

AIR3D: A Browser-Based Gesture-Controlled 3D Modeling Studio Using Real-Time Hand Tracking and WebGL

Tamim Ahmad Kadry¹, Himani Sharma², Piyush Raj³, prof. Chanchal Shori⁴

^{1,2,3}Department of CSE, SSTC, Bhilai, India

⁴Assistant Professor, SSTC, Bhilai, India

Abstract

AIR3D is a browser-based 3D modeling studio that leverages real-time hand gesture recognition via a standard webcam to enable fully touchless interaction with 3D content. Built on Three.js for WebGL rendering and MediaPipe Hands for 21-point hand landmark detection, AIR3D allows users to create 3D primitives such as boxes, spheres, cylinders, cones, tori, and prisms; manipulate objects through pinch-drag and two-hand scale or rotate gestures; draw freehand 3D lines and extrude them into surfaces; orbit and zoom the camera; and perform complete scene management including save and export. All interactions are driven purely by hand gestures captured through a standard webcam, with no plugins, no specialized hardware, and no installation required. This paper presents the system architecture including the layered z-index visual stack and shared APP state model; the gesture recognition pipeline; the Four-Pillar stabilization approach comprising Exponential Moving Average smoothing, vote buffering, hysteresis pinch thresholds, and orbit inertia; the eight independent gesture state machines; mirror compensation logic; the dwell-click activation system; and the freehand 3D Draw Line system with extrusion capability. Results demonstrate that accurate, low-latency gesture control is achievable in a pure browser environment, with broad implications for accessibility, creative tools, education, and human-computer interaction research. The entire application runs from three plain files with no build step, no bundler, and no framework, demonstrating that powerful real-time AI-driven creative tools can be delivered as lightweight, dependency-free web applications.

Keywords: Hand gesture recognition; 3D modeling; MediaPipe; Three.js; WebGL; Human-Computer Interaction; Real-time tracking; Touchless interface; Browser-based application

1. INTRODUCTION

Human-Computer Interaction (HCI) has traditionally relied on physical input devices such as keyboards, mice, and touchscreens. While effective, these peripherals create physical constraints and accessibility barriers. Gesture-based interfaces represent a compelling alternative, enabling natural, direct manipulation of digital content using the hands, which are the most fundamental human tools for interaction with the physical world.

AIR3D (Air-based 3D Studio) is a browser-native application that brings gesture-controlled 3D modeling to any device equipped with a standard webcam. By combining MediaPipe's real-time hand landmark detection with Three.js's WebGL rendering engine, AIR3D achieves a seamless overlay of a 3D modeling environment on top of a live camera feed, all running within a standard web browser with no installation, no plugins, and no specialized hardware. The system supports a comprehensive set of creative operations including creating and placing 3D primitives, grabbing and dragging objects, two-hand scaling and rotation, camera orbiting and zooming, freehand 3D line drawing, shape extrusion into surfaces, and scene save or export, all driven purely by hand gestures recognized in real time.

This paper describes the design, implementation, and technical architecture of AIR3D. Section 2 reviews related work. Section 3 describes the system architecture. Section 4 presents the gesture recognition pipeline and stabilization techniques. Section 5 details the eight gesture state machines. Section 6 covers the Draw Line system. Section 7 discusses the technology stack and browser compatibility. Section 8 presents conclusions and future directions. References are shown in square brackets.

2. LITERATURE REVIEW

Gesture recognition for HCI has been an active research area for decades. Early systems relied on instrumented gloves or depth sensors. Leap Motion [1] introduced markerless hand tracking via infrared sensors, enabling gesture-based interfaces without physical contact. However, hardware dependency limited widespread accessibility and adoption.

The advent of deep learning transformed hand tracking. MediaPipe Hands [2], proposed by Zhang et al. (2020), provides accurate real-time 21-point hand landmark detection from a single RGB camera using a lightweight CNN architecture, making webcam-based gesture interfaces practically viable without specialized hardware for the first time.

Browser-based 3D rendering has matured significantly with WebGL and Three.js [3]. Applications such as Tinkercad and Sketchfab demonstrate that professional-grade 3D interactions are achievable in the browser. However, these tools rely exclusively on traditional mouse and keyboard input rather than gesture control.

Work on gesture-controlled 3D environments has largely focused on VR and AR contexts requiring head-mounted displays or depth cameras [4]. Browser-native gesture control of 3D content using only a standard webcam remains relatively unexplored. AIR3D addresses this gap by integrating MediaPipe's monocular hand tracking with Three.js rendering to deliver a fully browser-native, hardware-agnostic 3D gesture interface. Prior work on gesture stabilization [5] highlights challenges of landmark noise and classification instability that AIR3D addresses with its Four-Pillar approach.

3. SYSTEM ARCHITECTURE

AIR3D is composed of three plain files with no build step, no bundler, and no framework.

Table 1: Architecture Overview

File	Role	Responsibility
index.html	UI Shell	Layer stack, layout, styles, shared APP state
scene.js	3D Engine	Three.js setup, objects, camera, raycasting, save/load/export
gesture.js	Gesture Engine	Webcam, MediaPipe, landmark processing, all gesture state machines

The file index.html serves as the UI shell defining the layer stack, layout, styles, and shared APP state. The file scene.js handles all Three.js setup, 3D objects, camera management, raycasting, save, load, and export. The file gesture.js manages the webcam, MediaPipe integration, landmark processing, and all gesture state machines.

3.1 Visual Layer Stack

The UI is built as a z-ordered layer stack that creates the illusion of 3D objects floating in physical space:

- z-index 1 — <video>: Live camera feed, CSS-mirrored (scaleX(-1)), dimmed to 38% opacity
- z-index 2 — Three.js <canvas>: Transparent 3D scene rendered over the camera feed

- z-index 3 — Hand skeleton <canvas>: MediaPipe landmark overlay, also CSS-mirrored
- z-index 10 — <div> panels: Sidebar, topbar, and status bar UI controls
- z-index 20 — <div> cursors: Dwell rings, toast notifications, hand cursors

The Three.js renderer uses `alpha: true` and `clearColor(0, 0)` so the background is fully transparent, allowing the camera feed to show through the 3D scene. This creates a seamless AR-like effect where virtual objects appear to inhabit the physical environment.

3.2 Shared Application State

Both `scene.js` and `gesture.js` share a global APP object defined in `index.html`. This object holds the current mode (create, edit, or draw), the selected shape, color, size, transform mode (move, rotate, or scale), and the draw plane height. Mode switching, shape selection, and color picking all modify APP, which both subsystems read reactively. This eliminates tight coupling between the rendering and gesture engines.

Both `scene.js` and `gesture.js` share a global APP object defined in `index.html`, enabling decoupled communication between the rendering and gesture subsystems:

```
APP = { mode: 'create' | 'edit' | 'draw', shape: 'box', color: '#60a5fa', sz: 1.0, tm: 'move' | 'rotate' | 'scale', drawY: 0 }  
(1)
```

Mode switching, shape selection, and color picking all modify APP, which both subsystems read reactively. This eliminates tight coupling between the rendering and gesture engines.

4. GESTURE RECOGNITION PIPELINE AND STABILIZATION

Raw MediaPipe output provides 21 normalized (x, y, z) landmark positions per hand at approximately 30 frames per second. While anatomically accurate, this data contains significant per-frame noise of approximately 5 to 8 pixel jitter, making direct gesture classification unreliable. AIR3D employs a Four-Pillar stabilization approach that stacks on top of raw MediaPipe output to solve this problem.

4.1 Pillar 1: Landmark EMA Smoothing

Every landmark position is smoothed using an Exponential Moving Average (EMA) before any gesture classification occurs:

$$\text{smoothed}[i] = \alpha * \text{raw}[i] + (1 - \alpha) * \text{prev}[i] \quad (\alpha = 0.50) \quad (2)$$

At $\alpha = 0.50$, each frame contributes equally with accumulated history. This reduces jitter from ~5-8 pixels to ~1 pixel. The hand cursor receives a second EMA pass at $\alpha = 0.40$ on top of the first, yielding a visually smooth cursor with minimal perceived latency.

4.2 Pillar 2: Gesture Vote Buffer

Instead of classifying each frame independently, the last 5 classified frames are stored in a ring buffer. A gesture registers only if it wins at least 3 of the 5 most recent frames ($VOTE_WINDOW = 5$, $VOTE_THRESH = 3$). This eliminates single-frame noise entirely. A momentary muscle twitch that produces one anomalous frame cannot disrupt an ongoing gesture state.

4.3 Pillar 3: Hysteretic Pinch Threshold

The pinch gesture uses asymmetric enter and exit thresholds: $PINCH_ON = 0.065$ normalized landmark units to enter the pinch state, and $PINCH_OFF = 0.095$ to exit. The gap between these values creates a dead zone that prevents oscillation at the boundary. Once a pinch is committed, the state remains stable until the user clearly opens their hand, eliminating accidental drop events during fine manipulation.

4.4 Pillar 4: Orbit Inertia

Camera orbit velocity is stored on gesture end and decays exponentially per frame with $ORBIT_DECAY = 0.88$ at approximately 30 fps, producing approximately 400 milliseconds of natural deceleration. The camera glides to a smooth stop rather than cutting abruptly, providing a kinematically natural feel consistent with physical inertia.

5. GESTURE STATE MACHINES

AIR3D implements eight independent gesture state machines, each with isolated state and no shared mutable variables between machines. This isolation prevents cross-contamination between gesture types and simplifies debugging.

Table 2: Mapping of Gesture State Machines

ID	Gesture	Input	Action
PD	Pinch-Drag	One-hand pinch	Drag objects, placement intent timer
TP	Two-Pinch	Both hands pinch	Two-hand scale, rotate, lift
FO	Fist-Orbit	One-hand fist	Camera orbit (wrist-anchored)
OO	Palm-Orbit	Open palm	Camera orbit (palm-centre-anchored)
BFZ	Both-Fist-Zoom	Both fists	Viewport zoom via fist separation
PH	Peace-Hold	Peace sign 1.2s	Delete object with arc progress visual
TV	Thumbs-View	Thumbs up	Cycle elevation views (top/front/side)
DL	Draw-Line	Pinch in Draw mode	Trace freehand 3D line on horizontal plane

5.1 Mirror Compensation

MediaPipe processes the un-mirrored raw camera feed and labels hands from the camera's perspective. Since the video element is CSS-flipped using $scaleX(-1)$ for natural mirror-like interaction, hand labels are inverted on input (Left becomes Right and vice versa). Landmark X coordinates are also flipped when converting to screen space using the formula $x_screen = (1 - x_raw) * window.innerWidth$, but drawn in raw space on the overlay canvas which is also CSS-flipped, ensuring full visual consistency.

5.2 Dwell Click

A ninth feature, Dwell Click, operates across all modes. When the user's pointing gesture remains over a UI button for 1.5 seconds, the button activates. A radial progress arc provides visual feedback proportional to dwell time, confirming the pending activation without requiring physical touch.

6. DRAW LINE SYSTEM

In Draw mode, pinching traces a 3D line projected onto a configurable horizontal plane at height `drawY`. An invisible Three.js mesh serves as the draw plane, into which rays from the camera are cast via the pinch screen position to determine the 3D hit point. Points are sampled only when the hand moves more than `DRAW_MIN_DIST = 0.010` normalized landmark units from the previous sample, preventing noise-generated point clusters when the hand is approximately stationary.

A live preview line is built with a pre-allocated `Float32Array` using `DynamicDrawUsage`, allowing Three.js to re-upload geometry to the GPU each frame without buffer reallocation. On pinch release, the live preview is replaced by a compact committed line. Committed lines support two operations: `Close`, which appends the start point to the end to form a closed polygon; and `Extrude`, which connects consecutive point pairs by quad triangles (two triangles per pair) to produce a 3D wall mesh, converting the 2D freehand drawing into a volumetric surface.

7. TECHNOLOGY STACK AND BROWSER COMPATIBILITY

Table 3: Technology Stack and Application Dependencies

Technology	Version	Purpose
Three.js	r128	3D scene, meshes, lights, raycasting
MediaPipe Hands	0.4	Real-time 21-point hand landmark detection
Web APIs	—	<code>getUserMedia</code> (camera), <code>localStorage</code> (save), <code>Blob/URL</code> (export)
Vanilla JavaScript	ES2020	No framework, no bundler, no build step
CSS / Google Fonts	—	Layer stack, glass panels, dwell ring, Inter + JetBrains Mono

The application uses Three.js r128 for 3D scene management, meshes, lights, and raycasting; MediaPipe Hands version 0.4 for real-time 21-point hand landmark detection; standard Web APIs including `getUserMedia` for camera access, `localStorage` for scene saving, and `Blob` and `URL` for JSON export; Vanilla JavaScript ES2020 with no framework or build tooling; and CSS with Google Fonts (Inter for UI text, JetBrains Mono for numeric labels). Chrome 90 and above and Edge 90 and above are fully supported. Firefox offers partial support with slower MediaPipe WASM performance. Safari requires HTTPS for camera access. Chrome is recommended due to optimal WASM performance in the V8 engine. The application must be served over HTTP or HTTPS as the file protocol does not permit camera access.

8. CONCLUSION

AIR3D demonstrates that accurate, responsive, gesture-controlled 3D modeling is achievable in a standard web browser using only a webcam, without plugins, specialized hardware, or installation. The Four-Pillar stabilization approach (EMA smoothing,

vote buffering, hysteretic thresholds, and orbit inertia) transforms noisy raw MediaPipe landmark data into stable gesture states suitable for creative 3D manipulation. Eight independent gesture state machines provide a rich vocabulary of natural interactions covering creation, manipulation, navigation, drawing, and scene management.

The architecture of three plain files with no build step demonstrates that powerful real-time AI-driven creative tools can be delivered as accessible, dependency-free web applications. Future work includes multi-user collaboration, persistent cloud storage, export to standard 3D formats such as OBJ and GLB, voice command integration for hybrid multimodal input, and mobile device optimization for front-facing camera environments. AIR3D contributes a concrete existence proof that natural gesture interfaces for 3D content creation are viable in the browser today, with significant implications for accessibility, inclusive design, and the future of touchless creative tools.

REFERENCES

- [1] Leap Motion Inc., “Leap Motion Controller,” 2013. [Online]. Available: <https://www.ultraleap.com>
- [2] F. Zhang et al., “MediaPipe Hands: On-device Real-time Hand Tracking,” arXiv:2006.10214, 2020.
- [3] R. Cabello et al., “Three.js — JavaScript 3D Library,” GitHub. [Online]. Available: <https://threejs.org>
- [4] D. Kim, O. Hilliges, S. Izadi et al., “Digits: Freehand 3D Interactions Anywhere Using a Wrist-Worn Gloveless Sensor,” in Proc. ACM UIST, 2012, pp. 167–176.
- [5] A. Kuznetsov and I. Panchenko, “Smoothing and Stabilization Techniques for Real-Time Gesture Recognition Systems,” Journal of Computer Vision and Image Processing, vol. 11, no. 2, 2021.
- [6] Arunthathi S, Logeshwari T, and Anuratha V, “A Research Paper on Online Job Portal System,” International Journal of New Innovations in Engineering and Technology, vol. 24, no. 1, March 2024. (Reference format adapted for structural guidance.)