# Integration Plan for Liquid Light Show Aesthetics in NFA Bears App

This plan outlines a **unified approach** to integrate high-impact, mobile-friendly *liquid light show* visuals into the NFA Bears application. We will consolidate multiple existing visual systems under a single orchestrated framework, prioritizing an authentic, psychedelic aesthetic ("dope as fuck") while maintaining stability on mobile devices. The strategy is informed by internal research findings that identified fragmentation and duplication in past attempts[[1]](file://file_000000004db46209a238eb3bab210df7#:~:text=1.%20%2A%2AIntegration%20Fragmentation%2A%2A%20,palettes%20and%20timing%20across%20layers)[[2]](file://file_000000004db46209a238eb3bab210df7#:~:text=4.%20%2A%2ACode%20Duplication%2A%2A%20,code%20paths%20with%20slight%20variations), and concludes that our current WebGL fluid implementation is a solid foundation, with other effects layered as enhancements[[3]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,or%20gated%20as%20optional%20enhancements). Key objectives include device-based feature gating, unified audio-reactive and color palette control, an option for a simplified “pure mode,” and a phased roadmap to implement, optimize, and culturally refine the experience.

## Consolidating Existing Visual Systems

We will **orchestrate all proven visual implementations** into a cohesive, layered system rather than separate, competing components. The following modules from the codebase and research will be unified:

* **WebGL Fluid Simulation (Baseline):** The webgl-fluid-enhanced engine (via LiquidLightBackground.tsx) serves as the core background effect[[3]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,or%20gated%20as%20optional%20enhancements). This GPU fluid simulation already delivers organic, lava-lamp-like motion at high framerates on mobile, with adaptive quality scaling and a clear fallback path[[4]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AStrengths%2A%2A%3A%20,motion%20creates%20authentic%20organic%20behavior). It provides ~90% of the classic liquid light show aesthetic with minimal complexity[[5]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,of%20the%20complexity). We will use this as the always-on foundation for all devices.
* **R3F Thin-Film Shader (Optical Overlay):** A React-Three-Fiber fragment shader adds authentic thin-film interference patterns (iridescent oil slick colors) on top of the fluid layer. This shader produces stunning, scientifically accurate 1960s-style visuals[[6]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,mode%20palettes%20provide%20cultural%20depth). However, it is heavy on fragment computations and incurs Three.js overhead, so **we will enable it only on high-tier devices** that can handle the load[[7]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AWeaknesses%2A%2A%3A%20,tier%20devices%20only). On capable hardware, this layer will run at reduced resolution and opacity to minimize performance impact while adding a colorful sheen.
* **WebGPU Particle System (Future Ultra-Mode):** An experimental WebGPU-based particle engine can create massive particle-based liquid effects (e.g. 100k+ particles with kaleidoscopic patterns) for an ultimate visual boost[[8]](file://file_0000000087f4620994e91fc9874a6455#:~:text=variants.%20,film%20needs%20direction%20when%20composed). Given WebGPU’s limited browser support in 2025, this will remain **an optional enhancement for the highest tier**, not part of the default experience until support broadens. It can be conditionally activated for forward-looking desktop environments to wow users, but will gracefully fall back to WebGL/CSS on unsupported platforms[[9]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Diminishing%20aesthetic%20returns)[[10]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=4.%20%2A%2AWebGPU%20Universal%20Deployment%2A%2A%20,support%20limited%20to%20Qualcomm%2FARM%20GPUs).
* **CSS Ambience Layers (Fallback & Depth):** A multi-layer CSS background (using conic/radial gradients, blur filters, and subtle textures) will run **at all times** behind the canvas elements[[11]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,layers%20for%20cohesion%3B%20intensity%20governance). This provides a zero-GPU ambient glow that enhances the visual richness and serves as a **safety net** if WebGL/WebGPU contexts are lost or disabled. The CSS layers are lightweight and include accessibility considerations (e.g. respecting prefers-reduced-motion by switching to static gradients)[[12]](file://file_000000004db46209a238eb3bab210df7#:~:text=,ambience%20requiring%20no%20GPU%20context)[[13]](file://file_000000004db46209a238eb3bab210df7#:~:text=,conversion%20matches%20real%201960s%20techniques). They ensure there's always some psychedelic backdrop, even on low-end or when other layers are turned off.

By **combining these systems**, we avoid the mistake of running multiple isolated effects that conflict or duplicate effort. Instead, a single orchestrator will manage these layers in concert, so they complement each other: the fluid sim provides dynamic motion, the thin-film shader adds color shimmer on capable devices, particles add extra sparkle on top-tier systems, and CSS ties everything together aesthetically and as fallback.

## Visual Impact vs. Performance Strategy

Our design philosophy is to **maximize visual authenticity and impact**, while using smart constraints to preserve mobile stability. In practice, this means we favor rich psychedelic visuals that capture the *essence* of 1960s liquid light shows over ultra-precise physics or overly conservative performance settings[[14]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Stokes%20equations). The internal analysis affirmed that *perfect Navier-Stokes simulation isn't necessary* – it's more important to achieve the swirling, vibrant look and musical responsiveness at 60 FPS[[5]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,of%20the%20complexity). Therefore, we will not hesitate to use complex shaders or large particle counts on devices that can handle them, embracing a “progressive enhancement” approach for high-end users.

At the same time, we recognize many NFA Bears users will be on smartphones during live events (outdoors, on battery)[[15]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AInsight%203%2A%2A%3A%20%2A%2AMobile,Conservative%20defaults%20are%20culturally%20appropriate). Crashes or slowdowns in those contexts are unacceptable. Thus, **mobile stability is ensured via adaptive quality and fallbacks** rather than by disabling effects globally. The baseline WebGL fluid will automatically reduce simulation resolution or step down effects if the framerate dips, preventing performance cliffs on weaker devices[[16]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,motion%20creates%20authentic%20organic%20behavior). We will also enforce conservative defaults (e.g. moderate starting intensity, limited post-processing) and only scale visuals up on devices proven capable. In summary, **we push the visuals as far as we can, but only when the device can keep up** – delivering awe-inspiring graphics on high-end hardware and a solid, if simpler, experience on everyday phones[[17]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,eliminates%20entire%20categories%20of%20bugs).

## Device Capability Detection & Tiered Enhancements

To achieve the above balance, we will implement a **capability detection system** that assigns each client device a profile (tier) and gates visual features accordingly. On app startup (or on demand), a CapabilityDetector module will gather metrics and feature support from the browser, such as:

* **Graphics API Support:** Check for WebGPU availability first, then WebGL2/WebGL support. This determines the highest rendering technology the device can use[[18]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver).
* **Hardware Performance Clues:** Read the navigator.hardwareConcurrency (CPU cores), navigator.deviceMemory (RAM), and maximum texture size and supported shader precision. Possibly do a quick trial frame render to gauge FPS. These data points feed into choosing low/medium/high profiles.
* **User Context Flags:** Detect if the user has battery-saver mode on or prefers-reduced-motion set, which would override or lower the profile to avoid heavy effects on battery or for accessibility[[18]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver).

Using this detection, we define **tiered visual profiles** in a VisualPolicy object: - **Low Profile:** Only the CSS ambience layers enabled (no WebGL). Used for very low-end devices or when GPU support is absent. This ensures basic color ambience with minimal performance cost[[19]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver). - **Medium Profile:** CSS + the WebGL fluid simulation (baseline). This is the default for most mobile and mid-range hardware – the core fluid effect runs, but no additional heavy shaders or particle systems[[19]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver). - **High Profile:** CSS + WebGL fluid + the R3F thin-film shader overlay. This profile is assigned to higher-end devices (e.g. recent desktops or flagship phones/tablets) that can handle an extra shader. The thin-film effect will be enabled at a modest resolution and blended gently on top of the fluid to enrich colors[[19]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver). If performance drops (e.g. FPS below ~45), the orchestrator can dynamically disable this layer to recover[[20]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,if%20FPS%20drops%20below%2045). - **Ultra Profile:** All layers – CSS, WebGL fluid, thin-film, **plus** the WebGPU particle system – enabled on devices that support WebGPU and have ample GPU power[[19]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver). This would be rare initially (e.g. only certain desktops or future mobile GPUs), and we may require an explicit user opt-in for this mode until WebGPU is more mature. Additionally, ultra mode will be automatically suppressed if the device is in battery saver mode or if the particle layer causes any instability.

Each profile dictates which layers to mount and with what quality settings. The **capability gating** ensures that advanced effects like the thin-film shader and particle halo are **never even initialized on weak devices**, avoiding wasted overhead[[7]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AWeaknesses%2A%2A%3A%20,tier%20devices%20only). Conversely, high-end users get the full spectacle by default. We will implement this gating both at load time (choose initial profile) and at runtime (adjust if conditions change). The fluid simulation already has an internal quality auto-adjust (e.g. lowering resolution if FPS falls) which we will integrate with our system[[21]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AFPS,Respects%20device%20maximum%20tier)[[22]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Respects%20device%20maximum%20tier). By using capability profiles, we solve the previous issue where multiple heavy effects ran concurrently without limits[[23]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AMulti,Diminishing%20aesthetic%20returns) – now, **only the appropriate combination of layers will run on each device**[[24]](file://file_0000000087f4620994e91fc9874a6455#:~:text=1,disabled%20on%20battery%E2%80%91saver).

## Unified Orchestrator & Shared Services

Central to this integration is the creation of a **Visual Orchestrator** module that will coordinate all visual layers and shared resources. Instead of each component operating in isolation (which led to inconsistent behavior and duplicated logic[[25]](file://file_000000004db46209a238eb3bab210df7#:~:text=1.%20%2A%2AIntegration%20Fragmentation%2A%2A%20,palettes%20and%20timing%20across%20layers)), the orchestrator ensures a single source of truth for the visual state. Key responsibilities and design elements of the orchestrator include:

* **Layer Management:** It will mount and unmount the CSS, WebGL fluid, thin-film, and particle layers according to the current VisualPolicy (profile). For example, on a medium-tier device it instantiates the fluid canvas and CSS background, whereas on a high-tier it also instantiates the thin-film R3F Canvas and syncs it with the fluid layer. All layers will be rendered in a proper z-index stack, with CSS at the back and special blending modes if needed (e.g. using a screen blend so that bright colors from fluid and thin-film combine without obscuring UI). This coordinated mounting prevents the prior **“multi-engine layering” problem** of independent canvases competing (e.g. double RAF loops or overdraw)[[23]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AMulti,Diminishing%20aesthetic%20returns) – the orchestrator can ensure one master render loop or at least align frame updates.
* **Unified Audio-Reactive Core:** We will introduce a single **AudioBus** service that analyzes the music and broadcasts normalized audio parameters to all visual elements. This replaces any per-component audio analysis. The AudioBus (likely built on the Web Audio API AnalyserNode) will compute frequency bands (bass, mids, treble), overall volume, and detect beats, at a steady interval[[26]](file://file_0000000087f4620994e91fc9874a6455#:~:text=2%29%20Unified%20Audio%E2%80%91Reactive%20Core%20,Beat%20%E2%86%92%20short%2C%20throttled%20pulses). These values are then injected into each active layer’s logic as inputs. Crucially, we will use **one consistent mapping** of audio to visual parameters across all layers[[27]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,Beat%20%E2%86%92%20short%2C%20throttled%20pulses). For example, a low-frequency bass beat might trigger a “splat” in the fluid sim (an injection of dye) and simultaneously increase the thickness or radius of oil droplets in the thin-film shader[[28]](file://file_0000000087f4620994e91fc9874a6455#:~:text=2%29%20Unified%20Audio%E2%80%91Reactive%20Core%20,Beat%20%E2%86%92%20short%2C%20throttled%20pulses). Mids could control flow speed or swirl intensity, treble might modulate color shimmer or particle sparkle, and so on (based on the mapping table defined internally[[27]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,Beat%20%E2%86%92%20short%2C%20throttled%20pulses)). By centralizing this, **all visual layers respond in unison to the music**, avoiding the previous issue of uneven or conflicting audio reactions between components. The AudioBus also simplifies adjustment of sensitivity or thresholds in one place.
* **Central Palette Management:** Visual coherence will be maintained through a shared **PaletteDirector** service. This will define a set of **era-inspired color palettes** (for authenticity and cultural relevance) and provide a uniform interface for retrieving colors or applying color changes. All layers (fluid simulation, shader, CSS gradients, particles) will pull their colors from the PaletteDirector so that the entire scene uses a coordinated palette[[29]](file://file_0000000087f4620994e91fc9874a6455#:~:text=4,layers%3B%20avoid%20double%20sRGB%20conversions). Internally, these palettes include classic liquid light show hues (e.g. magenta, cyan, yellow, orange as in 1960s shows) as well as Grateful Dead-themed variations (like “Dark Star” purples/blues, “Terrapin” greens)[[30]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AColor%20Palettes%2A%2A%20%28wavelength,orange). The PaletteDirector ensures the same selected palette is applied everywhere, preventing the past inconsistency of different modules using slightly different color sets or phase offsets. It will also handle color space conversions and tone mapping centrally (e.g. ensure we only apply gamma correction once overall, to avoid double-brightness issues)[[31]](file://file_0000000087f4620994e91fc9874a6455#:~:text=4,layers%3B%20avoid%20double%20sRGB%20conversions)[[32]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,layers%3B%20avoid%20double%20sRGB%20conversions). By unifying color and tone, we avoid clashing visuals (no more one layer being oddly brighter or differently tinted than another).
* **Global Visual State & Policies:** The orchestrator will maintain global state flags such as motionEnabled (whether animations are running) and an overall intensity level. These can be toggled by user settings or heuristics (for example, a global intensity dial that dims or amplifies all effects uniformly). Layers will all listen to these flags. For instance, if motionEnabled = false (perhaps set by an accessibility preference), the fluid and particle simulations can freeze or greatly slow their dynamics, and animated CSS classes can pause, yielding a mostly static backdrop[[33]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,and%20strobe%20frequency%3B%20document%20defaults). If intensity is lowered, the orchestrator can reduce effect strengths (e.g. fewer splats, dimmer colors, less opacity) across the board. This unified control prevents scenarios where one part of the system kept animating or stayed at full intensity while others did not. It also simplifies implementing a “kill switch” for visuals if needed during performance issues or user request.
* **Elimination of Duplicated Code:** As part of unification, we will refactor common logic out of individual implementations into shared modules. For example, if the thin-film shader and WebGPU engine both need a wavelength-to-RGB conversion function, we will implement it once (perhaps in the PaletteDirector or a shader include) and reuse it, rather than having two slightly different versions as currently observed[[2]](file://file_000000004db46209a238eb3bab210df7#:~:text=4.%20%2A%2ACode%20Duplication%2A%2A%20,code%20paths%20with%20slight%20variations). Similarly, any noise/curl functions or easing curves used in multiple shaders will be centralized. This prevents divergent behavior and reduces maintenance. The orchestrator will enforce consistency by injecting the same constants and parameters into all layers. Our internal review noted that previously, slight differences in shader code and constants led to uneven visuals[[34]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,capability%20negotiation%2C%20uneven%20color%2Ftone%20mapping)[[35]](file://file_000000004db46209a238eb3bab210df7#:~:text=,palettes%20and%20timing%20across%20layers) – unifying these will ensure all layers truly feel part of one visual system.

In summary, the Visual Orchestrator acts as the **conductor of a “liquid light orchestra”**, making sure each visual instrument (layer) plays in harmony under the same tempo (audio input), key (color palette), and sheet music (policy). This addresses the prior integration fragmentation and provides one cohesive pipeline for future tuning[[17]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,eliminates%20entire%20categories%20of%20bugs). Once in place, this orchestrator becomes the backbone for all visual effects in the app, greatly simplifying further enhancements or adjustments.

## Defaults, User Controls & “Pure Mode”

While pushing visual flair, we will also provide clear controls and fallbacks to accommodate user preferences and debugging needs:

* **Sensible Default Experience:** On first run, the app will default to a moderate visual profile suited for common devices (likely the Medium profile: fluid sim + CSS). This ensures new users see the signature liquid light effect immediately without needing to tweak anything. High-tier enhancements (thin-film, etc.) will auto-activate if the device is detected as capable, but we will keep their influence somewhat subtle out of the box (e.g. thin-film overlay at 30-50% opacity[[36]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=float%20n_water%20%3D%201,)) to not overwhelm the UI or users initially. The default color palette might be a balanced "Classic 60s" mix unless a specific event or theme is active. All defaults will be chosen to impress the user while maintaining readability of content (e.g. using darker or muted background colors under text).
* **“Pure Mode” (Enhancements Off):** For development, debugging, or users who prefer a simpler look, we will implement a toggle to disable all optional enhancements. In **Pure Mode**, only the baseline visuals run – that means the CSS background and core fluid simulation remain, but fancy extras like the thin-film shader and particles are turned off (even if the device could handle them). Essentially, Pure Mode forces the visual profile to “Medium” or lower regardless of device. This is useful for isolating performance issues or comparing how much the enhancements add. It also serves as a fallback if, say, a particular shader effect is suspected to cause a bug – one can switch to pure baseline to verify the baseline is stable. We may expose this as a developer setting (e.g. a URL param or debug menu toggle) and/or an advanced user setting. In the UI it could be labeled something like “Visual Enhancements: On/Off” or “Pure Analog Mode” for clarity. This mode aligns with internal recommendations to have an easily accessible baseline-only option for testing and conservative scenarios[[37]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,in).
* **User Intensity & Motion Controls:** We will add a **global intensity slider** and a **motion toggle** in the app settings or as on-screen controls. The intensity slider lets users dial down (or up) the overall brightness and activity of the visuals – effectively scaling the effect from subtle ambient mood lighting to full psychedelic explosion. This is helpful because what is “dope” can be subjective: some might love the maximum setting at a dance party, while others browsing content quietly might prefer a toned-down background. The motion toggle (or an “Animated Background: On/Off” switch) allows users who are motion-sensitive or simply want less distraction to pause the dynamic movement. Under the hood, turning motion off would freeze the fluid sim in a slowly morphing static state (or significantly slow its timestep) and stop any audio-reactive eruptions, while perhaps still cycling colors very gently so the screen isn’t entirely static[[33]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,and%20strobe%20frequency%3B%20document%20defaults). These controls give users agency and also double as safety measures (e.g. the motion toggle is crucial for accessibility to respect prefers-reduced-motion preferences automatically[[38]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,slow%20drift%20or%20static%20gradients), and intensity limiting can help prevent triggering photosensitivity).
* **Battery and Fail-safe Defaults:** In addition to user-exposed toggles, the system will automatically adjust in certain conditions. If the device indicates low battery or power saver mode, the orchestrator will proactively drop to a lower profile (e.g. disabling thin-film/particles or even pausing the fluid sim) to conserve energy. If a WebGL context is lost or an error occurs loading a shader (which can happen on some older GPUs), the orchestrator will catch that and revert to the CSS-only backdrop (ensuring a blank screen or crash is avoided)[[39]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Fire%20on%20the%20Mountain). These fallbacks will be tested to be seamless – the user might just see the visuals simplify, but the app remains running. We will document a “Pure Mode” and fallback for support so that if any user reports issues, they can be instructed to use that mode while we troubleshoot.

By providing these controls and modes, we accommodate both ends of the spectrum: those who want **all the trippy visuals** can get them (when hardware permits), and those who need or want a toned-down experience can easily opt out of enhancements. This flexibility also aids development and live-event operations (for example, if at an event the visuals are interfering with a presentation, an admin could quickly switch to pure mode temporarily). All the while, our default settings ensure that the typical user gets a **thrilling yet reliable** visual experience without any configuration.

## Implementation Roadmap

To execute this integration, we will follow a phased roadmap. Each phase builds on the last, ensuring we gradually unify the system, harden performance, and then add polish and cultural flavor. Below is a multi-phase plan with concrete steps:

### Phase 1: **Stabilize & Unify Baseline (Week 1)**

**Objective:** Clean up the existing code and establish the foundation for orchestration. We focus on removing blockers (like debug code) and consolidating core functionality into shared services.

* **Purge Test/Debug Code:** Remove or gate any development hacks that affect visuals in production. This includes things like the hard-coded "BRIGHT MAGENTA TEST" fill or forced shader overrides that were left in some components[[40]](file://file_000000004db46209a238eb3bab210df7#:~:text=2.%20,Debug%20code%20affecting%20user%20experience). By eliminating these, the visuals will consistently reflect real data and settings. If any such code is needed for future debugging, it will be behind explicit debug flags rather than always running.
* **Centralize Audio Analysis:** Implement the **AudioBus** module (@/visual/audio-mapping). Migrate all components to use this single audio source instead of creating their own AnalyserNode. Ensure it computes bass, mids, treble, etc., and verify that it produces a smooth, stable stream of values. Remove old per-layer audio code. All visual elements will now subscribe to the AudioBus outputs[[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR).
* **Implement PaletteDirector Service:** Create a shared @/visual/palette module that holds all color palettes and provides utilities (e.g. function to get colors by index or time, interpolate colors, convert wavelengths to RGB). Integrate the existing palette data from fluid config and shader code into this service. Then, refactor the fluid sim, thin-film shader, CSS backgrounds, etc., to all pull colors from PaletteDirector (e.g. via context or imported module) instead of hardcoding their own palette arrays[[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR). This enforces unified color usage from the start.
* **Introduce CapabilityDetector & VisualPolicy:** Develop a **CapabilityDetector** utility that runs on app startup to determine the device profile (low/med/high/ultra). Use simple criteria initially (e.g. check for WebGL2 vs WebGPU, deviceMemory thresholds) to assign a profile, and store this in a global VisualPolicy object or context provider. The VisualPolicy will include the chosen profile and any global flags (like motion enabled default, intensity default)[[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR). For now, we can hard-code the profile presets (which layers correspond to each) in this object. Also clamp the device DPR for canvas rendering (e.g. max DPR of ~1.5 on mobile) to avoid extreme GPU load from high pixel densities[[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR). At the end of Phase 1, we should have the basic scaffolding: one audio service, one palette service, and a known device profile on startup, but still essentially running the same baseline fluid as before (just cleaner).

### Phase 2: **Orchestration & Consistency (Week 2)**

**Objective:** Build the orchestrator that ties layers together and ensure all layers behave consistently (colors, controls, etc.). Integrate the thin-film shader and other layers via the orchestrator with the new unified services.

* **Build the VisualOrchestra Provider:** Create a React context provider (or equivalent) called VisualOrchestra that wraps the app UI. This component will read the VisualPolicy (from Phase 1) and conditionally render the appropriate visual layers as its children. It mounts the CSS ambience (likely as global styles or a background div), the FluidCanvas component, the ThinFilmCanvas component, and the ParticleCanvas if applicable[[42]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%202%20%E2%80%94%20Orchestration%20and,GPU%20paths%3B%20wire%20UI%20controls). Use the policy to decide which ones to include. Ensure that these layers all receive the unified inputs: e.g., pass the AudioBus data and PaletteDirector output into the shader uniforms/props of the thin-film and particle layers (so they use the same audio reactive values and colors as the fluid). This step effectively **brings all pieces together on screen simultaneously** under one controller.
* **Normalize Color/Tone Across Layers:** Using the PaletteDirector, adjust each layer to use it correctly. For example, remove any extra gamma correction in the shader if the fluid already outputs in sRGB, or vice versa, so that both layers blend nicely without double correction[[31]](file://file_0000000087f4620994e91fc9874a6455#:~:text=4,layers%3B%20avoid%20double%20sRGB%20conversions). Set consistent brightness levels – e.g., if the fluid sim has an intensity multiplier and the thin-film has its own, calibrate them so that "100% intensity" means a similar level of brightness in each. Establish an intensity clamp to avoid over-saturation (the PaletteDirector can enforce a max brightness)[[43]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%202%20%E2%80%94%20Orchestration%20and,GPU%20paths%3B%20wire%20UI%20controls). The goal is to make the composite visual look like one single coherent effect. We will test with various palettes to ensure, for instance, that a magenta from the fluid and the same magenta in the shader look identical on screen.
* **Wire Up Global Controls:** Connect the **motion toggle and intensity slider** to the orchestrator. The VisualOrchestra context should expose motionEnabled and globalIntensity values (initially true and 1.0 by default). The fluid simulation and other layers will read these. Implement the logic such that toggling motionEnabled to false will pause or greatly slow any procedural animations (we can call an internal method on the fluid engine to stop adding new splats and perhaps reduce velocity to near zero, and similarly ensure the shader stops any time-based interference oscillation). The intensity value can be piped into shader uniforms (modulating color brightness) and fluid parameters (e.g. scaling dye injection amount). Also ensure the orchestrator checks prefers-reduced-motion: if a user has that setting, we initialize motionEnabled = false (or a very low motion mode) automatically[[33]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,and%20strobe%20frequency%3B%20document%20defaults). These controls should be accessible via UI for testing now.
* **Ensure Uniform Audio & Timing:** Confirm that all layers are using the AudioBus values correctly and at the same update frequency. There may be slight differences in how often they were updating (fluid might be tied to its frame loop, shader might use a separate loop). We should ideally synchronize their update loops if possible (e.g. have the R3F thin-film render on the same rAF tick as the fluid by coordinating contexts, or simply let each run but since both subscribe to the same audio data which is updated at a fixed cadence, they should naturally sync on beat events). We may consider merging multiple rAF loops into one if feasible to further ensure synced timing and reduce overhead[[44]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=6.%20%2A%2AParallel%20RAF%20Loops%2A%2A%20,to%20single%20loop%20per%20engine). By end of Phase 2, we should have the **full visual stack rendering together**, controllable, and looking consistent in color and movement. The advanced layers will still be off on low-tier devices (because VisualPolicy will not mount them), but we will manually test switching profiles to ensure each combination works.

### Phase 3: **Performance Optimization & Fallbacks (Week 3)**

**Objective:** Rigorously improve performance on lower-end devices and implement robust fallback behavior for edge cases. This phase makes sure the orchestrated system is stable under real-world conditions.

* **Adaptive Resolution & Frame Pacing:** Enhance the fluid simulation and any heavy shaders with adaptive quality logic. The fluid engine already monitors FPS and adjusts simulation resolution – we will double-check these thresholds and possibly tighten them (e.g. drop quality if <30 FPS)[[21]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AFPS,Respects%20device%20maximum%20tier). Similarly, for the thin-film shader, we can adjust its internal resolution or frequency of updates based on performance: for instance, render the thin-film texture at half-res on mobile and only every other frame if needed (since its changes are subtle) – essentially frame skipping or reducing update rate when needed. Implement a simple **FPS monitor** in the orchestrator: measure the combined frame rate of our visuals. If it drops below ~45 FPS for more than a couple seconds on a high-tier profile, automatically downgrade the profile (e.g. disable the thin-film layer) to recover[[45]](file://file_0000000087f4620994e91fc9874a6455#:~:text=6%29%20Performance%20Governance%20,5s%20within%20device%20capability)[[46]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,to%20WebGL%2FCSS%20preserving%20palette%2Faudio%20state). Conversely, if performance is a steady 60 FPS and the device is high-end, we might cautiously enable an enhancement. This dynamic tier switching ensures we stay within performance budgets. We'll also clamp the **device pixel ratio** for canvases (if not done in Phase 1) – e.g. on mobile, never use DPR > 1.5 regardless of device, to prevent 4K screens from choking on full resolution rendering[[47]](file://file_0000000087f4620994e91fc9874a6455#:~:text=6%29%20Performance%20Governance%20,5s%20within%20device%20capability).
* **Robust Fallback for WebGPU:** Implement thorough feature detection and error handling around the WebGPU particle layer. Before enabling WebGPU, check not only for API presence but also for required features (like shader float16 support) and perhaps run a small test compute shader to verify it works. If any check fails or if the particle initialization throws an exception, the orchestrator will log it and **fall back gracefully** by excluding the particle layer and switching to High profile (fluid + thin-film only)[[46]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,to%20WebGL%2FCSS%20preserving%20palette%2Faudio%20state). This way, even if a device falsely reports WebGPU available but then fails, the app continues with the other visuals. We will keep the WebGPU layer code split (dynamically imported) so that if it's not used, it doesn't block main thread or bundle loading. Essentially, WebGPU will be a *bonus* that activates only on proven systems, and otherwise everything else runs normally without it[[48]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AWeaknesses%2A%2A%3A%20,Unpredictable%20support%20across%20iOS%2FAndroid%20variants)[[49]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,universal%20baseline%20yet).
* **Memory and Resource Management:** Audit and fix any potential memory leaks or heavy resource usage now that multiple layers can be active. For instance, ensure that when we unmount the thin-film R3F component (on a profile downgrade or navigation away), we properly dispose of its Three.js renderer, textures, and materials to free GPU memory. The orchestrator should manage mounting/unmounting in a way that avoids accumulating canvases or contexts (only one fluid context and one R3F context max at any time). We should also verify the fluid sim’s context loss recovery works with our new setup – e.g. simulate a webglcontextlost event and confirm the CSS fallback kicks in and then the canvas restores correctly[[39]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Fire%20on%20the%20Mountain). This phase might include adding a **minimal debugging HUD** (for development builds only) that shows FPS, current profile, memory, etc., over the app[[50]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%203%20%E2%80%94%20Performance%20and,to%20WebGL%2FCSS%20preserving%20palette%2Faudio%20state). That will help testers observe when the profile auto-switches or if any performance issues arise in real-time.
* **Cross-Device Testing:** Test the integrated system on a range of devices: low-end Android phone (should default to low/medium and run smoothly at ~30-60 FPS on just CSS or fluid), mid-range laptop (fluid and maybe thin-film at 60 FPS), high-end gaming PC (should engage all layers at 60 FPS). Also test on an iPhone with Safari that supports WebGPU to see the ultra mode in action. Use these tests to fine-tune threshold values for tier decisions and to catch any platform-specific issues (e.g. certain mobile browsers might struggle with offscreen canvas usage, etc.). By end of Phase 3, we expect the visual system to **self-regulate its performance** and not crash or lag on any device – it should seamlessly scale down if needed. Fallbacks for WebGPU or any failure should be in place so the user always has at least the fluid or CSS visuals.

### Phase 4: **Cultural Polish & Presets (Week 4)**

**Objective:** Add the finishing touches that emphasize cultural authenticity and provide curated experiences. This includes fine-tuning the art direction (colors, themes, transitions) and ensuring the visuals enhance the content without overpowering it.

* **Curate Signature Palettes & Themes:** Finalize 3-5 **preset themes** that users (or event organizers) can choose from[[51]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%204%20%E2%80%94%20Art%20Direction,UI%3B%20add%20scrims%20where%20necessary). These might be named after Grateful Dead references or classic light show motifs, for example: "Dark Star Deep Space" (cool purples and blues, slower motion), "China Cat Sunburst" (bright warm oranges and yellows, playful motion), "Terrapin Station Earth Tones" (greens and browns, organic feel), "Ripple Rainbow" (full-spectrum cycling), and "Classic Analog" (the default 60s oil colors)[[51]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%204%20%E2%80%94%20Art%20Direction,UI%3B%20add%20scrims%20where%20necessary). Each theme would configure the PaletteDirector with a specific palette and maybe adjust intensity/motion profiles (e.g. a “Trip Mode” theme might crank intensity and motion). By providing these presets, we give a nod to the cultural context (Deadhead lore and 60s art) and make the experience feel tailored and meaningful to fans.
* **Integrate Theming UI:** Expose the above themes in the UI (perhaps as part of settings or as special event-based toggles). For instance, when a certain song is playing, the app could automatically switch to the corresponding theme palette. Ensure that switching themes is smooth – possibly crossfade between palettes or gradually blend new colors in to avoid jarring changes. Also, document the default theme and allow an easy reset to default.
* **UI Legibility & Overlay Adjustments:** Review all app screens with the new dynamic backgrounds to ensure text and content remain readable. Identify any areas where bright colors or motion could interfere (for example, white text might become hard to read if a very light color flows behind it). Implement **scrims or backdrop filters** behind text as needed[[52]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,UI%3B%20add%20scrims%20where%20necessary)[[53]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,scrims%20under%20animated%20high%E2%80%91contrast%20regions). This could be a semi-transparent dark layer under modals or increasing the opacity of a blur in the CSS layer right behind text. The goal is to strike a balance: keep the visuals visible and cool, but never at the expense of usability. We will especially test edge cases like high-contrast palettes (the "Ripple Rainbow" might have very light colors) to ensure the UI has adaptive contrast measures (like automatically darkening the background slightly when needed).
* **Historical Authenticity Check:** Do a final pass comparing our visuals to known historical references of liquid light shows (e.g. videos of Joshua Light Show performances). Ensure that our motion and color feel “right” – the thermal convection upward drift, the color mix that comes from oil and water separation, etc., are as authentic as feasible. Our internal research validates that we included the key physics (buoyancy, interference patterns)[[54]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=5.%20%2A%2AThermal%20Convection%20Physics%2A%2A%20,5%20second%20intervals)[[55]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,alongside%20film%20projections%20and%20slides). In this step, we might tweak parameters like the fluid’s curl (swirliness) or the thin-film’s color intensity to better emulate the real analog look. We will also gather feedback from a few team members or community beta testers who are fans of the Grateful Dead to see if the vibe resonates with them. Any feedback about certain colors or speeds being off will be considered for adjustment.
* **Finalize Documentation & Controls:** Document all the settings in a **Visual Readme** or internal wiki: e.g. what each theme signifies, how to activate pure mode, how the auto-tiering works, etc. Also, double-check that the user controls (intensity slider, motion toggle, theme selector) are discoverable and working properly. We should also include a mention of these features in the app’s FAQ or tutorial, so users know they can customize the experience. This phase is about making the feature not just technically sound but also **culturally delightful and user-friendly**.

By the end of Phase 4, the liquid light show integration will not only be technically robust, but also rich with the cultural essence that inspired it. The visuals will have the **authentic psychedelic soul** of a 1960s Grateful Dead light show, integrated seamlessly into a modern app environment.

## Conclusion & Next Steps

Following this plan, we will achieve a unified, adaptive liquid light show feature that greatly enhances the NFA Bears app’s visual appeal without sacrificing stability. We have identified that our existing WebGL fluid engine is a strong foundation[[3]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,or%20gated%20as%20optional%20enhancements), and by layering on thin-film shaders and other effects in a controlled manner, we harness the best of each approach. The new Visual Orchestrator ensures **one coherent experience**, addressing past integration issues where multiple implementations clashed[[1]](file://file_000000004db46209a238eb3bab210df7#:~:text=1.%20%2A%2AIntegration%20Fragmentation%2A%2A%20,palettes%20and%20timing%20across%20layers). With device-based gating, we **favor authenticity and intensity on capable devices** while still delivering a smooth experience on mobile[[56]](file://file_0000000087f4620994e91fc9874a6455#:~:text=orchestration%2C%20device%E2%80%91tiering%2C%20consistency%2C%20and%20fallbacks,perceived%20authenticity%20over%20maximal%20physics)[[15]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AInsight%203%2A%2A%3A%20%2A%2AMobile,Conservative%20defaults%20are%20culturally%20appropriate). User and developer controls like intensity tuning and Pure Mode provide flexibility and safety nets.

After implementing Phase 4, the team should rigorously QA the system during a live music stream or similar scenario to see it in action. Metrics like frame rate, memory usage, and user feedback (“Is it visually amazing?”, “Is anything distracting or slow?”) will guide any final tweaks. Given the phased approach, we can also choose to deploy incrementally (e.g. deploy Phase 1 and 2 changes first as they mainly refactor and unify the baseline visuals, then enable Phase 3 optimizations, and finally roll out Phase 4 themes for a big event).

Ultimately, this integration will transform the NFA Bears app’s look and feel, **immersing users in an authentic liquid light show** that honors the Grateful Dead legacy. By following the above steps and timeline, we ensure that the result is not only spectacular and "dope" but also **technically sound, accessible when needed, and culturally respectful** of the art form that inspired it[[17]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,eliminates%20entire%20categories%20of%20bugs). We will proceed with Phase 1 immediately, setting the stage for the enhanced visual journey to come.

**Sources:** Internal analysis reports on liquid light implementations and project documentation[[3]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,or%20gated%20as%20optional%20enhancements)[[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR), which provided the basis for this plan’s recommendations. All technical and cultural considerations above are drawn from those findings and will guide the implementation.

[[1]](file://file_000000004db46209a238eb3bab210df7#:~:text=1.%20%2A%2AIntegration%20Fragmentation%2A%2A%20,palettes%20and%20timing%20across%20layers) [[2]](file://file_000000004db46209a238eb3bab210df7#:~:text=4.%20%2A%2ACode%20Duplication%2A%2A%20,code%20paths%20with%20slight%20variations) [[12]](file://file_000000004db46209a238eb3bab210df7#:~:text=,ambience%20requiring%20no%20GPU%20context) [[13]](file://file_000000004db46209a238eb3bab210df7#:~:text=,conversion%20matches%20real%201960s%20techniques) [[25]](file://file_000000004db46209a238eb3bab210df7#:~:text=1.%20%2A%2AIntegration%20Fragmentation%2A%2A%20,palettes%20and%20timing%20across%20layers) [[35]](file://file_000000004db46209a238eb3bab210df7#:~:text=,palettes%20and%20timing%20across%20layers) [[40]](file://file_000000004db46209a238eb3bab210df7#:~:text=2.%20,Debug%20code%20affecting%20user%20experience) liquid-light-show-analysis-cursor-2025-10-29-1956.md

<file://file_000000004db46209a238eb3bab210df7>

[[3]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,or%20gated%20as%20optional%20enhancements) [[4]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AStrengths%2A%2A%3A%20,motion%20creates%20authentic%20organic%20behavior) [[5]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,of%20the%20complexity) [[6]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,mode%20palettes%20provide%20cultural%20depth) [[7]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AWeaknesses%2A%2A%3A%20,tier%20devices%20only) [[9]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Diminishing%20aesthetic%20returns) [[10]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=4.%20%2A%2AWebGPU%20Universal%20Deployment%2A%2A%20,support%20limited%20to%20Qualcomm%2FARM%20GPUs) [[14]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Stokes%20equations) [[15]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AInsight%203%2A%2A%3A%20%2A%2AMobile,Conservative%20defaults%20are%20culturally%20appropriate) [[16]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,motion%20creates%20authentic%20organic%20behavior) [[17]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,eliminates%20entire%20categories%20of%20bugs) [[20]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,if%20FPS%20drops%20below%2045) [[21]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AFPS,Respects%20device%20maximum%20tier) [[22]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Respects%20device%20maximum%20tier) [[23]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=3.%20%2A%2AMulti,Diminishing%20aesthetic%20returns) [[30]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AColor%20Palettes%2A%2A%20%28wavelength,orange) [[36]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=float%20n_water%20%3D%201,) [[37]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,in) [[39]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,Fire%20on%20the%20Mountain) [[44]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=6.%20%2A%2AParallel%20RAF%20Loops%2A%2A%20,to%20single%20loop%20per%20engine) [[48]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=%2A%2AWeaknesses%2A%2A%3A%20,Unpredictable%20support%20across%20iOS%2FAndroid%20variants) [[49]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,universal%20baseline%20yet) [[54]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=5.%20%2A%2AThermal%20Convection%20Physics%2A%2A%20,5%20second%20intervals) [[55]](file://file_000000003b5c6209b14492a399ae2d72#:~:text=,alongside%20film%20projections%20and%20slides) claude-code-liquid-light-analysis.md

<file://file_000000003b5c6209b14492a399ae2d72>

[[8]](file://file_0000000087f4620994e91fc9874a6455#:~:text=variants.%20,film%20needs%20direction%20when%20composed) [[11]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,layers%20for%20cohesion%3B%20intensity%20governance) [[18]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver) [[19]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,disabled%20on%20battery%E2%80%91saver) [[24]](file://file_0000000087f4620994e91fc9874a6455#:~:text=1,disabled%20on%20battery%E2%80%91saver) [[26]](file://file_0000000087f4620994e91fc9874a6455#:~:text=2%29%20Unified%20Audio%E2%80%91Reactive%20Core%20,Beat%20%E2%86%92%20short%2C%20throttled%20pulses) [[27]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,Beat%20%E2%86%92%20short%2C%20throttled%20pulses) [[28]](file://file_0000000087f4620994e91fc9874a6455#:~:text=2%29%20Unified%20Audio%E2%80%91Reactive%20Core%20,Beat%20%E2%86%92%20short%2C%20throttled%20pulses) [[29]](file://file_0000000087f4620994e91fc9874a6455#:~:text=4,layers%3B%20avoid%20double%20sRGB%20conversions) [[31]](file://file_0000000087f4620994e91fc9874a6455#:~:text=4,layers%3B%20avoid%20double%20sRGB%20conversions) [[32]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,layers%3B%20avoid%20double%20sRGB%20conversions) [[33]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,and%20strobe%20frequency%3B%20document%20defaults) [[34]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,capability%20negotiation%2C%20uneven%20color%2Ftone%20mapping) [[38]](file://file_0000000087f4620994e91fc9874a6455#:~:text=5,slow%20drift%20or%20static%20gradients) [[41]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%201%20%E2%80%94%20Stabilize%20and,with%20profile%20presets%3B%20clamp%20DPR) [[42]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%202%20%E2%80%94%20Orchestration%20and,GPU%20paths%3B%20wire%20UI%20controls) [[43]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%202%20%E2%80%94%20Orchestration%20and,GPU%20paths%3B%20wire%20UI%20controls) [[45]](file://file_0000000087f4620994e91fc9874a6455#:~:text=6%29%20Performance%20Governance%20,5s%20within%20device%20capability) [[46]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,to%20WebGL%2FCSS%20preserving%20palette%2Faudio%20state) [[47]](file://file_0000000087f4620994e91fc9874a6455#:~:text=6%29%20Performance%20Governance%20,5s%20within%20device%20capability) [[50]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%203%20%E2%80%94%20Performance%20and,to%20WebGL%2FCSS%20preserving%20palette%2Faudio%20state) [[51]](file://file_0000000087f4620994e91fc9874a6455#:~:text=Phase%204%20%E2%80%94%20Art%20Direction,UI%3B%20add%20scrims%20where%20necessary) [[52]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,UI%3B%20add%20scrims%20where%20necessary) [[53]](file://file_0000000087f4620994e91fc9874a6455#:~:text=,scrims%20under%20animated%20high%E2%80%91contrast%20regions) [[56]](file://file_0000000087f4620994e91fc9874a6455#:~:text=orchestration%2C%20device%E2%80%91tiering%2C%20consistency%2C%20and%20fallbacks,perceived%20authenticity%20over%20maximal%20physics) codex-cli-liquid-light-visual-orchestra-report-2025-10-29T19-32-37Z.md

<file://file_0000000087f4620994e91fc9874a6455>