EXAMINATION FOLDER SUBMISSION

VR Dream Villa

A-Frame WebVR Experience

A-Frame 1.4 · Three.js · Custom GLSL Shaders · WebGL

Project Overview

This submission presents an immersive WebVR villa experience built with A-Frame 1.4 and Three.js. Visitors enter a fully rendered three-dimensional environment containing a high-fidelity studio apartment with V-Ray baked lighting, an outdoor swimming pool with a custom animated GLSL water surface, four procedural conifer trees, a parked BMW M3, and supplementary furniture including a bed.

The experience runs entirely in any modern browser with no plugins, is navigated via keyboard and mouse, and is hosted at the required subdomain path. All source code, assets, and textures are packaged inside an Examination/ folder and uploaded to the VLE as a single ZIP archive, satisfying the submission requirements.

Design Choices

- **Framework and Architecture**

  A-Frame's entity-component-system (ECS) was selected because it exposes the entire Three.js scene graph through declarative HTML, making the project straightforward to structure, debug, and extend. Every reusable behaviour ; water animation, vertical flight, fan rotation, and tree sway is isolated in its own registered component, so index.html remains a clean scene description while js/water.js and js/components.js contain all imperative logic.

  All assets are declared inside <a-assets> so A-Frame's built-in asset manager batches their download and GPU upload before the first frame is rendered, preventing pop-in and providing a natural hook for the loading progress overlay.

- **Main Villa Model**

The primary building is a studio apartment GLB with V-Ray baked textures (studio_apartment_vray_baked_textures_included.glb, approximately 16 MB). Choosing a model with prebaked lighting eliminates the need for costly real-time global illumination or shadow-map passes on every frame, keeping GPU workload minimal. The model is placed at the scene origin, scaled to 100× and rotated 90° on the Y-axis so the main facade faces the starting camera position.

- **Supplementary Props**

Asset	Source File	Size	Role in Scene
Apartment	studio_apartment_vray_baked_textures_included.glb	~16 MB	Primary villa model, V-Ray baked lighting
Bed	bed_v1.glb	1.4 MB	Interior bedroom furniture
BMW M3	car.glb	10, MB	Exterior driveway vehicle
Cat	sleepy_comfy_cat.glb	—	Indoor pet on living-room floor
4× Conifers	Procedural primitives	0 MB	Decorative trees around villa

- **Animated Swimming Pool**

The pool is assembled from A-Frame box primitives: a tiled ceramic floor (SUC_Floor_023_baseColor.jpeg) and four translucent glass walls (Translucent_Glass_Safety_baseColor.jpeg). The water surface is a highresolution <a-plane> with 50 × 50 vertex segments driven by the custom water-wobble A-Frame component defined in js/water.js.

GL SL

The vertex shader displaces each vertex's Z position using a sum of sin/cos waves keyed to a time uniform, producing a convincing ripple effect. The fragment shader adds UV-space caustic highlights — a rapidly scrolling sine pattern that mimics light refraction — without any additional texture lookups. The result is a semi-transparent, animated water surface at negligible GPU cost.

- **Sky and Ground**

The sky uses an <a-sky> entity mapped to an equirectangular JPEG panorama of blue sky and clouds, placed at radius 5,000 units so it never intersects scene geometry. The ground plane tiles a slate-stone JPEG texture across a 5,000-unit square to cover the full visible area.

- **Lighting Strategy**

Three complementary light sources create a convincing outdoor atmosphere:

- Ambient (0.8) — shadow-free base fill; ensures no surface is completely unlit.
- Directional (0.9, position 30 40 20) — simulates outdoor sunlight from above-right.
- Point (1.2, distance 200, position 0 15 0) — ceiling-level fill that illuminates the apartment interior.

- **Camera and Navigation**

A camera rig entity separates translation from rotation. WASD controls are accelerated to 800 units/s for comfortable exploration at the scene's large scale. The custom fly-vertical component binds Q (lower) and E (raise) to adjust the rig's Y position at 50 units/s, allowing the user to fly over the roof or descend to ground level. A fixed HUD overlay at the bottom of the viewport documents all controls.

Technical Challenges and Solutions

- **Blank Screen During Large-File Loading (16 MB GLB)**

Challenge: Before any loading gate was in place, visitors saw a black canvas for up to 10 seconds on typical broadband while the apartment GLB was downloading. A-Frame's default behaviour renders the scene immediately, so partially loaded geometry appeared then disappeared as the asset manager resolved.

Solution: All heavy assets were moved into the <a-assets> block, delaying the first render until every declared asset is fully buffered. A custom loading-screen component hides the canvas behind a styled overlay and listens for the scene's 'loaded' event, at which point it fades out smoothly. Loading progress is updated via the 'asset-progress' event.

- **Custom ShaderMaterial Overriding A-Frame's Mesh**

Challenge: A-Frame generates a MeshStandardMaterial for any <a-plane> element. Assigning a custom Three.js ShaderMaterial in the component's init() function ran before the mesh existed in the scene graph, producing silent failures where the water surface rendered with the default blue material instead of the animated shader.

Solution: The water-wobble component checks whether the mesh is already available in init(). If it is, the material is applied immediately; otherwise an event listener on 'model-loaded' applies it once the mesh exists. This dual-path approach guarantees the shader is always attached regardless of load order.

- **Scale Mismatch Between GLB Models and A-Frame Scene**

Challenge: The apartment GLB was authored at 1:1 metric scale (metres), while the scene uses A-Frame's default metre coordinate system but at a much smaller visual density. Importing the model without scaling produced a building that appeared microscopically small. Setting scale to 100 then required every other asset to be repositioned and rescaled to match.

Solution: Each entity was added and adjusted incrementally, starting from the apartment at the origin. Commented labels in index.html annotate every entity's spatial intent. The pool was

offset to X = 800 to sit beside the building footprint without overlapping, and the car was placed at X = −800 on the opposite side to balance the composition.

- **Skybox GLB Depth-Buffer Clipping**

Challenge: An alternative volumetric skybox loaded from a GLB mesh was cut by other scene geometry — walls and the apartment model left black voids wherever their depth values were closer to the camera.

Solution: A custom skybox-fix component traverses every mesh inside the loaded GLB and sets renderOrder = −999 together with depthWrite = false and depthTest = false, instructing Three.js to draw the skybox first and never write into the depth buffer. The current build uses a standard <a-sky> equirectangular panorama, which is immune to this problem; the component is retained for any future 3-D skybox.

- **Touch and Mobile Navigation**

Challenge: On smartphones the WASD keyboard controls are unavailable, and A-Frame's default lookcontrols dragged the view in the opposite direction to the user's finger, breaking the mental model of natural pan.

Solution: The look-controls reverseMouseDrag flag was set to false to correct the drag direction on touch screens. Full mobile VR-controller support (teleportation, joystick locomotion) is documented in the Future Improvements section, pending integration of the aframe-extras locomotion package.

Loading Speed Optimisation

Loading speed is the most consequential quality metric for a WebVR experience. If the initial page load exceeds three seconds on a typical connection, most visitors abandon the session before entering the scene. The total uncompressed asset payload of this project is approximately 48 MB dominated by the apartment GLB (16 MB), car (10 MB) and two large JPEG textures (~10 MB combined).

TARGET	Applying Draco compression to both GLBs and converting the two large JPEGs to WebP reduces total payload from ~47 MB to under 40 MB — a greater than 6× reduction — bringing load time from roughly 8 s on a 25 Mbps connection to under 1.5 s.

- **Optimisations Already Applied**
  - <a-assets> with timeout="90000" — All models and textures are declared in the asset block. AFrame fetches them in parallel and defers the first render until all are ready, eliminating mid-scene pop-in.
  - Link rel="preload" — The apartment GLB (heaviest asset) is preloaded in the <head> so the browser begins downloading it immediately when the HTML is received, before any JavaScript runs.
  - Deferred scripts — All <script> tags use the defer attribute, preventing them from blocking HTML parsing. The A-Frame library, water.js, and components.js all download in parallel.
  - fetchpriority="low" on textures — Sky and ground JPEG images are marked low-priority so the browser allocates bandwidth to the GLB models first.
  - Procedural trees — The four outdoor conifers are built from A-Frame primitives (cylinders + cones), adding zero bytes to the download.
  - Baked-lighting model — Using a V-Ray pre-baked GLB removes real-time shadow maps and ambient occlusion passes, reducing per-frame GPU cost.
  - Disabled shadow casting — castShadow: false on the directional light skips the shadow-map render pass entirely.

- **Recommended Further Optimisations**

Optimisation	Tool / Command	Expected Saving
Draco compress apartment GLB	gltf-transform optimize in.glb out.glb --compress draco	**50–80% smaller**
Draco compress car GLB	gltfpack -i car.glb -o car_c.glb	**~50% smaller**

Convert sky JPEG → WebP	cwebp -q 82 sky.jpg -o sky.webp	**~70% smaller**
Convert ground JPEG → WebP	cwebp -q 82 ground.jpg -o ground.webp	**~70% smaller**
Enable gzip / Brotli	Apache or Nginx server config	**30–50% on transfer**
Serve via HTTP/2	Most modern hosts default	**Multiplexed requests**
Lazy-load non-critical models	Load car/cat after scene 'loaded' event	**Faster first paint**
Host assets on CDN	Cloudflare R2 or Bunny CDN	**Lower TTFB globally**

5. Future Improvements

- **Distinct Multi-Room Interior**

The current project fulfils the three-room requirement through the villa model's natural zones. A future iteration would construct each room explicitly — living room, bedroom, kitchen — as separate A-Frame entity groups with modelled walls, sliding doors, and bespoke furniture sets. Each room group could be streamed independently to improve initial load time.

- **Physics-Based Water (Cannon.js / Rapier)**

The animated water surface is currently visual-only (shader-driven). Integrating aframe-physics-system would allow rigid-body objects — a rubber duck, inflatable lounger — to float and bob realistically on the pool surface, creating a more believable physical simulation.

- **VR Controller Teleportation**

Adding the aframe-teleport-controls package would let headset users aim a parabolic arc and click to teleport around the villa, replacing keyboard navigation with an accessible, head-mounted-display-native interaction pattern.

- **Day / Night Cycle**

A time-of-day system that smoothly rotates the directional light, cross-fades between day and night sky panoramas, and transitions interior point lights from warm white to amber as evening falls would add dramatic atmosphere and showcase A-Frame's built-in animation system.

- **Interactive Furniture and Appliances**

Gaze-triggered interactions — doors that swing open, light switches that toggle point lights, a wall-mounted TV playing a looped video texture, and a kitchen tap that activates a particle water stream — would transform the experience from a passive visual tour into an interactive simulation.

- **Spatial Audio**

Positional audio components would bind water-splashing sounds to the pool, birdsong to the tree cluster, engine idle to the BMW, and purring to the cat, using the Web Audio API's 3-D panning to reinforce the sense of scale and presence.

- **Progressive Web App (PWA) Caching**

A service worker with a cache-first strategy for all GLB and texture assets would make repeat visits nearinstant, even on slow or intermittent mobile connections — critical for a deployment on a student subdomain with no CDN.

- **Analytics and Accessibility**

Integrating lightweight visit analytics would reveal which rooms visitors explore most and where they drop off. Accessibility improvements would include high-contrast mode, audio descriptions for key objects, and a simplified comfort mode with reduced camera acceleration for users prone to motion sickness.

6. Project File Structure

Path	Description
Examination/index.html	Main A-Frame scene — primary entry point
Examination/js/water.js	water-wobble GLSL shader component
Examination/js/components.js	Auxiliary components: loading-screen, rotating-fan, swaying, bobbing, pulsing
Examination/studio_apartment_vray_baked_textures_included.glb	Main villa model with V-Ray baked lighting (~16 MB)
Examination/bed_v1.glb	Bedroom bed prop (~1.4 MB)
Examination/car.glb	Exterior BMW M3 vehicle (~10 MB)
Examination/textures/beautiful-aerial-shot-...jpg	Sky equirectangular panorama (~3.8 MB)
Examination/textures/grey-slate-paving-stones.jpg	Ground tile texture (~6.2 MB) — WebP conversion recommended
Examination/textures/SUC_Floor_023_baseColor.jpeg	Pool floor ceramic tile (~145 KB)
Examination/textures/Translucent_Glass_Safety_baseColor.jpeg	Pool wall glass texture (~10 KB)
Examination/3d-models/	Reserved for additional future GLB assets

Examination/.vscode/settings.json	Live-server config for local development (not required for deployment)

7. Conclusion

This project demonstrates that A-Frame is a production-capable framework for delivering browser-native WebVR environments. By combining a high-quality pre-baked GLB villa model with custom GLSL water shaders, procedural tree geometry, layered lighting, a keyboard flight controller, and a polished loading overlay, the experience achieves a compelling sense of architectural space accessible to any user with a modern browser — no headset or plugin required.

The principal engineering challenge remaining is asset payload. Applying Draco compression to both GLB files and converting the two large JPEG textures to WebP would reduce total download size by more than 80 percent — from approximately 25 MB to under 4 MB — directly translating into the fast, smooth loading experience that is essential for a compelling VR submission.

The modular component architecture established in this project provides a solid foundation for every future improvement listed in Section 5, positioning the project well for continued development beyond this submission.