

PROJECT DOCUMENTATION

3D INTERACTIVE PORTFOLIO

Syed Roshan Shareq
Project Owner and Developer

Architecture, interaction workflows, rendering, game logic, feedback, deployment and maintenance.



I built this portfolio around a desk rather than a conventional scrolling page. The scene gives each action a physical place while keeping the resume and detailed reading surfaces in HTML.

Project source snapshot. These notes describe the implementation in that folder, including its current constraints and maintenance requirements.

Contents

Project Boundary	3
Files and Dependencies	3
Startup and Device Routing	4
Scene Construction	4
Camera and Application State	5
Object Interaction and Overlays	6
Simple View and Resume Content	7
Tic-Tac-Toe	8
Feedback Submission	9
Rendering and Performance	10
Accessibility and Responsive Behavior	10
Running and Publishing	10
Maintenance Workflows	11
Troubleshooting and Verification	12
Source Navigation	13

Project Boundary

The portfolio is a static browser application. HTML supplies the interface and content, CSS supplies the overlays and responsive presentation, Three.js draws the room, and GSAP controls transitions. There is no application server, database, authentication layer, npm manifest, bundler, or automated test suite in the the source project folder.

The references to Node.js, Supabase, Sharp, machine learning, and moderation in the resume describe other projects. Those services do not power this portfolio. The Tic-Tac-Toe opponent is a local algorithm, not a machine-learning model or a remote AI service.

Feedback is the exception to the otherwise client-side workflow: submissions are sent to the hosting origin and require Netlify Forms to receive and store them. Cloudflare-related scripts are also present in the source HTML, but the snapshot contains no Cloudflare hosting configuration.

Files and Dependencies

```
project/
|-- index.html           Main document, scene canvas, overlays, metadata
|-- simple-view.html     Standalone resume and its own mini-game
|-- assets/
|   |-- css/main.css     Main interface and responsive overrides
|   |-- js/app.js        Scene, transitions, embedded resume, main game
|-- Syed_Roshan_Shareq_CV.pdf  Downloadable resume
|-- favicon.ico          Browser icon
|-- site.webmanifest      App name and standalone display metadata
|-- robots.txt           Crawl policy and sitemap location
|-- sitemap.xml           Root-page sitemap entry
|-- artifacts-desktop.png  Desktop preview artifact, not a runtime texture
`-- README.md            Original overview
```

Dependency	Version / source	Responsibility
Three.js	r128, cdnjs	Meshes, materials, perspective camera, WebGL renderer
GSAP	3.12.5, cdnjs	Camera timelines, object response, fades
Font Awesome	6.5.0, cdnjs	HTML interface icons
Inter	Google Fonts	Main and Simple View typography
Netlify Forms	Hosting service	Feedback ingestion
Cloudflare scripts	HTML references	Email decoding and analytics beacon

Three.js and GSAP are classic scripts exposing `window.THREE` and `window.gsap`. They load before `assets/js/app.js`. The code is not written as ES modules. Upgrading Three.js needs a deliberate compatibility pass; replacing r128 with a current release is not just a URL change.

The Cloudflare email-decoding path is `/cdn-cgi/scripts/5c5dd728/cloudflare-static/email-decode.min.js`. A basic local server does not provide that route. The analytics beacon is an external script with integrity metadata. Neither is the engine for the scene.

Startup and Device Routing

The main setup runs inside `window.addEventListener("load", ...)`. Before that, a global error listener logs errors, dismisses the loader, and can reveal the missing-library message if desktop mode lacks Three.js or GSAP.

The initial routing condition is:

```
const MOBILE_SIMPLE_ONLY_MEDIA =
  "(max-width: 880px), (hover: none) and (pointer: coarse)";
```

This means either a viewport at or below 880px, **or** a device with no hover and a coarse pointer, receives the Simple View path. A wide touch-only device can therefore receive it too.

Simple-only startup

1. Collect DOM references and prepare the embedded-resume loader.
2. Add `mobile-simple-mode` to the body.
3. Dismiss the loading screen and hide the canvas and desk controls from the visible interface.
4. Bind the entry button, close button, backdrop dismissal, and Escape.
5. Return before creating the Three.js scene, renderer, or main game.

Opening the resume hides the intro and assigns the decoded HTML to the iframe's `srcdoc`. Closing it returns to the mobile notice. If the mobile-only media query later stops matching, this branch reloads the page to initialize the desktop experience.

The CDN script tags still exist in the document on mobile. Skipping WebGL initialization does not mean that the browser skips downloading those libraries.

Desktop startup

1. Check for `window.THREE` and `window.gsap`; show the error panel and return if either is absent.
2. Create the scene, camera, renderer, clock, and procedural textures.
3. Build the room and desk objects synchronously.
4. Register transitions, input handlers, resize handling, and overlay controls.
5. Start the render loop and initialize the main game functions.
6. Dismiss the loader after a 750ms timeout scheduled near the end of setup.

The loader is a timed presentation state, not asset-download progress. Texture generation and scene creation happen before the timeout is scheduled.

Scene Construction

I keep the room and interaction surfaces in Three.js, but the detailed reading interfaces remain ordinary HTML. This avoids putting long resume text on a 3D texture and allows the same content to have a simpler presentation.

Geometry and groups

`createMat()` provides a `MeshStandardMaterial` helper. `mesh()` applies geometry, material, position, rotation, scale, and shadow flags. Related meshes are assembled into `THREE.Group` objects so their transforms can move together.

The major groups include the city windows, desk, chair, file, pen, globe, books, plant, handheld, paper stack, and particles. The desk is centered around a tabletop surface height of `1.1555`. That shared height is reused for object positions, camera targets, and overlay anchors.

The room is assembled from primitives and generated textures. There are no GLB/GLTF models or separate image-texture files in this source's assets directory. The globe's map is a generated visual texture, not a geographically authoritative map or an interactive geographic dataset.

Procedural textures

`makeTexture(size, draw)` creates a canvas, draws with its 2D context, and wraps it in `THREE.CanvasTexture`. Helpers generate wood grain, roughness, desk imperfections, marble veins, file artwork, certificate paper, city buildings, fog layers, contact shadows, and reflection-like overlays.

Color maps determine appearance. Roughness maps vary highlights. Bump maps create the impression of small surface detail without adding geometry. Some texture functions use `Math.random()`, so subtle visual details can differ between page loads.

Lighting and depth

The lighting combines hemisphere, ambient, directional, and point lights. The key directional light casts a `3072 x 3072` shadow map; the renderer uses `PCFSoftShadowMap`. Additional lights give the desk, windows, chair, and objects separation.

The city view is layered behind transparent glass and tint planes. Parallax layers and moving fog add depth during transitions. Floor reflections are constructed with textured planes and opacity; this is not a ray-traced reflection system or a postprocessing pipeline. `scene.fog` is explicitly `null`; the window fog is separate geometry.

Camera and Application State

The main state string starts at `intro`. `beginExperience()` changes it to `transition`, hides the intro, moves the camera, then enters `desk`. File and project transitions temporarily use `opening` before restoring the previous state. Overlay visibility is also stored in DOM classes, so the string alone is not the entire application state.

`animationBusy` prevents most interaction entry points from starting a second transition while one is running. `refreshDeskUI()` derives which object buttons should be active from the state, animation lock, overlay classes, game visibility, and globe-focus flag.

Camera shots

The `shots` object stores a camera position (`pos`) and look-at point (`look`) for the intro, desk, file, books, pen, controller, globe, and retained mobile views. `moveCameraTo()` tweens the camera position, look target, and up vector together. Updating the look target calls `camera.lookAt()` during the tween.

The normal desktop entry uses `moveDesktopCameraSmoothToDesk()` with a 1.6-second transition. `moveCameraWalkingToDesk()` and its walking-bob path remain in the source, but are not called by the current desktop entry flow.

Desktop camera settings are a 40-degree field of view, near clipping plane **0.1**, and far plane **100**. The retained narrow-screen scene branch uses a 46-degree field of view.

Desk controls

Pointer dragging updates bounded offsets rather than using OrbitControls:

Setting	Lower bound	Upper bound
Yaw	-0.42	0.42
Pitch	-0.06	0.24
Zoom offset	-0.02	0.72

`getDeskCameraPose()` applies those offsets to the hero shot. Wheel input adds `deltaY * 0.0012` to the zoom offset, then clamps it. This is a custom constrained desk camera, not unrestricted navigation through the room.

Resize updates the camera aspect, projection matrix, renderer dimensions, capped pixel ratio, and HTML target positions. The startup device routing should be tested separately from resizing: a desktop-initialized session still has retained narrow-screen scene handlers, while a page initially loaded narrow returns before scene setup.

Object Interaction and Overlays

How clicks reach the objects

This snapshot does **not** use `THREE.Raycaster`. It uses five HTML buttons in a click layer. `updateClickTargets()` projects fixed world-space anchors through the camera:

```
const point = worldPosition.clone().project(camera);
const left = (point.x * 0.5 + 0.5) * window.innerWidth;
const top = (-point.y * 0.5 + 0.5) * window.innerHeight;
```

Anchors outside the accepted projected range are moved off-screen. This runs each rendered frame so the controls follow the camera. Moving a mesh does not automatically change its anchor; both positions need to be maintained together.

Object	HTML target	Handler	Outcome
Portfolio file	<code>fileTarget</code>	<code>openPortfolio()</code>	Resume and project book overlay
Books	<code>projectsTarget</code>	<code>openProjectsRedirect()</code>	GitHub / LinkedIn redirect overlay
Handheld	<code>controllerTarget</code>	<code>animateToControllerAndOpenGame()</code>	Main Tic-Tac-Toe overlay
Pen	<code>penTarget</code>	<code>animateToPenAndOpenFeedback()</code>	Feedback form
Globe	<code>globeTarget</code>	<code>openGlobeView()</code>	Focused, draggable globe

The file's project page contains descriptions. The books do not fetch GitHub repositories or render a live project feed. Their buttons call `window.location.assign()` with the configured public profile URLs, navigating the current tab.

Overlay lifecycle

The file and books animate toward their close-up shot, fade the transition shade, activate the overlay, set `aria-hidden="false"`, and lock body scrolling. On desktop, the camera is resynchronized to the desk pose behind the overlay. Close handlers remove the active class, restore `aria-hidden`, and refresh the desk controls.

The feedback path focuses its textarea after a short delay. Simple View loads lazily. Most reading overlays close via their button, a click on the backdrop, or Escape. The main game's close element is a clickable `div`, and the general Escape handler does not close it.

Globe focus

Opening the globe moves the camera to its focus shot and activates a blur layer and translucent backdrop. The camera is then locked to that shot each frame. Pointer capture tracks horizontal drag; normalized movement is multiplied by `Math.PI * 2.35` and added to `dragYaw`. Automatic rotation and user rotation are combined.

Double-clicking the globe target or the canvas while focused returns to the desk. The globe is an exploration detail rather than a navigation destination.

Labels and exploration completion

`setUiVisibility()` updates the labels class and toggle's pressed state. Hiding labels also increases some scene glow effects in the render loop.

The closing-note arrow is unlocked by in-memory flags for file, pen, successful feedback, gamepad, Simple View, Easy, and Hard. Difficulty flags are set on an accepted player move, not on a match win or completion. The books action also marks file. Globe interaction is not required.

The arrow additionally requires desk state, an active HUD, no animation lock, and the relevant overlays to be closed. Feedback submission failure prevents the required feedback flag from being set. Restart clears exploration flags and restores the intro camera. It is not a complete reconstruction of the runtime, and it does not recreate textures or reset the embedded iframe.

Simple View and Resume Content

There are three separately maintained resume surfaces:

1. The book pages inside `index.html`.
2. The standalone `simple-view.html`.
3. The base64-encoded HTML constant `SIMPLE_VIEW_HTML_B64` inside `app.js`.

The in-app iframe uses the third surface. `ensureSimpleViewLoaded()` calls `atob()`, converts the binary string to a `Uint8Array`, decodes UTF-8 with `TextDecoder`, and assigns `simpleFrame.srcdoc`. `simpleLoaded` makes this happen only once per page session. It does not fetch `simple-view.html`.

In the inspected snapshot, decoding the embedded payload produces exactly the same UTF-8 content as the standalone document: both are 65,054 bytes. They are synchronized here, but there is no generator or synchronization command in the source folder to keep them aligned after edits.

The standalone document includes its own CSS, scripts, expandable resume sections, Expand / Collapse controls, a PDF link, and a Quick Break game. Its game has independent board state inside the iframe when embedded; it does not contribute to the outer desk's Easy / Hard exploration flags.

The standalone PDF link is [./Syed_Roshan_Shareq_CV.pdf](#). When equivalent HTML is used through `srcdoc`, relative links normally resolve against the embedding document unless that HTML supplies a different base URL. Keep the PDF at the expected base path when moving the experience.

The source resume identifies me as a student and includes older education figures. The newer Overhaul portfolio has updated graduate information. This documentation preserves the technical behavior of the source project rather than treating its older resume as the current personal record. The source itself tells readers to visit the official site for updated resume details.

Contact details visually obscured by CSS remain in the delivered HTML. The lock-style panel is presentation, not access control. Any detail that should not be public must be removed from the markup and embedded copy.

Tic-Tac-Toe

The main game is defined near the end of the desktop load handler. Functions are exposed on `window` because the HTML uses inline handlers such as `closeGame()` and `resetGame()`. Simple View carries a separate inline implementation of the same game pattern.

State and move sequence

`board` is a nine-element array containing empty strings, `X`, or `O`. The visitor always plays `X`; the computer plays `O`. `gameOver` stops input after a terminal result, and `isAITurn` blocks additional visitor moves while the opponent is pending. `current` is updated as turn bookkeeping; the move functions write explicit symbols.

1. Reject an occupied cell, a finished game, or input during the AI turn.
2. In the main game, mark the selected difficulty as explored.
3. Disable the difficulty select and write `X` to the board.
4. Render and check the eight possible winning lines, then check for a draw.
5. If still active, lock input and schedule `aiMove()` after 600ms.
6. Choose and write `O`, render, evaluate the result, and unlock input if play continues.

`render()` rebuilds the board's nine clickable `div` cells from state. Opening the game or choosing Restart clears the board, reenables difficulty selection, hides the result popup, and resets the turn flags.

Easy and Hard

Easy chooses a uniformly random index from empty cells. Hard tries each available `O` move and runs minimax over the remaining game tree. Each tentative move is undone after recursion so the search does not permanently alter the live board.

Terminal O win: 10 - depth
 Terminal X win: depth - 10
 Terminal draw: 0

O turn: maximize the score
 X turn: minimize the score

Depth scoring favors earlier computer wins and later computer losses. The search has no alpha-beta pruning; the 3x3 board is small enough for exhaustive recursion. Equal scores retain the first candidate found because the comparison is strictly `score > bestScore`.

Result timing and reset limitation

A terminal result schedules the popup after 800ms. The popup remains for 1800ms and then resets the board. These timers, including the pending AI timer, are not stored or canceled by `resetGame()` or `closeGame()`.

That matters when restarting within the 600ms AI delay: the old callback can write an `O` into the newly cleared board. Old result callbacks can also affect a restarted match. This is an existing limitation of the snapshot. A future fix should retain timer IDs, cancel them on reset/close, or guard callbacks with a match-generation token in both game copies.

Feedback Submission

The hidden form near the start of `index.html` declares `portfolio-feedback` and its fields for Netlify's build-time form detection. The visible form uses the same name, a required message capped at 1400 characters, and an optional name capped at 70 characters.

`submitFeedback()` prevents native submission, trims the message and name, rejects an empty message, and substitutes `Anonymous visitor` for a blank name. It builds a URL-encoded payload containing:

Field	Value
<code>form-name</code>	<code>portfolio-feedback</code>
<code>name</code>	Trimmed name or anonymous fallback
<code>message</code>	Trimmed feedback
<code>page</code>	<code>window.location.href</code>
<code>createdAt</code>	Browser-generated ISO timestamp

The request is `POST /` with `Content-Type: application/x-www-form-urlencoded`. The send button is disabled during the request and reenabled in `finally`. An HTTP success marks feedback as explored, displays confirmation, shows a toast, and resets the form. Network or HTTP failure displays an error without marking completion.

The HTML declares `netlify-honeypot="bot-field"`, but the JavaScript constructs its payload manually and does not append `bot-field`. Do not assume the submitted payload includes every input in the visible form. The timestamp is client supplied, and a successful HTTP response is the client's confirmation, not independent proof that an entry appeared in Netlify's dashboard.

There is no local persistence, email-sending code, or custom feedback API in this folder. A static server can serve the interface but generally cannot receive this POST. Verify real delivery on the intended Netlify deployment, including dashboard receipt.

Rendering and Performance

`WebGLRenderer` enables antialiasing and alpha, requests the high-performance GPU preference, and caps the device pixel ratio at 2. The cap limits rendering cost on high-density screens; it does not guarantee a particular frame rate.

`animate()` reads time, updates hover transforms and material emission, animates glows, lighting, fog, city layers, globe rotation, chair details, and particles, then updates the camera and projected buttons. It renders the scene and schedules itself with `requestAnimationFrame()`.

The room continues rendering behind open overlays. There is no explicit visibility-based pause, frame-rate governor, adaptive graphics quality, renderer disposal lifecycle, or WebGL context-restoration handler in this snapshot. Browsers may throttle background tabs, but that is browser behavior rather than project scheduling.

The scene trades external model downloads for startup work: canvas textures are generated and uploaded, many primitive meshes are created, and the shadow map consumes GPU resources. If initialization or sustained rendering is slow, profile those separately. Lowering pixel ratio, shadow resolution, texture dimensions, and mesh counts targets different costs.

The render loop calls `clock.getElapsedTime()` before `clock.getDelta()`. In Three.js r128, elapsed-time retrieval updates the clock internally; the second delta can therefore be much smaller than the full frame interval. Time-based globe and particle movement should be reviewed with that behavior in mind.

Accessibility and Responsive Behavior

The object targets are actual buttons with accessible names, and keyboard focus shares the visual hover response. Several modals expose `aria-hidden`, some provide dialog roles, and status messages use live regions. Simple View gives a conventional reading route that avoids requiring WebGL interaction.

These are useful foundations, not a claim of complete accessibility compliance. The snapshot has no consistent focus trap, opener-focus restoration, or background `inert` handling. The game's cell and close controls use clickable `div` elements rather than keyboard-native buttons. The focused globe relies on double-click to exit, and the general Escape handler does not handle globe focus or the main game.

`prefers-reduced-motion` in `main.css` shortens CSS animations and transitions. It does not disable GSAP timelines or the JavaScript render-loop motion. Mobile simple-only routing is independent of the reduced-motion preference.

The stylesheet contains extensive successive overrides, primarily around 880px, 900px, 700px, and 520px. Later rules can change earlier component styling. Read the cascade through to the end before changing a selector. The current default mobile entry is Simple View; retained touch-swiping and narrow-screen 3D camera code should not be advertised as the normal initial mobile experience.

Running and Publishing

Running locally

There is no install or build step for the source. Serve its folder with any static HTTP server. For example, with Node.js and npm available:

```
npx --yes http-server "." -p 8080 -c-1
```

Open <http://localhost:8080/> for the main experience and <http://localhost:8080/simple-view.html> for the standalone resume. The example server package is a development convenience, not a dependency shipped by the source project. Choose another port if 8080 is occupied.

Direct file opening can display the project, but HTTP is a better environment for checking paths and requests. CDN fonts, icons, Three.js, and GSAP require internet access unless replaced with local copies. Feedback requires a receiving host and cannot be meaningfully verified from <file://>.

Static deployment

Publish the source folder as a static site with [index.html](#) at the web root. Keep [assets/](#), the standalone resume, PDF, icon, manifest, crawl files, and their relative locations. Enable Netlify Forms detection if feedback is expected to work.

There is no [netlify.toml](#) in the source project. Its hosting behavior must be configured outside this snapshot. Check the form name and received fields on the deployed site. A host other than Netlify needs its own submission integration.

The source canonical URL, Open Graph URL, structured data, robots sitemap URL, and sitemap root point to <https://roshanshareq.online/>. The newer portfolio links to the 3D demo at <https://interactive-three-d.netlify.app/>. If that demo is the standalone production home, update metadata consistently rather than leaving it to identify a different site's root.

The manifest supplies name, colors, start URL, and standalone display mode. It has no icon list, and the project has no service worker or offline cache. It should not be described as an offline-capable PWA.

Maintenance Workflows

Update resume content

Edit the book markup and standalone resume, replace the PDF when required, then regenerate the embedded UTF-8/base64 resume payload. Check both [simple-view.html](#) and the iframe opened by the main page. Check the PDF from both entry points. Updating one copy is not enough.

Change a camera transition

Find the appropriate [shots](#) entry and the handler using it. Adjust position and look target together, then check that the return path restores the user's desk pose. For the default entry, edit the smooth-desk path rather than the unused walking helper. Check timing against [animationBusy](#) and button activation.

Move or add an interactive object

Change the Three.js group's placement, its anchor in [updateClickTargets\(\)](#), and its close-up camera shot as one operation. A new object needs a labeled button, an action handler, visibility rules in [refreshDeskUI\(\)](#), and hover/focus mapping. Adding a mesh alone does not make it interactive.

Change the game

Maintain both the main game's functions and the standalone/embedded game's implementation. Reset and close behavior should invalidate pending callbacks. Test terminal boards, difficulty locking, a restart while AI is pending, and reopening during a result popup.

Change feedback

Keep the hidden Netlify declaration, visible input names, and manually assembled request payload aligned. If adding a field or modifying spam protection, verify the transmitted request and the received entry. The browser's success message alone is not the end-to-end test.

Troubleshooting and Verification

Symptom	Where I would start
Missing-library panel on desktop	Network requests for Three.js and GSAP; CDN access and console errors
Loader dismisses but canvas is blank	WebGL support, renderer-construction errors, shader errors, GPU context
Mobile shows only a notice and resume	Expected routing; inspect the 880px/coarse-pointer media query
Resume edit is absent in the modal	Embedded base64 copy in <code>app.js</code> , not only <code>simple-view.html</code>
Button no longer lines up with its object	Fixed world-space anchor in <code>updateClickTargets()</code>
Feedback reports a technical failure	Hosting origin, POST response, Netlify Forms configuration
Closing-note arrow never appears	Required flags, especially successful feedback and a move in each difficulty
An O appears after restarting	Uncanceled pending AI timeout
Styles seem to ignore an edit	Later selectors and media-query overrides in <code>main.css</code>
Local Cloudflare script returns 404	The local server does not implement <code>/cdn-cgi/</code>

For a release check, cover desktop entry, drag and wheel bounds, all five object actions, label toggle, each close path, PDF download, both game difficulties, and a real feedback submission. Reload at a narrow viewport and on a coarse-pointer device to test startup routing. Resize a desktop session separately. Check keyboard navigation and reduced-motion behavior without assuming CSS changes also stop WebGL motion.

These checks describe the expected verification process; they are not a claim that the original source has an automated test suite or that production feedback has been submitted during documentation work.

Source Navigation

Line references below refer to the inspected the source project snapshot and may shift after edits.

Location	What to read
index.html:77	Hidden Netlify form declaration
index.html:196	Projected HTML object controls
index.html:281	Visible feedback form
index.html:308	Resume/project book
index.html:544	Simple View iframe
index.html:556	Main game markup
index.html:579	CDN and Cloudflare script references
assets/js/app.js:1	Initial device-routing query
assets/js/app.js:18	Embedded Simple View HTML
assets/js/app.js:148	Lazy iframe decoding
assets/js/app.js:172	Simple-only startup and early return
assets/js/app.js:224	Scene/camera/renderer setup
assets/js/app.js:283	Canvas texture helper
assets/js/app.js:1173	Material definitions and room construction
assets/js/app.js:1658	Core lighting
assets/js/app.js:1844	Desk surface and object construction
assets/js/app.js:2562	Camera shots
assets/js/app.js:2756	Closing-note visibility
assets/js/app.js:2775	Projection from world position to HTML position
assets/js/app.js:2823	Desk target visibility rules
assets/js/app.js:2929	Entry transition
assets/js/app.js:3000	Feedback request
assets/js/app.js:3126	File opening sequence
assets/js/app.js:3232	Experience restart
assets/js/app.js:3317	Globe focus
assets/js/app.js:3522	Globe and desk input
assets/js/app.js:3628	Continuous render loop
assets/js/app.js:3791	Main game state and functions
assets/css/main.css:1949	CSS reduced-motion rule
simple-view.html:896	Standalone PDF link
simple-view.html:1169	Resume expansion controls
simple-view.html:1212	Independent game state and minimax