the geometry, textures, and identifiers needed for viewing and querying. Keeping this boundary explicit is important: a browser scene should not attempt to reproduce every authoring operation, but it must preserve enough semantic association for a selected component to remain interpretable during review.

There are six typical kinds of engineering scenes included on the Runtime Layer: site layout, foundation pit, foundation, main structure, prefabricated structure, and interior finishing. Users can carry out operations such as orbiting, zooming in and out, clipping, selecting building elements, checking their information, and moving about within the scenario itself from a first-person viewpoint; IFC 2x3 files and IFC 4 files can also be loaded. Some previous studies have found that there is no problem combining IFC access with WebGL interaction and attribute inspection in a browser-based viewer [3]. In the present case, an IFC file is parsed by means of a Web worker so that the main thread remains free for interface updates and rendering.

The runtime layer also separates downloadable assets from operations performed after loading. Scene files are requested only when a user enters the related engineering scene, rather than being loaded as one undifferentiated package. The interface then applies visibility controls before interaction calculations are performed. This reduces visual clutter and makes it easier to work with a large number of components. It also provides a basis for later additions such as discipline-based views, phase-specific display, and scene switching without requiring the full model to remain visible at all times.

2.2 Interaction and rendering

Component picking is done by means of a Raycaster test between the pointer position and scene meshes. After one component has been picked, OutlinePass draws an outline on top of the selected item. Then, the picked ID can be looked up against the metadata pulled from the original model file, so both geometry and engineering attributes are available at this point. Filtering options per discipline, floor, or element type also help reduce the amount of displayed information and hence allow better control over what users see, especially when dealing with large amounts of data.

Interaction is organized around a small set of explicit states: the current camera position, the visible object set, the selected component, and the associated metadata record. When the pointer intersects a mesh, the system first determines whether the object is currently selectable and then reads its identifier before updating the information panel. The outline, property query, and filtering functions are linked to the same component reference, which reduces the risk that the displayed attribute record and the highlighted geometry diverge after a view or filter has changed.

First-person navigation employs the Pointer Lock API to change views and processes keyboard strokes for locomotion. All scene creation, input processing, and drawing occur in the main thread; parsing IFC files and preparing data occur in the worker. In this manner, one avoids keeping users waiting because parsing has started, and one also follows what others have done when implementing browser-based BIM systems using WebGL [1, 3].

3 Model conversion and performance optimization

3.1 Automated conversion and compression

The conversion pipeline preserves the component identity required for inspection while reducing the runtime asset size. The pyRevit stage exports model elements and identifiers. Blender converts the meshes, normalizes materials, and produces the GLB resource. Related studies have approached the same problem through semantic lightweighting, spatial indexing, and progressive scene management [2, 5, 6]. In the present workflow, Draco compression reduces the GLB size to approximately 10 to 30 percent of the original FBX size.

Compression alone is not sufficient for dependable web delivery. The conversion step also checks whether objects that need individual inspection remain separately addressable after mesh processing. Materials are simplified where detailed texture information is not required for the intended review task, and repeated resources are reused where possible. These choices balance visual readability against transfer and initialization cost. The resulting asset can be loaded directly by the browser renderer, while the original engineering model remains available outside the delivery package for authoring, revision, and more detailed analysis.

A typical 5,000-component model produced a 14.8 MB GLB file. Under a 100 Mbps network condition, its first load took 2.8 s. The pipeline also limits texture quality and groups geometry that can be rendered together. These settings reduce transfer volume and the number of resources that the browser must create during initialization. The model preparation process can handle Revit 2023 models from different disciplines and can process up to 5,000 elements in one conversion run.

3.2 IFC parsing and runtime memory

IFC support is useful when the user has no Revit license or receives an exchange file from another authoring tool. The web-ifc library is compiled to WebAssembly and exposes IFC geometry and property data to JavaScript [6]. The worker receives the file buffer, performs the parsing, and returns identifiers and geometry data to the main thread. Scene objects are then created from the

returned data. This order avoids building a complete scene before the parser has finished.

The worker-based arrangement is important because IFC files may contain both geometric descriptions and extensive property data. If parsing, geometry preparation, and interface rendering compete for the same execution path, visible operations such as rotation or selection can become unresponsive. In the prototype, the main thread remains responsible for the scene and user interface, while the worker returns prepared information through message passing. The boundary also makes error handling clearer: an unsupported or incomplete exchange file can be reported without interrupting a model that has already been loaded as a GLB scene.

Loading the typical building model consumed approximately 280 MB of runtime memory, below the 500 MB target used in the project. At 1,920 x 1,080 resolution, the frame rate reached 85 to 120 FPS with an NVIDIA RTX 4060. At 1,366 x 768 resolution, it reached 45 to 60 FPS with Intel UHD Graphics 730 integrated graphics. The latter result is relevant for laboratories and office computers that do not have a discrete GPU.

Performance observations should be read together with the test conditions rather than as a universal hardware guarantee. Resolution, graphics capability, browser implementation, scene composition, and the number of simultaneously visible objects all affect the rendering result. The measurements therefore provide a practical reference for the representative model used in this work. They also show why the system keeps model delivery, visibility control, and browser-side rendering as separate technical concerns. The reported values should be treated as a deployment-oriented baseline and supplemented by project-specific checks when model scale or client hardware changes.

4. Technical evaluation and limitations

4.1 Compatibility and deployment

Therefore, the size of the production package (HTML + JS + CSS) is less than 5 MB. Model files are fetched separately when needed. This setup was tested under the latest released versions of Chrome, Edge, Firefox, and Safari and worked well on these browsers. Likewise, it also succeeded on several operating systems such as Windows, macOS, Linux, iOS, and Android. Since the execution environment is provided by browsers, users do not have to install full copies of BIM authoring software or another kind of VR engine onto each machine. Changes may occur only at the level of applications or assets; therefore, there is no need to repeat this procedure on all workstations later.

Deployment also benefits from separating the application shell from model resources. The browser can cache the HTML, JavaScript, style sheets, and commonly used interface assets, whereas a model is requested only when it is needed. This makes routine updates more manageable because a change to the interaction code does not require repackaging every model file. The same principle supports incremental expansion: additional project scenes can be placed beside existing assets and exposed through the application configuration, while the client-facing operating procedure remains unchanged.

4.2 Limitations and next steps

At present, the prototype provides browsing, querying properties, and navigating capabilities for ordinary engineering models, but it does not support very large federated models, simultaneous editing by multiple users, full-construction-process simulation, etc. For larger scenes, server-side spatial index construction, model subdivision into small pieces (tiles), LOD scheduling, and cloud-edge-browser architectures may help shorten loading time and reduce network burden [4]; semantically based scheduling together with view-dependent progressive transmission can reduce

the amount of actually visible data [5, 6]. More work still needs to be done on the issues of IFC property mapping and versioning, since authoring tools sometimes fail to produce exactly the same property structure when performing the export operation. Extension to collision detection, construction sequence animation, and other kinds of common marks may prove useful if more attention is focused on technical briefing and enterprise training.

Further optimization can be organised as a staged process. At the asset level, geometry can be partitioned according to spatial location and usage frequency. At runtime, visibility rules and level-of-detail policies can limit the number of meshes submitted for rendering. At the service level, a manifest can describe model versions, dependencies, and loading priorities. These measures would allow the same browser application to accommodate larger projects gradually while retaining the present lightweight deployment pattern. Clear version metadata would also make it easier to identify which asset revision is being viewed during engineering review and maintenance activities across distributed project teams.

5. Conclusions

The prototype combines an automated RVT to FBX to GLB pipeline with Draco compression, browser-side IFC parsing, and worker-based data processing. Raycaster, OutlinePass, and Pointer Lock API support component inspection, visual feedback, and first-person navigation. Tests on a 5,000-component model produced a 14.8 MB GLB asset, a 2.8 s first load, approximately 280 MB of runtime memory, and interactive frame rates on both discrete and integrated graphics. These results show that a browser-based workflow can handle practical engineering visualization while reducing the installation and maintenance burden associated with desktop BIM and VR software.

Acknowledgement

This work was supported by Scientific Research Program of Hubei Provincial Department of Education(B2023478).

References

[1] Hu Z Z, Zhang X Y, Wang H W, Kassem M. Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with web-based tools[J]. Automation in Construction, 2016, 66: 29-42. DOI: 10.1016/j.autcon.2016.02.001.

[2] Liu X J, Xie N, Tang K, Jia J Y. Lightweighting for Web3D visualization of large-scale BIM scenes in real-time[J]. Graphical Models, 2016, 88: 40-56. DOI: 10.1016/j.gmod.2016.06.001.

[3] Deng H X, Liu Y, Liu Y M, Xu L, Wang C G. Web-based approach for systematic analysis and interactive visualization of the Industry Foundation Classes data: A case study in duct design[J]. Journal of Asian Architecture and Building Engineering, 2023, 22(3): 1361-1372. DOI: 10.1080/13467581.2022.2085715.

[4] Li K, Zhao H T, Zhang Q, Jia J Y. CEBOW: A Cloud-Edge-Browser Online Web3D approach for visualizing large BIM scenes[J]. Computer Animation and Virtual Worlds, 2022, 33(2): e2039. DOI: 10.1002/cav.2039.

[5] Chen Q X, Chen J, Huang W M. Visualizing large-scale Building Information Modeling models within indoor and outdoor environments using a semantics-based method[J]. ISPRS International Journal of Geo-Information, 2021, 10(11): 756. DOI: 10.3390/ijgi10110756.

[6] Sun Y C, Ma J S, She J F, Zhao Q, He L X. View-dependent progressive transmission method for 3D building models[J]. ISPRS International Journal of Geo-Information, 2021, 10(4): 228. DOI: 10.3390/ijgi10040228.