The 2026 Scroll Architecture: Reverse-Engineering High-Performance Interactive Stacks

1. Introduction: The Bifurcation of the Interactive Web

By early 2026, the landscape of web interactivity has undergone a radical transformation, characterized not by a single dominant technology, but by a sophisticated bifurcation of the "Scroll Stack." The monolithic JavaScript libraries that defined the early 2020s—where a single dependency managed scrolling, triggering, and animation rendering—have given way to a modular, hybrid ecosystem. This shift is driven by the browser vendors' aggressive optimization of the "Pixel Pipeline" and the maturation of native APIs such as CSS Scroll-Driven Animations and WebCodecs.[1]

The modern web architect designing for award-winning experiences (typified by the work of Apple, Linear, and Refokus) no longer asks, "Which library should I use?" Instead, the critical question of 2026 is, "Which thread should this interaction occupy?" The "Gold Standard" for performance is now defined by the invisibility of cost to the user's device. Animations that trigger layout shifts or dominate the main thread are viewed as technical debt, while "Composite-Only" animations—those handled exclusively by the GPU—are the baseline expectation for production-grade interfaces.[3]

This report provides an exhaustive technical analysis of this new architecture. It reverse-engineers the specific patterns enabling "scrollytelling," sticky stacking cards, and frame-perfect video scrubbing, dissecting the performance trade-offs between the **Main Thread** (JavaScript execution, Layout calculations) and the **Compositor Thread** (GPU acceleration). The findings indicate that while **Lenis** remains the dominant solution for normalizing scroll input, the mechanism for *triggering* animations is shifting toward native CSS scroll-timeline for UI transitions, reserving **GSAP ScrollTrigger** for complex, narrative sequencing.[5] Furthermore, the traditional "Image Sequence" canvas hack for video scrubbing is being aggressively replaced by **WebCodecs**, which allows for hardware-accelerated, memory-efficient video frame extraction.[7]

---

2. The Smooth Scroll Debate in 2026: Input Normalization vs. Native Behavior

The foundational layer of any high-end scroll experience is the scroll behavior itself. The debate between "scroll-jacking" (replacing native scroll with virtual translation) and

"scroll-smoothing" (interpolating native scroll values) has largely settled in favor of the latter. However, the ecosystem remains divided between heavy, immersive "Brand" experiences and lightweight, utility-driven "Product" interfaces.

2.1 The Dominance of Lenis: The "Brand" Standard

As of 2026, Lenis has cemented its position as the preferred smooth scroll library for award-winning sites, effectively replacing heavier predecessors like Locomotive Scroll.[8] The architectural distinction is critical: Locomotive Scroll traditionally relied on translating a fixed container (virtual scrolling), which often broke native browser features like position: sticky, cmd+f (find on page), and accessibility tools. Lenis, conversely, operates by interpolating the native scroll position, preserving the document flow.[6]

2.1.1 The Mechanics of Interpolation (Lerp)

Lenis functions by intercepting the wheel or touch events and applying a damping algorithm—typically Linear Interpolation (Lerp)—to the scroll position.[11] The core mathematical concept driving this "luxurious" feel is the delay between the *target* scroll position (where the user's input says the page should be) and the *current* scroll position (where the viewport actually is).

In a typical 2026 implementation, the update loop runs on requestAnimationFrame:

1. **Event Capture:** The library listens for wheel events, calculating the deltaY to determine the target position.
2. **Damping:** Instead of jumping to the target, the current position is updated by a fraction of the distance to the target: current = current + (target - current) * friction.
3. **Normalization:** Lenis normalizes input across devices. A trackpad on macOS (which has its own physics) generates different delta values than a generic mouse wheel on Windows. Lenis homogenizes these inputs, ensuring that a "100px" scroll feels consistent regardless of the hardware.[12]

2.1.2 Main Thread Cost and Performance

While efficient (~4KB minified), Lenis runs on the main thread. This presents a "Medium" performance cost. Every scroll event triggers JavaScript execution. If the main thread is blocked by other heavy tasks (e.g., React hydration or large data parsing), the scroll can stutter. However, its impact is minimized because it does not force layout recalculations (reflows) as aggressively as virtual scrollers. It simply sets the scroll position of the window or container.[6]

2026 Integration with GSAP: Lenis effectively acts as "middleware" for GSAP ScrollTrigger. By normalizing the scroll velocity before GSAP reads the value, it ensures that scrubbed animations feel weighty and fluid rather than jittery. This synergy is why the Lenis + GSAP

stack remains the default for high-budget sites like Refokus.[9]

2.2 The "Lightweight" Alternatives & The Linear Approach

While Lenis dominates immersive sites, utility-focused sites—typified by Linear, Vercel, and modern SaaS dashboards—are increasingly rejecting JS-based smoothing in favor of "pure" or "near-native" approaches. The philosophy here is **Zero Latency**.

2.2.1 Trig.js and Motion One

Emerging libraries like Trig.js (approx. 4KB) and Motion One offer lighter alternatives to the Lenis/GSAP heavyweight combo.[8] These libraries focus on utilizing the Web Animations API (WAAPI) and CSS variables to drive motion.

- **Motion One:** Leverages hardware acceleration by using the browser's scroll() function to drive animations on the compositor thread. This bypasses the main thread entirely for the animation itself, even if the trigger logic remains in JS.[3]
- **Trig.js:** Focuses on a CSS-first approach, toggling classes or updating variables based on scroll position, rather than managing a complex timeline engine.[14]

2.2.2 The "Vibe" Stack (Linear)

Linear's homepage rebuilds demonstrate a preference for native behavior overlaid with high-performance transforms. By using CSS transforms triggered by IntersectionObserver or minimal scroll listeners, they achieve 60fps (or 120fps on ProMotion displays) performance. The key distinction is that Linear *does not* smooth the scroll itself; they allow the native OS physics to dictate the viewport movement, only smoothing the *elements* responding to that movement.[16]

2.3 Accessibility and Touch Normalization in 2026

A major regression in early smooth scroll libraries was the breaking of native mobile gestures and accessibility tools. In 2026, compliance with WCAG 2.2 and the upcoming 3.0 standards is a strict requirement.[17]

2.3.1 The prefers-reduced-motion Mandate

High-end sites now implement a global "kill switch" or "dampener" for animations based on the user's OS settings. This is not optional; it is a core accessibility requirement.

- **Detection:** Developers use window.matchMedia('(prefers-reduced-motion: reduce)') to detect the preference.[19]
- **Implementation Strategy:**
  - **Lenis:** If reduce is true, the Lenis instance is often initialized with a lerp value of 1 (instant) or destroyed entirely, reverting to native scrolling.[19]
  - **GSAP:** ScrollTriggers are configured to complete immediately or switch to

opacity-only fades rather than large movement transforms. GSAP's matchMedia functionality allows for specific animation contexts that respect this preference automatically.[20]

2.3.2 Mobile Touch & The Rubber Band Effect

Early libraries often disabled touch events to implement custom physics, breaking "pull-to-refresh" and the native "rubber band" overscroll effect on iOS.

- **The 2026 Solution:** Modern implementations like Lenis utilize overscroll-behavior: none or auto selectively. They listen for touch start/move events to allow native propagation when at the top of the viewport. This ensures that the native browser chrome (address bar collapsing) and overscroll effects remain intact, preserving the organic feel of the device while still smoothing the internal content.[12]

2.3.3 Horizontal Scroll Accessibility

Horizontal scrolling within a vertical page (a common pattern) poses significant accessibility challenges, particularly for keyboard users and screen readers.

- **Focus Management:** The 2026 standard requires that focusable elements within a horizontal section remain accessible via the Tab key. Simply transforming a container off-screen does not remove it from the focus order.
- **WCAG Reflow:** WCAG 1.4.10 (Reflow) requires that content be viewable without two-dimensional scrolling at 400% zoom. Horizontal sections must typically switch to vertical stacking on mobile or high-zoom contexts to remain compliant.[23]

2.4 Comparative Architecture: Smooth Scroll Solutions

Feature	Lenis (2026)	Locomotive (Legacy)	Native CSS / Motion One
Scrolling Logic	Interpolated Native	Virtual Transform (TranslateY)	Native OS Physics
Main Thread Cost	**Medium** (JS Loop)	**High** (Layout & JS)	**Zero / Low**
position: sticky	Supported Natively	Requires Hacks / Polyfills	Supported Natively
Accessibility	Good (Configurable)	Poor (Breaks flow)	Excellent (Native)

Touch Gestures	Preserved (Configurable)	Often Hijacked	Native
Primary Use Case	Immersive Storytelling	Deprecated / Legacy	SaaS / Utility / Product

---

3. The Orchestration Engine: GSAP vs. CSS Scroll-Timeline

Once the scroll input is normalized, the next architectural layer is the *trigger*. How does the scroll position drive animation? The industry has moved from a "GSAP for everything" approach to a nuanced selection based on the complexity of the requirement.

3.1 GSAP ScrollTrigger: The Heavy Lifter for Narrative

GSAP (GreenSock Animation Platform) remains the industry standard for complex, sequenced choreographies. Its ScrollTrigger plugin allows for precise pinning, scrubbing, and timeline management that CSS alone cannot yet match.[26]

3.1.1 The Mechanism of Layout Thrashing

The primary performance killer in scroll animation is "layout thrashing" (or forced synchronous layout). This occurs when JavaScript reads a layout property (like offsetHeight, scrollTop, or getBoundingClientRect()) and then writes a style (like height or width) in the same frame.

- **The Cycle:**
  1. **Read:** JS asks "How tall is this div?" Browser calculates layout.
  2. **Write:** JS says "Make it 10px taller." Browser invalidates layout.
  3. **Read (Again):** JS asks "How tall is it now?" Browser must *immediately* recalculate layout (Reflow) to answer, pausing the JS execution.
- **Performance Impact:** If this happens inside a scroll loop (which runs 60-120 times per second), it creates significant jank.[26]

3.1.2 2026 Best Practices for GSAP Performance

To mitigate thrashing and main-thread blocking, expert implementations in 2026 employ specific strategies:

1. **Batching and Up-Front Calculation:** GSAP ScrollTrigger calculates start/end positions *once* (on load or refresh), not on every tick. It caches these values to avoid constant DOM reads.[26]
2. **fastScrollEnd:** On long scrollytelling pages, if a user flicks the scrollbar rapidly, the engine might try to render thousands of intermediate frames. The fastScrollEnd

configuration forces the animation to jump immediately to its final state if the scroll velocity exceeds a threshold (default 2500px/s). This prevents the CPU from melting down trying to catch up.[28]

3. **refreshPriority:** When using pinning, the "start" position of lower elements depends on the "pinned" duration of upper elements. If these are calculated out of order, the layout breaks. Setting refreshPriority ensures that triggers are calculated in strict top-to-bottom order.[28]
4. **clamp() for Boundaries:** To prevent logic errors where triggers fire past the viewport bounds (common with elastic scrolling on macOS), wrapping start/end values in clamp() ensures they stay within valid ranges.[26]

3.2 CSS Scroll-Driven Animations: The Production-Ready Challenger

As of February 2026, **Safari 26.0** and **Firefox** (via Interop 2026) have shipped stable support for CSS Scroll-Driven Animations (scroll-timeline, view-timeline).[2] This marks a watershed moment where purely declarative CSS can replace JavaScript for scroll interactions.

3.2.1 The Technical Specification

The API consists of two primary timelines:

1. **scroll():** Tracks the scroll progress of a container (0% to 100%).
2. **view():** Tracks the visibility of a specific element within the viewport (similar to IntersectionObserver but continuous).

Example Syntax (2026 Standard):

CSS

```
@keyframes reveal-card {
  entry 0% { opacity: 0; transform: translateY(50px); }
  entry 100% { opacity: 1; transform: translateY(0); }
  exit 0% { opacity: 1; transform: scale(1); }
  exit 100% { opacity: 0; transform: scale(0.9); }
}

.card {
  animation: reveal-card linear both;
  animation-timeline: view();
  animation-range: entry 10% cover 90%;
}
```

3.2.2 Performance Profile: The Compositor Thread Advantage

The decisive advantage of CSS Scroll-Driven Animations is that they run off the **Main Thread**.

- **Mechanism:** The browser's compositor thread handles the animation. Even if the main thread is completely blocked (frozen) by heavy JavaScript execution, the scroll animation continues to render smoothly. This provides a level of responsiveness that GSAP, running on the main thread, effectively cannot guarantee under heavy load.[3]
- **Adoption:** Utility-focused sites (Linear, Vercel) utilize this for "reveal" animations, reserving JS for complex logic.

3.3 Comparative Architecture: Orchestration

Feature	GSAP ScrollTrigger	CSS Scroll-Timeline
Execution Thread	**Main Thread** (JS)	**Compositor Thread** (GPU)
Logic Complexity	High (Pinning, Scrubbing, Callbacks)	Low/Medium (Reveals, Parallax)
Browser Support	Universal (Back to IE with polyfills)	Modern Only (Safari 26+, Chrome, FF)
State Management	External (Can toggle classes/state)	Visual Only (Cannot affect logic)
2026 Verdict	**Essential** for Storytelling	**Default** for UI Transitions

---

4. Complex Pattern Analysis: Reverse-Engineering the Stack

This section deconstructs three specific "high-value" patterns found on sites like Refokus and Apple, providing the logic and architectural decisions behind them.

4.1 Pattern: Sticky Stacking Cards

Visual: A vertical scroll where cards enter from the bottom, stick to the top (or a precise offset), and subsequent cards slide over them, creating a stack. **Reference:** Linear’s project views, Refokus portfolio.[32]

4.1.1 The "Grid Overlay" Technique (2026 Best Practice)

A common novice mistake is using position: absolute and manually calculating top values using JavaScript. This removes elements from the document flow, making the container height collapse and requiring expensive manual height calculations. The expert pattern in 2026 utilizes **CSS Grid** for structural stacking and GSAP for the visual reveal.[33]

Implementation Logic:

1. **Container Setup:** The parent container is set to display: grid with a single grid cell: grid-template-areas: "stack".
2. **Cell Assignment:** *All* card elements are assigned to the same grid area (grid-area: stack). This naturally places them one on top of another in the document flow (z-axis stacking) without removing them from the DOM flow like absolute positioning would.
3. **Spacer Management:** To create the scrollable distance, a separate spacer element or padding is used, or the cards are allowed to naturally stack and then are pinned.
4. **Animation (.from() Logic):** The brilliance of this pattern lies in the animation direction. Instead of calculating where a card needs to go (.to()), the stack is rendered in CSS in its *final* state (all cards stacked). GSAP then animates them .from() a yPercent: 100 (off-screen bottom) based on the scroll progress.
    - **Advantages:** This is a "progressive enhancement" approach. If JavaScript fails or is blocked, the user simply sees a static stack of cards, which is still readable. The layout remains robust.[34]
    - **Performance:** The animation uses transform: translateY, a compositor-only property. No top or margin properties are animated, preventing Layout Thrashing.[29]

Performance Impact Rating: Low/Medium. The cost depends heavily on the complexity of the card content (shadows, blurs).

4.2 Pattern: Frame-Perfect Video Scrubbing (WebCodecs)

Visual: A 3D product render (e.g., AirPods case opening) that rotates precisely with the scrollbar, with zero latency and high fidelity. **Reference:** Apple Product Pages, Refokus implementations.[7]

4.2.1 The Evolution from Canvas Hacks

- *2020-2024 (The "Image Sequence" Era):* Developers would export a video as 300 individual JPEG or WebP images. JavaScript would listen to scroll events and draw image_001.jpg, then image_002.jpg to a <canvas>.
    - *Flaws:* Massive network payload (often 10MB+), high number of HTTP requests (unless spritesheets were used), and high main-thread CPU usage for image decoding.[7]
- *2025-2026 (The WebCodecs Era):* The standard has shifted to using the **WebCodecs API** to manually decode video frames.

4.2.2 The WebCodecs Architecture

This technique uses a single video file (MP4/WebM) but bypasses the HTML <video> element's unreliable currentTime seeking.

1. **Demuxing:** A lightweight library (like mp4box.js or mediabunny) is used to fetch the video file and "demux" (separate) the video track from the container.[7]
2. **Decoding:** The compressed video chunks are fed into the VideoDecoder API. This gives the developer direct access to VideoFrame objects—raw bitmaps residing in GPU memory.
3. **GOP (Group of Pictures) Optimization:** Standard video compression uses "Inter-frame" compression, where a full image (I-frame) is followed by many partial updates (P-frames/B-frames). Standard videos might have an I-frame only every 250 frames (every 10 seconds).
    - *The Scroll Problem:* To scrub to frame 150, the decoder must start at frame 0 and decode all 149 subsequent frames to build the image. This causes massive lag.
    - *The Encoding Solution:* For scroll-scrubbing, videos must be encoded with a **Short GOP**. Keyframes (I-frames) are placed every **15-30 frames** (or even less). This increases file size by 10-20% but makes random access seeking nearly instantaneous.[36]
    - *Codec Choice:* **AV1** is the 2026 preference due to superior compression efficiency (30-50% better than H.264) and widespread hardware decoding support on modern devices. H.264 remains a fallback for legacy hardware.[37]
4. **Ring Buffer:** The JS implementation maintains a "sliding window" or ring buffer of decoded VideoFrame objects (e.g., ±1 second around the current scroll timestamp) kept ready in GPU memory. As the user scrolls, the matching frame is painted to a WebGL or 2D Canvas immediately.
    - *Memory Management:* VideoFrame objects must be manually closed (frame.close()). Failure to do so results in rapid GPU memory exhaustion and page crashes.[7]

Performance Impact Rating: Medium. High efficiency compared to image sequences, but GPU memory intensive. Requires rigorous memory management.

4.3 Pattern: Seamless Horizontal Scroll (in Vertical Layout)

Visual: The user scrolls down, the viewport locks (pins), and content moves horizontally (left-to-right), then unlocks and continues vertical scrolling. **Reference:** Portfolio galleries, Timelines.[40]

4.3.1 The Logic: Pin and Translate

1. **The Pin:** A wrapper container is pinned (pin: true) using GSAP ScrollTrigger for a duration defined by the width of the horizontal content (e.g., end: "+=3000"). This creates a "virtual" scroll space where the user is scrubbing time/distance without moving vertically.

2. **The Translation:** Inside the pinned container, the "filmstrip" of content is animated on the **x-axis** using xPercent: -100 * (sections.length - 1).
3. **Ease:** The animation ease must be set to ease: "none". Any other easing (like power1.out) creates a disconnect between the scrollbar movement and the content movement, breaking the physical illusion of direct control.[40]

4.3.2 Accessibility and Touch Gestures

- **The Trap:** Horizontal scrolling on a vertical page is often flagged as an accessibility violation (WCAG 1.4.10 Reflow) if not handled carefully.[23]
- **Touch Solution:** On desktop, the vertical mouse wheel drives the horizontal movement via GSAP. On mobile/touch devices, users naturally swipe *horizontally*.
    - **Implementation:** The code must detect touch input. A common solution is using a "Proxy" or scrollerProxy in GSAP. When a horizontal swipe is detected on the pinned section, it must be translated into vertical scroll progress for the window, or the animation must respond to *both* inputs. If not handled, the user gets stuck: swiping up/down doesn't move the page (because it's pinned), and swiping left/right doesn't move the content (because GSAP is listening to vertical scroll).[40]

Performance Impact Rating: Medium. Pinning often requires position: fixed, which creates a new composite layer. Large horizontal strips can create massive texture layers that consume significant video memory.

---

5. Performance Architecture Guide: The 2026 Pixel Pipeline

To build these experiences without degrading user experience, architects must adhere to the rules of the **Pixel Pipeline**. This section rates the performance impact of various techniques based on which stage of the pipeline they trigger.

5.1 The Pipeline Stages

1. **JavaScript / Main Thread:** Handling events, calculating logic.
2. **Style:** Calculating CSS rules.
3. **Layout (Reflow):** Calculating geometry (width, height, position) of elements. **(Most Expensive)**
4. **Paint:** Filling in pixels (backgrounds, text, shadows). **(Expensive)**
5. **Composite:** arranging layers on the GPU. **(Cheapest)**.[3]

5.2 Performance Impact Ratings

Technique	Implementation Stack	Performance Cost	Layout Thrashing Risk	2026 Verdict
Smooth Scrolling	Lenis	**Medium** (Main Thread)	Low	**Essential** for high-brand sites; overkill for SaaS.
Simple Reveals	CSS view-timeline	**Low** (Compositor)	Zero	**Default**. Replace JS for opacity/transform reveals.
Sequenced Story	GSAP ScrollTrigger	**Medium/High**	Medium	**Required** for complex orchestration. Use fastScrollEnd.
Video Scrubbing	WebCodecs (AV1)	**Medium** (GPU Memory)	Low	**Standard**. Replaces Image Sequences. Requires short GOP.
Sticky Cards	CSS Grid + Transform	**Low/Medium**	Low	**Standard**. Avoid top/left animation; use translate.
Horizontal Scroll	GSAP Pin + Transform	**Medium**	Medium	Use sparingly. Ensure touch gesture support.

5.3 The Golden Rules of 2026

1. **Animate Only Composite Properties:** Only animate transform (translate, scale, rotate),

opacity, and filter (sometimes). Never animate width, height, top, left, margin, or padding during a scroll. Animating width triggers **Layout** (Reflow) for the whole document. Animating scaleX triggers only **Composite**.[3]

2. **Lazy Loading via ScrollTrigger:** High-performance sites like Apple don't just use scroll triggers for animation. They use them for data fetching.
    - *Technique:* A start: "top 200%" trigger (2 viewports away) initiates the fetch/decode of a WebCodecs video or high-res texture. This ensures the heavy asset is ready in memory exactly when the user arrives, preventing the "pop-in" effect.[28]
3. **Will-Change Usage:** Use will-change: transform sparingly. It tells the browser to promote an element to its own layer (GPU). Overusing it consumes excessive video memory. Apply it only to elements that are currently animating or about to animate.[3]

---

6. Conclusion: The "Hybrid" Future

In 2026, the "Scroll-Driven Animation" stack is no longer about choosing *one* library. It is about architectural stratification. The most successful sites employ a layered approach:

1. **Base Layer (CSS):** Handle all simple entry/exit animations, hover states, and layout (Grid/Flex) using native CSS and view-timeline. This ensures the site remains functional and fast even if JS hangs.
2. **Middle Layer (Lenis):** Normalize the input. Provide a consistent "feel" across mouse, trackpad, and touch, but respect prefers-reduced-motion to remain accessible.
3. **Top Layer (GSAP/WebCodecs):** Inject the "Magic." Use GSAP strictly for orchestration (pinning, timing) and WebCodecs for high-fidelity media that requires frame-accurate control.

The sites winning awards in 2026 succeed not because they have *more* animation, but because they have rigorously engineered the *cost* of that animation, ensuring the browser spends its time painting pixels, not recalculating layouts.

Report compiled based on technical analysis of 2026 web standards and community discussions. Sources:.[1]

Referências citadas

1. CSS in 2026: The new features reshaping frontend development ..., acessado em fevereiro 16, 2026, https://blog.logrocket.com/css-in-2026/
2. Launching Interop 2026 - Mozilla Hacks - the Web developer blog, acessado em fevereiro 16, 2026, https://hacks.mozilla.org/2026/02/launching-interop-2026/
3. The Web Animation Performance Tier List - Motion Magazine, acessado em fevereiro 16, 2026, https://motion.dev/magazine/web-animation-performance-tier-list
4. Towards an animation smoothness metric | Articles | web.dev, acessado em

fevereiro 16, 2026, https://web.dev/articles/smoothness
5. Announcing Interop 2026 | WebKit, acessado em fevereiro 16, 2026, https://webkit.org/blog/17818/announcing-interop-2026/
6. darkroomengineering/lenis: Smooth scroll at it should be - GitHub, acessado em fevereiro 16, 2026, https://github.com/darkroomengineering/lenis
7. A Tutorial: WebCodecs Video Scroll Synchronization | by Keng Lim, acessado em fevereiro 16, 2026, https://lionkeng.medium.com/a-tutorial-webcodecs-video-scroll-synchronization-8b251e1a1708
8. Best AOS (Animate on Scroll) Libraries in 2025 - DEV Community, acessado em fevereiro 16, 2026, https://dev.to/idevgames/best-aos-animation-on-scroll-libraries-in-2025-c9o
9. Smooth Scroll Meditation: GSAP ScrollSmoother vs Lenis - Zun Creative, acessado em fevereiro 16, 2026, https://zuncreative.com/blog/smooth_scroll_meditation/
10. Smooth Scroll Website - The Setup for Beginners - Kontra Agency, acessado em fevereiro 16, 2026, https://kontra.agency/smooth-scroll-the-setup-for-beginners/
11. Why Lenis Smooth Scroll needs to become a browser standard. | by Apnatva - Medium, acessado em fevereiro 16, 2026, https://medium.com/@nattupi/why-lenis-smooth-scroll-needs-to-become-a-browser-standard-62bed416c987
12. Lenis – Get smooth or die trying, acessado em fevereiro 16, 2026, https://lenis.darkroom.engineering/
13. Use ScrollSmoother features with lenis scroll - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/36030-use-scrollsmoother-features-with-lenis-scroll/
14. Trig.js vs AOS.js vs ScrollTrigger – Which Scroll Animation Library Wins? - DEV Community, acessado em fevereiro 16, 2026, https://dev.to/idevgames/trigjs-vs-aosjs-vs-scrolltrigger-which-scroll-animation-library-wins-4i36
15. Trig.js vs. AOS.js vs. ScrollTrigger: Choosing the Right Scroll Animation Library - Kite Metric, acessado em fevereiro 16, 2026, https://kitemetric.com/blogs/trig-js-vs-aos-js-vs-scrolltrigger-choosing-the-right-scroll-animation-library
16. Part 2: Rebuilding Linear's Homepage with Next.js and Tailwind - YouTube, acessado em fevereiro 16, 2026, https://www.youtube.com/watch?v=R5PjNcIdAzU
17. New Digital Accessibility Requirements in 2026 - Best Best & Krieger LLP, acessado em fevereiro 16, 2026, https://bbklaw.com/resources/new-digital-accessibility-requirements-in-2026
18. WCAG 2.1 Accessibility Checklist - Pilot Digital Marketing, acessado em fevereiro 16, 2026, https://pilotdigital.com/blog/wcag-2-1-accessibility-checklist/
19. prefers-reduced-motion - CSS | MDN - Mozilla, acessado em fevereiro 16, 2026, https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@media/prefers-reduced-motion

20. Design accessible animation and movement with code examples - Pope Tech Blog, acessado em fevereiro 16, 2026, https://blog.pope.tech/2025/12/08/design-accessible-animation-and-movement/
21. Technique C39:Using the CSS prefers-reduced-motion query to prevent motion - W3C, acessado em fevereiro 16, 2026, https://www.w3.org/WAI/WCAG22/Techniques/css/C39
22. Take control of your scroll - customizing pull-to-refresh and overflow effects | Blog, acessado em fevereiro 16, 2026, https://developer.chrome.com/blog/overscroll-behavior
23. Technique G225:Section panels that scroll horizontally are designed to fit within a width of 320 CSS pixels on a vertically scrolling page - W3C, acessado em fevereiro 16, 2026, https://www.w3.org/WAI/WCAG22/Techniques/general/G225
24. WCAG 1.4.10: Reflow (Level AA) - Silktide, acessado em fevereiro 16, 2026, https://silktide.com/accessibility-guide/the-wcag-standard/1-4/distinguishable/1-4-10-reflow/
25. Consider accessibility when using horizontally scrollable regions in webpages and apps, acessado em fevereiro 16, 2026, https://cerovac.com/a11y/2024/02/consider-accessibility-when-using-horizontally-scrollable-regions-in-webpages-and-apps/
26. ScrollTrigger | GSAP | Docs & Learning, acessado em fevereiro 16, 2026, https://gsap.com/docs/v3/Plugins/ScrollTrigger/
27. ScrollTrigger Best Practices - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/32787-scrolltrigger-best-practices/
28. ScrollTrigger tips & mistakes | GSAP | Docs & Learning, acessado em fevereiro 16, 2026, https://gsap.com/resources/st-mistakes/
29. One page ScrollTrigger app - Best practice - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/44857-one-page-scrolltrigger-app-best-practice/
30. ScrollTrigger: Extremely laggy/jerky/poor performance on laptops only, especially Chrome, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/33489-scrolltrigger-extremely-laggyjerkypoor-performance-on-laptops-only-especially-chrome/
31. Scroll Animation Tools 2026: Which One Should You Actually Use?, acessado em fevereiro 16, 2026, https://cssauthor.com/scroll-animation-tools/
32. This Sticky Scroll Animation Unfolds Like a Mini Timeline of Cards (ScrollTrigger) - YouTube, acessado em fevereiro 16, 2026, https://www.youtube.com/watch?v=KE2NeszFQI0
33. ScrollTrigger stacking cards animation logic to create any effect, yes ..., acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/39367-scrolltrigger-stacking-cards-animation-logic-to-create-any-effect-yes-even-yours/
34. Scaling and sticky animation of boxes while scrolling - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/42981-scaling-and-sticky-animation-o

f-boxes-while-scrolling/
35. which is more performant, or with video embed? - Stack Overflow, acessado em fevereiro 16, 2026, https://stackoverflow.com/questions/30806905/which-is-more-performant-video-or-canvas-with-video-embed
36. Trying to understand GOP/keyframes : r/ffmpeg - Reddit, acessado em fevereiro 16, 2026, https://www.reddit.com/r/ffmpeg/comments/1gz72ky/trying_to_understand_gopkeyframes/
37. AV1 Codec Dominates Streaming Landscape as 2026 Begins, acessado em fevereiro 16, 2026, https://www.free-codecs.com/news/av1-codec-dominates-streaming-landscape-as-2026-begins.htm
38. Video Streaming with the AV1 Video Codec in Mobile Devices - Engineering at Meta, acessado em fevereiro 16, 2026, https://engineering.fb.com/wp-content/uploads/2025/09/Meta-AV1-White-Paper-FINAL.pdf
39. Video Codecs Explained: H.264, H.265, AV1 & VP9 - Ant Media Server, acessado em fevereiro 16, 2026, https://antmedia.io/video-codecs-streaming-guide/
40. Horizontal Scroll with Sticky Panels Effect - GSAP - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/35512-horizontal-scroll-with-sticky-panels-effect/
41. Horizontal Scrolling: A Guide to Implementation and Benefits - Lenovo, acessado em fevereiro 16, 2026, https://www.lenovo.com/us/en/glossary/horizontal-scrolling/
42. I want horizontal and vertical scroll sections together with scroll snapping - GSAP, acessado em fevereiro 16, 2026, https://gsap.com/community/forums/topic/39464-i-want-horizontal-and-vertical-scroll-sections-together-with-scroll-snapping/
43. darkroomengineering lenis · Discussions · GitHub, acessado em fevereiro 16, 2026, https://github.com/darkroomengineering/lenis/discussions
44. Enhance Webflow with Lenis: Smooth Scrolling Made Easy - Tilipman Digital, acessado em fevereiro 16, 2026, https://www.tilipmandigital.com/resource-center/webflow-development-guides/webflow-lenis-smooth-scrolling