Product Requirements Document: Integrated Animation Architecture for React Bits – Complementary Animation Engines

1. Executive Summary and Architectural Vision

1.1. Strategic Context and Project Scope

The digital landscape for enterprise-grade web applications has shifted precipitously toward immersive, spatial computing experiences. "React Bits," as a platform, sits at the intersection of high-performance WebGL rendering and modern, reactive user interfaces. This Product Requirements Document (PRD) establishes the definitive technical standard for the **Complementary Animation Libraries** module. The directive is clear: to engineer a "Fortune 500" caliber system that bridges the gap between the Document Object Model (DOM) and the WebGL Canvas without compromising the performance metrics essential for live, web-based deployment.

The core challenge in hybrid 2D/3D development is the synchronization of two disparate rendering engines. The DOM, governed by the browser's layout engine and React's reconciliation cycle, operates on a distinct cadence from the WebGL context, which is driven by the requestAnimationFrame (RAF) loop and the GPU's immediate mode execution. Attempts to force one engine to control the other—such as driving 3D transforms via React state or manipulating DOM layout via WebGL ticks—historically result in "jitter," frame drops, and a disjointed user experience often described as the "uncanny valley" of web animation.

To resolve this, this PRD mandates a **Bimodal Animation Architecture**. This strategy explicitly decouples the interface layer from the scene layer, assigning "best-in-class" libraries to their respective domains while establishing a rigorous, high-speed communication bus between them. The architecture leverages **Motion** (formerly Framer Motion) for the declarative, state-driven world of UI overlays, and **GSAP** (GreenSock Animation Platform) for the imperative, time-based orchestration of the 3D scene. These are unified by **Lenis** for scroll normalization and **Zustand** for transient state synchronization.

1.2. The Business Case for Hybrid Animation

In a production environment, the animation stack is not merely aesthetic; it is a critical component of the user experience (UX) and conversion funnel. High-fidelity animations guide user attention, mask loading latencies, and provide spatial context. However, poorly implemented animations that induce layout thrashing or memory leaks can severely degrade

Core Web Vitals, specifically Interaction to Next Paint (INP) and Cumulative Layout Shift (CLS).

The selection of Motion and GSAP is strategic. Motion provides the developer velocity required for complex React UI states (mounting, unmounting, layout shifts) which are notoriously brittle to implement manually. GSAP offers the robust timeline management and browser compatibility—handling transform inconsistencies across Safari, Chrome, and Firefox—that is non-negotiable for a public-facing, high-traffic application. This duality ensures that the application remains maintainable by React developers while achieving the cinematic fidelity of a high-end creative coding project.

1.3. LLM Code Generation Directive

This document serves as the authoritative source of truth for Large Language Models (LLMs) tasked with generating code for React Bits. The specifications detailed herein are not suggestions; they are constraints. Specifically, the architectural patterns defined in **Section 5 (Implementation Specifications)** and the prohibition of deprecated libraries in **Section 6 (Anti-Patterns)** must be strictly adhered to during code synthesis to prevent technical debt and ensure compatibility with React 19 and the latest React Three Fiber (R3F) ecosystem.

---

2. Technical Stack Definition and Dependency Versioning

The stability of a production application relies heavily on strict dependency management. The following table defines the mandatory technology stack. This selection process was driven by an analysis of long-term maintainability, community support, and performance benchmarks specific to the React 18/19 transition.

Component	**Library / Package**	**Version Requirement**	**Architectural Justification**
2D UI Engine	motion (formerly framer-motion)	**v11.0+**	The industry standard for React DOM animation. Motion v11 introduces significant performance optimizations and prepares the

			codebase for the React 19 compiler. The rebranding from framer-motion to motion is a critical import distinction.[1]
3D Timeline Engine	gsap	**v3.12.5+**	The only viable solution for complex timeline sequencing and ScrollTrigger functionality that performs reliably across all browsers. The @gsap/react package is mandatory for handling React Strict Mode cleanup.[3]
State Bridge	zustand	**v4.5+**	Selected for its flux-like simplicity and, crucially, its support for **transient updates**. This allows state to be read inside the R3F loop without triggering component re-renders.[4]
Scroll Normalization	lenis	**v1.1+**	Essential for intercepting and smoothing scroll events before they reach GSAP or the Canvas, ensuring 3D scroll

			animations are fluid rather than stepped.[6]
3D Reconciler	@react-three/fiber	**v8.15+**	The React renderer for Three.js. Version alignment with React 18/19 is critical for concurrent mode support.[8]
3D Utilities	@react-three/drei	**v9.100+**	Provides essential abstractions like View, Html, and useGLTF.

2.1. Critical Deprecation Notice: framer-motion-3d

It is imperative to note that **framer-motion-3d is officially deprecated** and must be excluded from the project's dependency tree.[9]

- **Status:** The library is unmaintained and incompatible with React 19.
- **Risk:** Usage of this library introduces "invalid hook call" errors and breaks the reconciliation of Three.js elements in modern React environments.[11]
- **Directive:** All LLM-generated code must avoid importing from framer-motion-3d. 3D animations previously handled by this library (e.g., <motion.mesh>) must be refactored to use **GSAP** or native R3F useFrame hooks.

---

3. Component A: The Declarative 2D Layer (Motion)

3.1. Role Definition and Architectural Fit

Motion (formerly Framer Motion) is assigned the exclusive responsibility of managing the **User Interface (UI) Layer**. In the context of "React Bits," the UI acts as the control surface for the 3D experience—comprising navigation menus, Heads-Up Displays (HUDs), modal overlays, and informational tooltips.

The architectural decision to use Motion for the DOM (and not GSAP) stems from React's declarative nature. React developers describe *states* (e.g., isOpen, isActive), and React renders the resulting DOM. Motion hooks directly into this lifecycle. It excels at handling the "between" states—specifically the entry and exit transitions of components—which are notoriously difficult to coordinate in imperative libraries like GSAP without verbose scaffolding.

3.2. Core Functional Requirements

3.2.1. Presence Management (Exit Animations)

A recurring requirement in 3D web applications is the smooth transition of overlay panels as the user navigates the 3D scene.

- **Requirement:** The system must utilize <AnimatePresence> to manage the unmounting of UI components.
- **Use Case:** When a user clicks a "Close" button on a detailed product view, the 2D information panel must animate its opacity to 0 and slide off-screen *before* it is removed from the DOM. This prevents jarring layout shifts and maintains the illusion of a polished application.[13]
- **Implementation Note:** React 18's concurrent features require strict adherence to the mode="wait" or mode="popLayout" props to ensure layout stability during rapid state toggles.

3.2.2. Layout Projection (The "Magic Motion" Effect)

To seamlessly blend the 2D and 3D worlds, the UI must appear to physically react to 3D triggers. Motion's layout projection engine is critical here.

- **Requirement:** Use the layout and layoutId props to morph shared components between different parts of the React tree.
- **Scenario:** A user selects an item from a 2D grid list (representing 3D objects). Upon click, the thumbnail in the grid must morph into the header of the full-screen details page. This 2D continuity masks the loading time required for the high-fidelity 3D asset to center itself in the viewport.[13]

3.2.3. Accessibility Integration (Reduced Motion)

Fortune 500 compliance mandates strict adherence to WCAG accessibility standards.

- **Requirement:** All Motion components must respect the user's prefers-reduced-motion operating system setting.
- **Implementation:** The useReducedMotion hook must be utilized to conditionally swap complex transforms (e.g., scale/translate) for simple opacity fades.
- **Constraint:** If "Reduced Motion" is active, auto-playing animations in the UI must be disabled.[15]

3.3. Structural Implementation: The HUD Pattern

A common anti-pattern in R3F development is embedding complex UI directly inside the Canvas using the <Html> component from @react-three/drei. While useful for simple labels, this approach causes event bubbling issues and ties the UI render rate to the WebGL render rate, which can lead to input lag on lower-end devices.

The Mandated Pattern: The UI layer must exist as a generic HTML sibling to the Canvas, positioned absolutely on top.

TypeScript

```
// Architectural Pattern: The Sibling Layout
export const ExperienceLayout = () => {
 return (
  <div className="relative w-full h-screen">
   {/* Layer 1: The 3D Render Context */}
   <div className="absolute inset-0 z-0">
    <Canvas>
     <SceneGraph />
    </Canvas>
   </div>

   {/* Layer 2: The UI Overlay Context */}
   <div className="absolute inset-0 z-10 pointer-events-none">
    <InterfaceLayer />
   </div>
  </div>
 );
};
```

- **Note on Events:** The pointer-events-none class on the container is crucial. It allows mouse interactions to pass through the UI layer to the 3D Canvas below. Interactive children within InterfaceLayer (buttons, inputs) must explicitly re-enable interactions via pointer-events-auto.[16]

---

4. Component B: The Imperative 3D Layer (GSAP)

4.1. Role Definition and Architectural Fit

GSAP is designated as the engine for **Cinematic Orchestration and Complex 3D Transforms**. While React Three Fiber allows for basic animation via useFrame loops, managing multi-stage sequences (e.g., "Camera pans to X, then light intensifies, then object Y rotates") via React useEffect chains creates "callback hell" and unmanageable state complexity.

GSAP resolves this by treating animations as **Timelines**. A Timeline is a deterministic object that can be played, paused, reversed, or scrubbed. This imperative model is superior for 3D because it allows the developer to modify the properties of Three.js objects (Meshes, Lights, Cameras) directly, bypassing React's reconciliation process entirely. This is essential for maintaining a stable 60 FPS.

4.2. The Integration Pattern: useGSAP

Historically, integrating GSAP with React was fraught with issues related to useEffect cleanup, especially with React 18's Strict Mode, which mounts/unmounts components twice in development. This often led to "ghost" ScrollTriggers or duplicate animations running in parallel.

Requirement: All GSAP code within "React Bits" must be encapsulated within the **useGSAP hook** from the @gsap/react package.

- **Mechanism:** useGSAP automatically wraps animation code in a gsap.context. When the React component unmounts, the context is reverted, killing all associated tweens, timelines, and ScrollTriggers instantly.
- **Scope:** The scope prop must be used to limit selector reach. This ensures that a GSAP selector like ".box" only targets elements within the current component, enabling safe component reusability.[3]

4.3. Functional Requirements

4.3.1. Cinematic Camera Control

The 3D camera is the user's window into the world. Its movement must be fluid and cinematic.

- **Constraint:** Do not animate the Camera object directly if using OrbitControls or CameraControls. These helpers fight for control of the camera matrix.
- **Solution:** Animate the target and position properties of the CameraControls instance, or wrap the camera in a "Rig" Group and animate the rig's transforms.
- **GSAP Usage:** Use GSAP to tween these values using custom easing functions (e.g., power3.inOut) to simulate the weight and inertia of a physical camera dolly.[19]

4.3.2. ScrollTrigger Integration (Scrollytelling)

A primary feature of React Bits is "Scrollytelling"—using scroll interaction to progress the 3D narrative.

- **Requirement:** Use GSAP's ScrollTrigger plugin to bind Timeline progress to the scrollbar.
- **Implementation Detail:** Since the 3D Canvas is often fixed position, ScrollTrigger must watch a proxy DOM element (often a wrapper <div> with defined height) to calculate progress.

- **Performance:** scrub: true or scrub: 0.5 (for smoothing) must be used to link the animation directly to the scrollbar, allowing the user to reverse the animation by scrolling up.[20]

4.3.3. Timeline Orchestration

For complex scenes, "fire-and-forget" tweens are insufficient.

- **Requirement:** Construct gsap.timeline() instances to sequence events.
- **Benefit:** This allows for precise control over overlap (using the position parameter, e.g., "<", "+=0.5") and enables global control (pause/resume) of the entire scene's logic, which is vital for debugging and loading states.[21]

---

5. Component C: The Integration Bridge (Lenis & Zustand)

The "Bridge" represents the logic that synchronizes the disparate clocks of the browser (DOM) and the GPU (Canvas). Without this bridge, scroll-driven 3D animations will stutter or drift.

5.1. Scroll Normalization: Lenis

Native browser scrolling is processed on a separate thread (the compositor thread) to ensure responsiveness. However, JavaScript-based animations (like WebGL updates) run on the main thread. This disconnect causes "desynchronization" or jitter, where the 3D scene lags behind the scroll position by a single frame.

Lenis solves this by implementing smooth scrolling on the main thread, allowing for frame-perfect synchronization.

5.1.1. The Synchronization Loop

For the system to function at "Fortune 500" quality, Lenis and GSAP must share the exact same heartbeat.

- **Requirement:** The Lenis requestAnimationFrame (raf) must be manually piped into GSAP's ticker.
- **Code Standard:**

  JavaScript
  ```javascript
  // Critical Synchronization Pattern
  useEffect(() => {
   const lenis = new Lenis({
    duration: 1.2,
    easing: (t) => Math.min(1, 1.001 - Math.pow(2, -10 * t)), // Exponential easing for premium feel
    smoothWheel: true,
  ```

```
  syncTouch: true, // Critical for mobile fidelity
 })

 // 1. Force ScrollTrigger to update whenever Lenis scrolls
 lenis.on('scroll', ScrollTrigger.update)

 // 2. Inject Lenis into GSAP's Ticker
 // GSAP Ticker runs at the display's native refresh rate (usually 60hz or 120hz)
 // This ensures Lenis updates exactly when GSAP updates.
 gsap.ticker.add((time) => {
  lenis.raf(time * 1000) // Convert seconds to MS
 })

 // 3. Disable GSAP's internal lag smoothing
 // Lag smoothing compensates for CPU spikes by "jumping" the animation.
 // In scroll-driven 3D, this looks like a glitch. We disable it.
 gsap.ticker.lagSmoothing(0)

 return () => {
  gsap.ticker.remove(gsap.ticker.lagSmoothing)
  lenis.destroy()
 }
},)
```

Source Support: [23]

5.2. State Synchronization: Zustand

Communication between the DOM (Motion) and the Canvas (GSAP/R3F) cannot rely on React Context, as the Canvas creates a separate React root (reconciler). Passing props through the bridge is possible but cumbersome for deep trees.

Zustand is the mandated state management solution because it exists outside the React tree, acting as a global message bus.

5.2.1. Transient Updates

In high-performance animation, triggering a React re-render 60 times a second (e.g., creating a store that updates scrollPosition) will destroy performance. Zustand allows for **transient updates**.

- **Concept:** Components can subscribe to state changes via the useStore.subscribe method purely for side effects (like updating a ref) without triggering a re-render of the component view.
- **Application:** A GSAP onUpdate callback can write to a Zustand store

(setScrollProgress(0.5)). A component inside the Canvas can subscribe to this change and update a shader uniform directly.

- **Pattern:**
  ```JavaScript
  // Transient subscription pattern
  const useStore = create((set) => ({ progress: 0 }))

  // In Canvas Component
  useEffect(() => useStore.subscribe(
    (state) => (materialRef.current.uniforms.uProgress.value = state.progress)
  ),)
  ```

  This pattern ensures the UI stays responsive without React overhead.[4]

---

6. Implementation Specifications and Folder Structure

To ensure scalability and ease of navigation for large engineering teams, the project must adhere to a strict **Feature-Based Architecture**.

6.1. Folder Structure Standard

The codebase should be organized by domain feature rather than by file type. This creates self-contained modules that are easier to test and refactor.

```
src/
├── features/
│   ├── ProductShowcase/ # Feature Domain
│   │   ├── components/
│   │   │   ├── ProductCanvas.tsx # R3F Scene (GSAP logic here)
│   │   │   ├── ProductOverlay.tsx # Motion UI (DOM logic here)
│   │   │   └── Viewport.tsx # Layout container
│   │   ├── hooks/
│   │   │   └── useProductAnimation.ts # Encapsulated GSAP timelines
│   │   ├── store/
│   │   │   └── useProductStore.ts # Localized Zustand store
│   │   └── index.ts
│   └── Navigation/
│   ├── components/...
│   └──...
├── shared/
│   ├── components/
│   │   ├── canvas/ # Reusable 3D components
│   │   │   ├── CameraRig.tsx
```

```
│ │ │ └── Lighting.tsx
│ │ └── dom/ # Reusable 2D components
│ │ ├── SmoothScroll.tsx # Lenis Wrapper
│ │ └── Loader.tsx
│ ├── hooks/
│ │ └── useReducedMotion.ts
│ └── utils/
│ └── math.ts
└── App.tsx
```
Source Support: 26

6.2. Coding Rules and Anti-Patterns

6.2.1. The "No Motion in Canvas" Rule

Rule: Never import or use framer-motion components (e.g., <motion.div>) inside the <Canvas> component tree.

- **Reasoning:** The <Canvas> renders to a WebGL context, not the DOM. HTML elements inside it (via Drei Html) are expensive to reconcile. Mixing Motion's DOM-based physics with Three.js's frame-based physics leads to timing mismatches.
- **Exception:** If "spring" physics are needed for a 3D object (e.g., a wobbly jelly effect), use @react-spring/three or maath, which are designed for the R3F reconciler.[28]

6.2.2. The Ref Persistence Rule

Rule: Always use useRef to hold references to GSAP timelines and Three.js objects. Never rely on useState for animation controllers.

- **Reasoning:** Storing a timeline in state triggers a re-render when the timeline is created. Since timelines are mutable objects, they should exist in refs to persist across renders without triggering them.

6.2.3. The View vs. Canvas Rule

Rule: When requiring multiple 3D scenes on a single page (e.g., an e-commerce grid where every product is 3D), use **@react-three/drei's <View> component** with a single shared <Canvas> at the root.

- **Anti-Pattern:** Do not render multiple <Canvas> elements on the same page.
- **Reasoning:** Each <Canvas> creates a new WebGL context. Browsers have a hard limit (usually 8-16) on active WebGL contexts. Exceeding this crashes the tab. The <View> component uses gl.scissor to render multiple "virtual" canvases on a single context, offering massive performance gains.[29]

---

7. Performance Optimization & Production Readiness

Reaching "Fortune 500" quality requires rigorous performance profiling. The goal is to maintain **60 FPS** (16.6ms frame budget) consistently.

7.1. Draw Call Management

A primary bottleneck in R3F is the number of draw calls. Each unique geometry/material combination requires a CPU-to-GPU command.

- **Optimization:** Use **InstancedMesh** (via <Instances> from Drei) for repeated objects.
- **GSAP Implication:** When animating instances with GSAP, do not animate the React component. Instead, access the InstancedMesh ref and update the instanceMatrix directly at the specific index using tempObject.updateMatrix(), then set mesh.instanceMatrix.needsUpdate = true in the animation tick.[32]

7.2. Geometry & Material Disposal

Three.js objects must be manually disposed of to free GPU memory.

- **R3F Behavior:** R3F handles disposal automatically when a component unmounts.
- **GSAP Conflict:** If a GSAP timeline holds a reference to a material and prevents it from being garbage collected, a memory leak occurs.
- **Solution:** The useGSAP hook's context cleanup is critical here. It ensures that any GSAP-held references are released when the component unmounts.[3]

7.3. Browser Performance Monitoring

Production builds must include telemetry to monitor performance in the wild.

- **Tooling:** Use r3f-perf during development to monitor draw calls and shader compilation times.
- **Adaptive Quality:** Implement the <PerformanceMonitor> component from Drei. This component detects frame drops and can trigger a "fallback mode" (e.g., disabling expensive shadows or post-processing effects) to preserve usability on low-end devices.[33]

7.4. Post-Processing Optimization

Post-processing effects (Bloom, Depth of Field) are expensive.

- **Strategy:** Avoid full-screen passes where possible.
- **Selection:** If highlighting an object, use the Selection and EffectComposer selectively rather than applying effects globally. Use the Multisampling prop on <EffectComposer> with caution (0 is best for performance).[34]

---

8. Detailed Use Cases and Integration Scenarios

To further illustrate the complementary nature of these libraries, we define three core interaction patterns required for the React Bits project.

8.1. Use Case: The "Exploded View" Scroll

Objective: As the user scrolls down a product page, a 3D model of a mechanical watch disassembles into its components.

- **GSAP (3D):** A timeline is created where the x/y/z positions of the watch gears are tweened away from the center. This timeline is hooked to ScrollTrigger with scrub: 1. The "scrub" value introduces a 1-second delay, giving the explosion a weighty, physics-like feel.
- **Motion (2D):** Text labels describing each gear are positioned absolutely over the canvas. They use initial={{ opacity: 0 }}.
- **The Bridge:** The GSAP timeline has callbacks (onStart, onReverseComplete) or "stagger" points. When the timeline reaches a specific label's timestamp, it calls a Zustand action setActiveGear('gear-1'). The Motion component listening to activeGear triggers an animate={{ opacity: 1, y: 0 }} transition.

8.2. Use Case: The Seamless Page Transition

Objective: Clicking a 3D object navigates to a new URL, but the object appears to persist and expand.

- **Motion (2D):** The layoutId prop is used on the container of the Canvas. When the route changes, Motion calculates the transform difference between the "thumbnail" container and the "hero" container and animates the DOM wrapper.
- **GSAP (3D):** Simultaneously, a useEffect on the destination page triggers a GSAP tween that matches the camera's position to the new "hero" angle.
- **Result:** The user perceives a single continuous environment, despite a React Router navigation event occurring.

8.3. Use Case: The Interactive Cursor

Objective: A custom cursor that magnetically snaps to 3D objects and 2D buttons.

- **Implementation:** The cursor is a DOM element managed by Motion (using useMotionValue for high-performance updates without re-renders).
- **3D Interaction:** R3F events (onPointerOver) on meshes update the shared Zustand store state to cursorState: 'hover-3d'.
- **2D Interaction:** Motion components update the store to cursorState: 'hover-ui'.
- **Feedback:** The Motion cursor component subscribes to this state, animating its scale and mix-blend-mode based on the context (3D vs 2D).[6]

---

9. Conclusion

The "React Bits" Complementary Animation Libraries module is designed to withstand the scrutiny of modern web standards. By rejecting the "one tool fits all" approach and instead adopting a specialized stack—**Motion for Interface, GSAP for Scene, Lenis for Scroll, and Zustand for State**—the architecture maximizes the strengths of each library while mitigating their weaknesses.

This strict separation of concerns ensures that:

1. **Performance is preserved:** 3D logic runs off the React render cycle.
2. **Maintainability is high:** UI developers can work in standard React/Motion without needing deep WebGL knowledge.
3. **Stability is guaranteed:** The reliance on useGSAP and modern R3F patterns prevents the memory leaks and race conditions that plague lesser implementations.

This document serves as the blueprint for all future development on the animation module. Adherence to these specifications is mandatory for maintaining the project's "Fortune 500" classification.

Works cited

1. Motion for React — Install & first React animation | Motion, accessed January 1, 2026, https://motion.dev/docs/react-quick-start
2. Framer Motion is now independent. Introducing Motion. : r/reactjs - Reddit, accessed January 1, 2026, https://www.reddit.com/r/reactjs/comments/1gqadrv/framer_motion_is_now_independent_introducing/
3. React & GSAP | GSAP | Docs & Learning, accessed January 1, 2026, https://gsap.com/resources/React/
4. How to use state management with react-three-fiber without performance issues, accessed January 1, 2026, https://discourse.threejs.org/t/how-to-use-state-management-with-react-three-fiber-without-performance-issues/61223
5. pmndrs/zustand: Bear necessities for state management in React - GitHub, accessed January 1, 2026, https://github.com/pmndrs/zustand
6. Working with lenis and R3F : r/nextjs - Reddit, accessed January 1, 2026, https://www.reddit.com/r/nextjs/comments/1hfuywi/working_with_lenis_and_r3f/
7. Lenis smooth scrolling (GSAP) does not work - Stack Overflow, accessed January 1, 2026, https://stackoverflow.com/questions/78017041/lenis-smooth-scrolling-gsap-does-not-work
8. pmndrs/react-three-fiber: A React renderer for Three.js - GitHub, accessed January 1, 2026, https://github.com/pmndrs/react-three-fiber
9. framer-motion-3d - NPM, accessed January 1, 2026,

https://www.npmjs.com/package/framer-motion-3d?activeTab=versions
10. framer-motion-3d - Yarn Classic, accessed January 1, 2026, https://classic.yarnpkg.com/en/package/framer-motion-3d
11. React 19 compatibility issue due to older framer-motion version #383 - GitHub, accessed January 1, 2026, https://github.com/nandorojo/moti/issues/383
12. [BUG] Incompatible with React 19 · Issue #2668 · motiondivision/motion - GitHub, accessed January 1, 2026, https://github.com/motiondivision/motion/issues/2668
13. Motion & Framer Motion upgrade guide, accessed January 1, 2026, https://motion.dev/docs/react-upgrade-guide
14. GSAP vs Framer Motion for React - YouTube, accessed January 1, 2026, https://www.youtube.com/watch?v=_1QAJaC6Xwc
15. Troubleshooting Animation Issues - Framer Help, accessed January 1, 2026, https://www.framer.com/help/articles/troubleshooting-animation-issues/
16. How to add mose events to - Questions - three.js forum, accessed January 1, 2026, https://discourse.threejs.org/t/how-to-add-mose-events-to-html/65494
17. How to position the React Three Fiber Drei Html as just a camera overlay? : r/reactjs - Reddit, accessed January 1, 2026, https://www.reddit.com/r/reactjs/comments/tpnb1c/how_to_position_the_react_three_fiber_drei_html/
18. React Three Fiber with GSAP to trigger scroll down sphere rotation, accessed January 1, 2026, https://gsap.com/community/forums/topic/40877-react-three-fiber-with-gsap-to-trigger-scroll-down-sphere-rotation/
19. How do you animate the camera with react-three-fiber? - Stack Overflow, accessed January 1, 2026, https://stackoverflow.com/questions/75562296/how-do-you-animate-the-camera-with-react-three-fiber
20. ScrollTrigger | GSAP | Docs & Learning, accessed January 1, 2026, https://gsap.com/docs/v3/Plugins/ScrollTrigger/
21. Scroll Driven presentation in Three.js with GSAP | by Bandinopla - Medium, accessed January 1, 2026, https://medium.com/@pablobandinopla/scroll-driven-presentation-in-threejs-with-gsap-a2be523e430a
22. [React Three Fiber] + GSAP to create animation - Questions, accessed January 1, 2026, https://discourse.threejs.org/t/react-three-fiber-gsap-to-create-animation/22038
23. Pattern(s) for synchronizing ScrollTrigger and Lenis in React/Next - GSAP, accessed January 1, 2026, https://gsap.com/community/forums/topic/40426-patterns-for-synchronizing-scrolltrigger-and-lenis-in-reactnext/
24. darkroomengineering/lenis: Smooth scroll at it should be - GitHub, accessed January 1, 2026, https://github.com/darkroomengineering/lenis
25. — A demo to evaluate react-three-fiber and its ecosystem | by Thomas - Thomas Rutzer - Medium, accessed January 1, 2026, https://thomasrutzer.medium.com/a-demo-to-evaluate-react-three-fiber-and-its

-ecosystem-84c52df4621c
26. Next.js R3F TypeScript Boilerplate | ThreeJS Marketplace, accessed January 1, 2026, https://threejsresources.com/marketplace/next.js-r3f-typescript-boilerplate
27. Best Folder Structure for Scalable React Apps: A Complete Step-by-Step Guide, accessed January 1, 2026, https://dev.to/codelamda/best-folder-structure-for-scalable-react-apps-a-complete-step-by-step-guide-p83
28. How to animate a React Three Fiber "group" object with NEW framer motion?, accessed January 1, 2026, https://stackoverflow.com/questions/79460727/how-to-animate-a-react-three-fiber-group-object-with-new-framer-motion
29. View - React Three Drei, accessed January 1, 2026, https://drei.docs.pmnd.rs/portals/view
30. Using View from drei decreases performance - Questions - three.js forum, accessed January 1, 2026, https://discourse.threejs.org/t/using-view-from-drei-decreases-performance/67831
31. multiple canvas in a component · pmndrs react-three-fiber · Discussion #2716 - GitHub, accessed January 1, 2026, https://github.com/pmndrs/react-three-fiber/discussions/2716
32. The reason for the very low performance of R3F with instances ? · Issue #3306 · pmndrs/react-three-fiber - GitHub, accessed January 1, 2026, https://github.com/pmndrs/react-three-fiber/issues/3306
33. Scaling performance - React Three Fiber, accessed January 1, 2026, https://r3f.docs.pmnd.rs/advanced/scaling-performance
34. Highlight selected geometries in Drei View with @react-three/postprocessing - Questions, accessed January 1, 2026, https://discourse.threejs.org/t/highlight-selected-geometries-in-drei-view-with-react-three-postprocessing/58441
35. Build a 3D Portfolio with React Three Fiber - Framer Motion Scroll Animations, accessed January 1, 2026, https://wawasensei.dev/tuto/build-a-3D-portfolio-with-react-three-fiber-framer-motion-scroll-animations