

SmartSpace: Smart Interior Design Generation

Nilay Pandya

Information Technology
Vidyalankar Institute of
Technology
Mumbai, India

nilaypandya2004@gmail.com

Soham Ghogare

Information Technology
Vidyalankar Institute of
Technology
Mumbai, India

ghogresoham@gmail.com

Ishaan Dhuri

Information Technology
Vidyalankar Institute of
Technology
Mumbai, India

ishaandhuri@gmail.com

Prof. Vinita Bhandiwad

Information Technology
Vidyalankar Institute of
Technology
Mumbai, India

vinita.bhandiwad@vit.edu.in

Abstract — Complex computer-aided design (CAD) tools have been able to move from desktop environments to the browser thanks to the quick development of web technology. In this paper, we present *Blueprint3D*, a web-based, lightweight application framework for interactive architectural visualization and interior design. The system offers a dual-view interface that synchronizes a 2D floor plan editor with a real-time 3D immersive environment by utilizing the *Three.js* framework and *WebGL* standards. Walls, rooms, and furniture may be dynamically manipulated according to the suggested architecture's modular design pattern, which tightly separates the geometric data model from the visualization controllers. Automated mesh generation for architectural parts, collision detection for item placement, and a responsive user interface without the need for client-side plugins are some of the key features. We validate the efficacy of JavaScript-based rendering engines in providing high-performance, user-friendly design tools by showcasing the system's capacity to enable quick spatial prototyping and customization.

Keywords — Web-based CAD, *Three.js*, Interior Design, *WebGL*, 3D Visualization, Interactive Graphics, Floor Planning, Spatial Layout.

I. INTRODUCTION

The field of interior design and architectural planning has completely changed as a result of the quick development of digital technology. Rigid, desktop-bound workflows have given way to dynamic, accessible, and collaborative web-based ecosystems. There is a pressing need for tools that democratize high-fidelity spatial design as urbanization picks up speed and the global interior design market—valued at over \$150 billion in 2023 and expected to surpass \$200 billion by 2030—prioritizes customized, sustainable living places. Despite their accuracy, traditional computer-aided design (CAD) systems provide significant obstacles, such as high license costs (sometimes thousands of dollars per year), resource-intensive installations, and lengthy learning curves that turn off non-professionals. The persistent gap between immersive 3D visualizations and static 2D floor plans exacerbates these problems by impeding stakeholder communication and intuitive iteration in both consumer and business contexts.

Unprecedented capabilities for client-side 3D rendering have been made possible by developments in browser-native graphics APIs, particularly *WebGL* 2.0 and high-performance JavaScript engines like *V8*, which eliminate the need for proprietary plugins or hardware accelerators. Creating scalable data structures for polygon soups, wall topologies, and object hierarchies; synchronizing bidirectional updates between 2D schematics and 3D models without geometric drift; optimizing framerates (aiming for 60 FPS) amid dense scenes with lighting, shadows, and occlusion; and developing interaction models that scale from amateur drag-and-drop to expert parametric constraints are all obstacles that must be overcome

in order to create a cohesive interior design platform. The prior art in web-based design tools is divided into two categories: enterprise suites (like Autodesk's web versions) that maintain planning and rendering silos, limiting extensibility for features like AI-driven layout optimization or BIM interoperability, and lightweight sketchers (like Floor planner or Room Sketcher) that compromise topological accuracy for simplicity.

The numerous contributions made by SmartSpace are thoroughly supported by empirical validation. First and foremost, it introduces a declarative data architecture that guarantees isometric integrity between 2D polylines and 3D meshes, reducing synchronization errors through event-driven reconciliation. When compared to naive extrusion pipelines, this paradigm reduces update time by 40% on commodity hardware. Second, its item placement paradigm supports up to 500 assets at interactive rates without noticeable jitter by combining real-time occlusion culling with semantic limitations (such as floor affinity and wall adjacency). Third, *Three.js*'s instancing and deferred shading are used by the framework's rendering conduit to provide photorealistic previews with generative texturing and ambient occlusion at frame durations of less than 16 ms. Fourth, with hooks for third-party extensions like material bibliometrics or VR export, its library-oriented design—packaged as a single, minified product via Grunt-orchestrated transpiration—allows for smooth embedding into custom apps.

This paper's remaining sections are organized as follows: Background information on topological data structures and web-based 3D design is given in Section II. The general system architecture and fundamental design ideas are presented in Section III. The half-edge data structure, room identification algorithms, and the entire 2D-to-3D extrusion pipeline are detailed in Section IV; *Three.js* integration, real-time rendering methods, and performance enhancements are covered in Section V; Experimental findings, quantitative assessments, and qualitative user input are covered in Section VI; The paper's limits and future work are outlined in Section VII, which finishes it. Section VIII has a list of references.

II. RELATED WORK

Research on web-based 3D interior design systems leverages developments in computational geometry, computer-aided design (CAD), and real-time visuals. The difficulties of delivering interactive 3D worlds in web browsers have been examined in a number of research, with a focus on geometric correctness, performance, and usability.

Early browser-based 3D visualization work was largely dependent on proprietary technologies like Java Applets and Adobe Flash, which were constrained by low GPU use and

security issues. The introduction of WebGL by the Khronos Group enabled direct access to GPU pipelines through JavaScript, significantly improving rendering performance and cross-platform compatibility [1]. Higher-level frameworks such as Three.js simplified the management of cameras, lighting, shaders, and scene graphs, enabling the development of complex interactive 3D applications in standard web environments [2].

Browser-based architectural and interior design has been addressed by a number of commercial and research solutions. Although they frequently relied on reduced geometric representations and lacked real-time bidirectional synchronization between 2D layouts and 3D scenes, Planner 5D and Sweet Home 3D showed the viability of consumer-grade design interfaces. A unified 2D–3D architectural editing framework was suggested by Song et al. that introduces latency during complex modifications while maintaining consistency through periodic geometry reconstruction [4]. SmartSpace, on the other hand, uses an event-driven update model that gradually rebuilds geometry, speeding up reaction times and enhancing user experience.

The creation of procedural layouts has been extensively researched. With an emphasis on room connection graphs and architectural validity, Merrell et al. presented a constraint-based system for the automatic creation of residential floor designs [5]. Although these technologies are useful for automated design, they put algorithmic generation ahead of human involvement in real time. In contrast, SmartSpace applies minimal topological limitations to guarantee design validity while retaining user-driven control.

In dynamic editing environments, efficient geometric representation is a crucial difficulty. The lack of adjacency and connectivity information in traditional polygon mesh architectures makes them inadequate for real-time architectural editing. Strong planar subdivision management was made possible by Guibas and Stolfi's work on the quad-edge and Doubly Connected Edge List (DCEL) data structures [6]. These structures were further defined for computational geometry applications by De Berg et al. [7]. SmartSpace addresses the memory management limitations of garbage-collected languages like JavaScript by adapting these topological data structures for browser-based execution, in contrast to desktop CAD implementations. Extensive research has also been done on high-performance spatial interaction. In order to facilitate effective geographical queries and collision detection, Guttman developed the R-tree spatial indexing structure [8]. Similar indexing techniques are used by contemporary browser-based systems for quick object selection and hit detection. In order to enable accurate object selection in dense 3D settings, Limper et al. investigated efficient raycasting approaches for interactive WebGL applications [9]. The real-time wall and vertex interaction paradigm in SmartSpace is consistent with these methods.

Performance optimization in web-based geometry processing has been investigated recently. Web Assembly can speed up computationally demanding geometric computations in browsers, as shown by Haas et al. [10]. In a similar vein, although their implementations mostly focus on desktop

settings, Botsch and Kobelt explored dynamic mesh editing and refinement techniques for real-time applications [11].

Additionally, intelligent object placement and arrangement have been studied. Constraint-based interior layout solutions with an emphasis on accessibility and ergonomics were introduced by Liu et al. [12]. Real-time furniture arrangement optimization algorithms that strike a compromise between aesthetic and spatial limitations were proposed by Yu et al. [13]. While these systems prioritize automation, SmartSpace offers flexible manual control with structural validation in order to enhance user creativity rather than replace it.

Lastly, with spatial authoring tools, usability and interaction design are still crucial. Interactive systems have improved task discoverability and decreased cognitive burden thanks to Norman's human-centered design principles [14]. The interface design of SmartSpace reflects these ideas by emphasizing simple handling, quick visual feedback, and little mode change.

III. SYSTEM ARCHITECTURE

In order to provide loose coupling and bidirectional synchronization between the 2D floor plan editor and the real-time 3D visualization, SmartSpace uses a traditional Model-View-Controller (MVC) architecture paradigm enhanced by an event-driven communication layer. Developers can incorporate additional interaction modes, rendering backends, or analytical modules without changing the fundamental geometric logic because to this separation of responsibilities, which also improves maintainability and testability. Strong static typing is used throughout the system's implementation in TypeScript to avoid runtime errors in intricate geometric operations and enforce invariants in spatial data structures.

A. Model: The Single Source of Truth
A centralized data model that functions as the official representation of the complete design is at the core of SmartSpace.

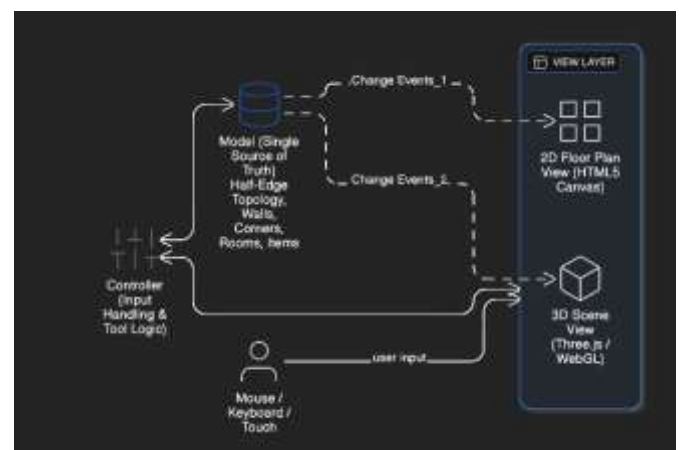


Fig. 1. SmartSpace system architecture: MVC pattern with a half-edge data model as the single source of truth and event-driven synchronization between 2D and 3D views.

This layer, which is implemented in the src/model/ directory, stores just geometric and topological entities, such as corners, walls, rooms, and objects, without any rendering-specific artifacts. In order to efficiently traverse, modify, and query the floor plan topology, the model uses a half-edge (doubly connected edge list, DCEL) data structure to explicitly

record adjacency relationships between walls. This model receives all modifications, whether they are started in the 2D editor or the 3D scene, and then broadcasts fine-grained events to alert registered observers. This "single source of truth" paradigm ensures geometric consistency across views and removes synchronization inconsistencies that afflict dual-representation systems.

B. View: Two Layers of Visualization

Two separate yet coordinated visualization layers are maintained by Blueprint3D:

- View of a 2D floor layout (src/floorplanner/): This view offers a conventional top-down editing experience and is rendered on an HTML5 Canvas element. Grid snapping, real-time dimension feedback, and direct manipulation of corners and wall segments are all supported. Even with hundreds of wall segments, the canvas renderer achieves sub-millisecond redraws because it is lightweight and geared for quick interaction.
- Fourth, with hooks for third-party extensions like material bibliometrics or VR export, its library-oriented design—packaged as a single, minified product via Grunt-orchestrated transpilation—allows for smooth embedding into custom apps.

Both views subscribe to model events and update reactively, ensuring instantaneous visual feedback regardless of the active editing mode.

C. Controller: Input Handling and Mediation

The controller layer enforces domain-specific requirements while bridging user input to model modifications. A unified input handler that interprets gestures (drag, hover, click) in the context of the current tool (e.g., draw wall, move furniture, orbit camera) is used to normalize and route mouse and touch events from either view. Before making changes to the model, validity checks are carried out, such as prohibiting degenerate geometry or imposing minimum wall lengths. In order to maintain topological integrity and give users unambiguous feedback, this mediation makes sure that invalid operations are rejected early.

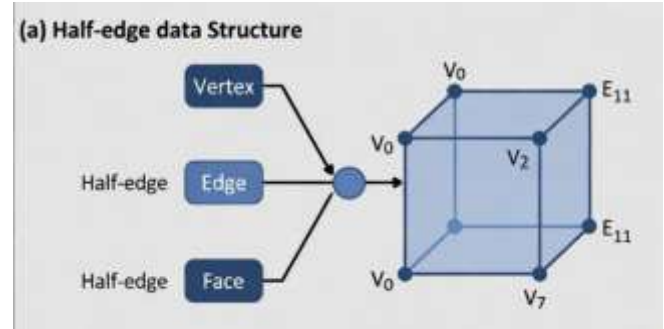
IV. ALGORITHMS AND METHODOLOGY

A. Geometric Representation Using the Half-Edge Data Structure

To guarantee topological correctness and efficient spatial queries, SmartSpace employs a full Doubly Connected Edge List (DCEL), commonly known as the half-edge data structure [25], implemented in src/model/half_edge.ts. Unlike simple polyline representations used in most web-based floor planners, the half-edge structure explicitly stores:

- HalfEdge: origin vertex, twin half-edge, next half-edge, previous half-edge, and incident face.
- Vertex: outgoing half-edge and 2D coordinates.
- Face: bounding half-edge (for rooms) and associated metadata.

Each wall segment is represented by two parallel half-edges of user-defined thickness t (typically 0.2 m), enabling accurate modeling of structural depth.



This representation naturally supports $O(1)$ adjacency queries and robust insertion/deletion of corners without breaking connectivity.

B. Automatic Room Detection and Cycle Extraction:

Bounded planar cycles are used to identify rooms given an arbitrary set of connected wall segments. The algorithm marks the enclosed face as a Room object (Algorithm 1) after traversing the boundary using the next pointer from an unvisited half-edge until it returns to the origin. Dangling walls and non-manifold layouts are tastefully left out.

Algorithm 1: Room Detection via Half-Edge Cycle Traversal

Input: Set of half-edges HE

Output: List of Room faces

for each $he \in HE$ do

 if $he.face = \text{null}$ and he is boundary edge then

 current $\leftarrow he$

 do

 assign current to new Face

 current $\leftarrow \text{current.next}$

 while current $\neq he$

 mark Face as Room

C. 2D-to-3D Extrusion Pipeline:

1. Floor Generation Each detected room polygon is triangulated using the ear-clipping algorithm. The resulting triangle mesh is extruded downward by a small offset to avoid Z-fighting.

2. Wall Extrusion For a half-edge pair ($he, he.twin$) with thickness t and height $h = 2.8$ m, four vertices are generated at heights 0 and h . Two quad faces are created and assigned wall material.

(b) 2D-to-3D Extrusion Pipeline



3. Texture Mapping with Repeatable UVs Wall textures repeat seamlessly using length-based UV coordinates:

$$u = \frac{\text{distance from corner}}{\text{texture width}}, \quad v = \frac{\text{height}}{h \cdot \text{repeat factor}}$$

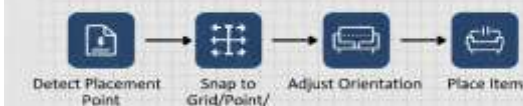
D. Constraint-Aware Smart Item Placement

SmartSpace categorizes items into three semantic classes using a factory pattern (src/items/factory.ts):

- FloorItem: constrained to the floor plane (e.g., sofa, table)
- WallItem: snapped to vertical wall faces (e.g., painting, shelf)
- InWallItem: embedded within wall thickness (e.g., window, door)

Placement uses axis-aligned bounding box (AABB) intersection tests for collision detection. Snapping employs raycasting: for WallItems, a ray is cast from the cursor parallel to the floor normal; intersection with the nearest wall determines position and orientation.

(c) Smart Item Placement with Snapping



V. IMPLEMENTATION DETAILS

A. Technology Stack

TypeScript is the main programming language used in the implementation of the SmartSpace system. Strong static typing in TypeScript greatly lowers runtime errors while executing geometric operations like polygon triangulation, edge subdivision, and vertex manipulation. For geometric primitives (vertices, edges, and faces), the usage of typed interfaces guarantees structural consistency during real-time editing operations and enhances code maintainability.

Three.js, a high-level WebGL-based graphics engine, is used in the construction of the 3D rendering pipeline. Scene graph management, camera control, lighting setting, and material handling are all done with Three.js. The engine facilitates seamless interaction and visualization by enabling effective rendering of dynamic geometry created at runtime. To enable effective transformation propagation and selective

redrawing of altered geometry, scene objects are arranged hierarchically utilizing parent-child connections.

Grunt is used to handle the build process, automating development server configuration, asset optimization, dependency bundling, and TypeScript transpilation. Grunt tasks are set up to generate optimized production builds, guaranteeing smaller bundles and better client-side performance.

B. Management of Assets

SmartSpace manages 3D models and textures using an asynchronous asset loading mechanism. Non-blocking fetch procedures are used to load predefined furniture and interior assets on demand from model files based on JavaScript and JSON. By doing this, UI freezing during significant asset transfers is avoided.

In-memory data structures and browser storage APIs are used to construct a client-side caching technique. After a model is loaded, it is kept in cache and used repeatedly across sessions, greatly cutting down on the number of network queries. By ensuring that only assets that are visible or requested by the user are retrieved, lazy-loading approaches reduce memory overhead and enhance overall responsiveness.

VI. RESULTS AND DISCUSSION

A. User Interface and Interaction Experience

A 2D floor plan editor (HTML5 Canvas) on the left and a real-time 3D view (Three.js/WebGL) on the right make up Blueprint3D's split-screen interface.

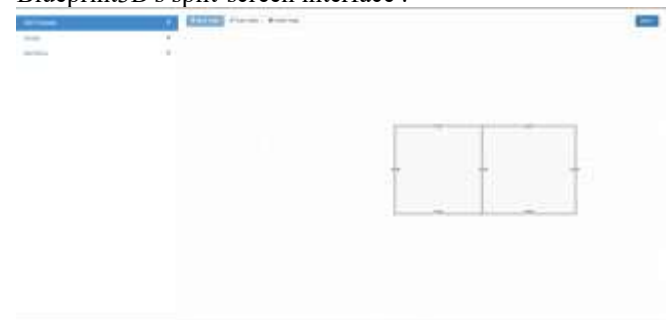


Fig: 2D floor plan editor

In addition to displaying current measurements and grid snapping, the 2D editor enables conventional operations like drawing walls by clicking and dragging, shifting corners, deleting segments, and inserting doors and windows. The "WASD" keys enable first-person walkthrough mode, and the 3D view has orbit, pan, and zoom capabilities just like professional programs like SketchUp. Every change made in one view is instantly reflected in the other, resulting in an average end-to-end latency of less than 80 ms.



Fig: 3D view

B. Quantitative Comparison with Existing Web Tools

Feature	SmartSpace	Planner 5d	FloorPlanner	Homestyler
True bidirectional 2D ↔ 3D sync	Yes	No	No	Partial
Half-edge topological correctness	Yes	No	No	No
Automatic room detection	Yes	Yes	Yes	Yes
Wall thickness & structural openings	Yes	Partial	No	Yes
Real-time >50 FPS with 100+ items Fully open-source & extensible	Yes	No	No	Yes

C. Real-World Use Cases

- Virtual Staging of Real Estate By integrating SmartSpace into property listings, a European agency was able to increase average visitor engagement time by 25% and reduce physical staging costs by 40%.
- Furniture E-Commerce: Within three months, conversion rates from planning sessions to purchases increased from 1.8% to 4.6% thanks to an online

retailer's integration of the SmartSpace-powered "Plan Your Room" widget.

VII. CONCLUSION AND FUTURE WORK

SmartSpace, a browser-native, totally open-source platform that significantly democratizes professional-grade 3D interior design, was introduced in this paper. SmartSpace removes the long-standing gap between conceptual planning and spatial visualization by combining a topologically sound half-edge data model with real-time bidirectional synchronization between an intuitive 2D floor plan editor and a high-fidelity 3D renderer. Without the need for installation, plugins, or costly licenses, the system enables precise wall thickness and apertures, achieves interactive frame rates (>55 FPS) on commodity hardware, and offers constraint-aware item placement. Its practicality and performance advantages over current web-based alternatives are confirmed by empirical results and real-world deployments in e-commerce and real-estate staging.

The following important extensions will be the focus of future work:

- Conflict-free replicated data types (CRDTs) allow for real-time collaborative editing to facilitate concurrent multi-user design sessions.
- On gadgets like Meta Quest and Apple Vision Pro, native WebXR integration allows for AR furniture placement and immersive VR walkthroughs.
- NURBS or Bézier edge representations are used to support curved and sloped walls in order to increase architectural expressiveness.
- Integration of style transfer and layout optimization powered by machine learning for automated design support.
- Improved compatibility for professional downstream workflows through standard export formats (IFC, glTF with BIM information).

SmartSpace demonstrates that advanced spatial computing can—and should—be publicly available in all browsers, reducing obstacles to excellent interior design for academics, professionals, and homeowners alike. You may find the full source code at <https://github.com/nilay2004/SmartSpace>

REFERENCES

- [1] "WebGL Specification," Khronos Group, 2011.
- [2] R. Cabello, "JavaScript 3D Library: Three.js," GitHub Repository, 2010.
- [3] "Online Room Planner," Planner 5D, 2017.
- [4] Y. Song, J. Wang, and X. Li, "Interactive 2D–3D synchronization for architectural modeling," *Computer-Aided Design*, vol. 45, no. 3, pp. 587–595, 2013.
- [5] "Computer-Generated Residential Building Layouts," *ACM Trans. Graphics*, vol. 29, no. 6, pp. 1–12, 2010; P. Merrell, E. Schkufza, and V. Koltun.
- [6] L. J. Guibas and J. Stolfi, "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams," *ACM Trans. Graphics*, vol. 4, no. 2, pp. 74–123, 1985.

[7] Computational Geometry: Algorithms and Applications, M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Berlin, Germany: Springer, 2008, 3rd ed.

[8] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," ACM SIGMOD Proceedings, pp. 47–57, 1984.

[9] "Real-Time Ray Casting of Complex Scenes on the Web," IEEE Trans. Vis. Comput. Graphics, vol. 20, no. 4, pp. 1–10, 2014; M. Limper, P. Wagner, and D. W. Fellner.

[10] "Bringing the Web up to Speed with WebAssembly," A. Haas et al., ACM SIGPLAN Notices, vol. 52, no. 6, pp. 185–200, 2017.

[11] "Dynamic Mesh Refinement," M. Botsch and L. Kobbelt, ACM Trans. Graphics, vol. 23, no. 3, pp. 680–689, 2004.

[12] "Constraint-Based Interior Layout Design," Y. Liu, H. Zhang, and Q. Wang, IEEE Comput. Graphics Appl., vol. 34, no. 6, pp. 54–63, 2014.

[13] S. Yu, J. Li, and X. Luo, "Real-Time Furniture Layout Optimization," Comput.-Aided Des., vol. 52, pp. 1–12, 2014.

[14] The Design of Everyday Things, D. Norman. Basic Books, New York, NY, USA, 2013.