1.1.3 — The “Animation Triad”: Timing, Motion, Meaning □□

Professional-looking animation isn’t primarily about “cool effects”—it’s about making the user *feel* that the interface is responsive, intentional, and clear. This lesson gives you a **repeatable decision framework** you can apply to *any* GSAP animation in WordPress/Bricks.

---

https://codepen.io/editor/ZcarecroW/pen/019da77a-6995-7501-9af8-4866e69b7f1c

---

1) Goal □

By the end of this page you will be able to:

1. **Diagnose** why an animation feels “off” using *Timing / Motion / Meaning*.
2. **Design** an animation intentionally (instead of guessing eases/durations).
3. **Implement** a tiny Bricks sandbox demo that lets you *feel* the triad by toggling settings.
4. Build the habit of writing a **1–2 line “meaning statement”** before animating.

---

2) The Triad (The Mental Model) □□

2.1 **Timing** = *When* and *how fast* things happen □

Timing includes:

1. **Duration**: how long a tween runs.
2. **Delay**: how long before it begins.

3. **Stagger**: spacing between repeated items (lists, cards).
4. **Rhythm**: consistency across the site (so the UI feels coherent).

What timing communicates:

1. *Fast* timing → responsiveness, precision, confidence (common in UI).
2. *Slower* timing → weight, calm, luxury (common in editorial / hero).
3. Inconsistent timing → "cheap" feeling because the site has no rhythm.

Practical rules of thumb (web UI):

1. Micro feedback (hover/press/focus):
   1. Usually **0.08–0.20s**
   2. Should feel instant, not "animated"
2. Component transitions (menu open, accordion, modal):
   1. Usually **0.25–0.60s**
3. Decorative/hero sequences:
   1. Often **0.6–1.2s**, but keep it purposeful

> If you're unsure: start at **0.35s** for UI transitions and adjust.

2.2 **Motion** = *What path/shape* the movement takes □□

Motion includes:

1. **Properties** you animate:
   1. Prefer `x/y`, `scale`, `rotate`, `opacity/autoAlpha`
   2. Avoid animating `top/left/width/height` unless you know why (layout + jank risk)
2. **Easing** (the velocity curve)
3. **Distance** (how far an element moves)
4. **Direction** (does it match the visual hierarchy and reading direction?)

What motion communicates:

1. Ease-out-ish motion → natural settling (common for UI entering)
2. Overshoot/bounce → playful or "toy-like" (use deliberately)
3. Too much distance → feels like the element is "traveling" instead of "appearing"

A simple motion quality checklist:

1. Does it feel like it has **mass**? (even a tiny bit)
2. Does it start/stop in a way that matches the product tone?
3. Does motion support the layout (not fight it)?

2.3 **Meaning** = *Why* the animation exists

Meaning is the most ignored—and the most “pro”—part.

Meaning includes:

1. **What the user learns** from the animation:
    1. “This opened.”
    2. “This is clickable.”
    3. “This is the next step.”
2. **What it prioritizes**:
    1. Which element is “primary” in the moment?
3. **What it prevents**:
    1. Confusion
    2. Misclicks
    3. Losing context during state changes

Meaning statement (habit): before animating, write:

1. **Trigger:** what causes it? (load / click / scroll)
2. **User benefit:** what clarity/feedback do they get?
3. **Constraint:** what must not happen? (motion sickness, layout shift, delay)

Example:

1. “When the modal opens, the backdrop fades in quickly to signal a new layer, then the dialog rises slightly to confirm focus moved to it—without shifting layout.”

If you can’t write the meaning statement, you’re probably adding motion “because it looks cool” (which is fine in a Dribbble shot, risky in production).

---

3) How the Triad Solves Real Problems (Common “It Feels Off” Diagnoses) □□

3.1 The animation feels **laggy** □□

Usually a **timing** issue (too slow, too much delay), or a **meaning** issue (feedback arrives too late).

Fixes:

1. Reduce delay (often to `0`).
2. Shorten duration by 20–40%.
3. Add a tiny *instant* cue:
    1. e.g. quick `autoAlpha` change first, then movement

3.2 The animation feels **floaty / cheap** □□

Usually a **motion** issue (ease doesn’t match UI), sometimes too much distance.

Fixes:

1. Use a more “UI” ease:
    1. `power2.out`, `power3.out`, `expo.out` (careful—expo can be dramatic)
2. Reduce travel distance:
    1. `y: 24` instead of `y: 80`

3.3 The animation feels **confusing** □□

Almost always a **meaning** issue.

Fixes:

1. Make the state change readable:
    1. Backdrop + dialog separation for overlays
2. Use direction to explain hierarchy:

    1. “New layer” often comes forward/up slightly
3. Don’t animate everything:
    1. Animate the *one* thing that clarifies what changed

3.4 The animation is “fine” but the site feels **inconsistent** □□

A **timing rhythm** problem (durations/eases vary randomly).

Fixes:

1. Define site motion tokens (you’ll formalize later in Chapter 14):
    1. 2–3 durations and 2–3 eases you reuse everywhere

---

4) Bricks Implementation (Sandbox Demo) □□

You’ll build a small section with:

1. A headline + paragraph + button.
2. A “Play” button that runs a reveal animation.
3. A “Mode toggle” that switches between:
    1. **UI mode** (snappy, restrained)
    2. **Hero mode** (slower, more cinematic)

4.1 Bricks structure (what to create)

Create a **Section** (or Container) and add:

1. A wrapper container with class: `triad-demo`
2. Inside it:
    1. Heading (H2) with class: `triad-title`
    2. Text (p) with class: `triad-text`
    3. Button row container with class: `triad-controls`
        1. Button 1: “Play” with class: `triad-play`
        2. Button 2: “Toggle mode” with class: `triad-toggle`

3. (Optional) small label span with class: `triad-mode`

Why these classes?
They're stable, readable, and easy to scope—perfect for WordPress where DOM can change.

4.2 Minimal CSS (Bricks → Page Settings → Custom CSS)

```
.triad-demo {
  max-width: 720px;
  margin: 0 auto;
  padding: 48px 24px;
}

.triad-controls {
  display: flex;
  gap: 12px;
  align-items: center;
  margin-top: 16px;
}

/* Optional: make the “mode” label subtle */
.triad-mode {
  margin-left: 8px;
  font-size: 14px;
  opacity: 0.75;
}
```

4.3 JavaScript (Bricks → Page Settings → Custom Code → Footer Scripts)

> Put it in **Footer** so the DOM exists when this runs (we'll formalize DOM-ready patterns in `2.1.1`)

```
(() => {
  // Scope everything to each .triad-demo instance (so duplicates on the page won’t collide).
  const demos = document.querySelectorAll(".triad-demo");
  if (!demos.length) return;

  demos.forEach((root) => {
    const title = root.querySelector(".triad-title");
    const text = root.querySelector(".triad-text");
    const playBtn = root.querySelector(".triad-play");
    const toggleBtn = root.querySelector(".triad-toggle");
    const modeLabel = root.querySelector(".triad-mode");

    if (!title || !text || !playBtn || !toggleBtn) return;

    // Two presets to FEEL the difference between "UI motion" and "Hero motion".
    const presets = {
      ui: {
        label: "UI mode",
        timing: { duration: 0.38, stagger: 0.06 },
        motion: { y: 18, ease: "power2.out" },
        meaning: "Fast clarity: content appears quickly with minimal travel."
      },
      hero: {
        label: "Hero mode",
        timing: { duration: 0.9, stagger: 0.12 },
        motion: { y: 42, ease: "power3.out" },
        meaning: "Cinematic emphasis: slower, longer travel, more presence."
      }
    };

    let mode = "ui";

    const updateModeUI = () => {
      if (modeLabel) {
        modeLabel.textContent = `${presets[mode].label} — ${presets[mode].meaning}`;
      }
      toggleBtn.setAttribute("aria-pressed", mode === "hero" ? "true" : "false");
    };

    // Build a function that returns a timeline, so it’s easy to reuse.
```

```
    const buildRevealTl = () => {
      const p = presets[mode];

      // Clear any inline transforms/opacity from previous runs (keeps repeat plays
consistent).
      gsap.set([title, text, playBtn], { clearProps: "all" });

      // Set initial state (meaning: they are "not yet here")
      gsap.set([title, text, playBtn], { autoAlpha: 0, y: p.motion.y });

      const tl = gsap.timeline();
      tl.to([title, text, playBtn], {
        autoAlpha: 1,
        y: 0,
        duration: p.timing.duration,
        ease: p.motion.ease,
        stagger: p.timing.stagger
      });

      return tl;
    };

    let activeTl = null;

    playBtn.addEventListener("click", () => {
      // Prevent double-firing overlapping timelines
      if (activeTl) activeTl.kill();
      activeTl = buildRevealTl();
    });

    toggleBtn.addEventListener("click", () => {
      mode = mode === "ui" ? "hero" : "ui";
      updateModeUI();
    });

    // Initial label
    updateModeUI();
  });
})();
```

5) Micro-Exercises (Prove You Understand the Triad) □□

Do these in order—each targets *one* part of the triad.

1. **Timing-only experiment (no motion changes):**
   1. Keep `y` and `ease` the same
   2. Change `duration` from `0.38` → `0.22`
   3. Observe: does it feel more “responsive” or more “harsh”?
2. **Motion-only experiment (same duration):**
   1. Keep duration constant
   2. Change ease:
      1. `power2.out` → `power1.out` (more linear feel)
      2. `power2.out` → `expo.out` (more dramatic snap)
   3. Describe in one sentence what changed emotionally.
3. **Meaning experiment (change what’s animated):**
   1. Animate only the **title** and leave text/button static
   2. Ask: is the state change still clear? What did you lose?
4. **Rhythm experiment (site consistency):**
   1. Pick one duration + ease you like for UI:
      1. e.g. `duration: 0.35`, `ease: power2.out`
   2. Write it down somewhere as your first “motion token”.

---

6) Troubleshooting (Common Failure Modes) □□

1. **Nothing happens when clicking “Play”**
   1. Confirm GSAP is actually loaded on the page:
      1. In DevTools Console, run:

         ```
         typeof gsap
         ```

         It should return `"function"` or `"object"` (not `"undefined"`).
   2. Confirm your class names match exactly:
      1. `.triad-demo`, `.triad-title`, `.triad-text`, `.triad-play`, `.triad-toggle`
2. **Only the first demo works (if you duplicated the section)**

    1. This lesson’s code scopes per `.triad-demo` instance—so it *should* work.
    2. If it doesn’t, you likely reused IDs or targeted global selectors elsewhere.
3. **Elements jump / flicker**
    1. Ensure the initial state is set *before* animating:
        1. We use `gsap.set(...autoAlpha: 0, y: ...)` for that purpose.
    2. If you have CSS transitions on these elements, remove them (CSS transitions can fight GSAP).
4. **The mode label doesn’t show**
    1. You didn’t add the optional `.triad-mode` element—either add it, or ignore it (the demo still works).

---

7) Verification Checklist (Do This Every Lesson) □

1. **Functional**
    1. Click Play: does it animate?
    2. Click Toggle mode then Play: does it feel noticeably different?
2. **Hard refresh**
    1. Reload with cache bypass (your browser’s hard reload) and retest.
3. **Two-instance test**
    1. Duplicate the `.triad-demo` section in Bricks
    2. Confirm both work independently
4. **Basic accessibility sanity**
    1. Toggle button has `aria-pressed` updating (it does)
    2. Nothing becomes invisible-but-clickable (we use `autoAlpha`, which toggles visibility)

---

8) Your Output (What to Save) □□

1. A short note in your project (or a text file) answering:
    1. **Which preset felt better** for UI?
    2. What duration/ease did you choose as your first “token”?
2. Save this snippet as:
    1. `pages/01-01-03-animation-triad.js`
3. Add a header comment at the top of the file like:
    1. What it does
    2. Where it’s used (Bricks sandbox page)
    3. Dependencies: GSAP core

Revision #2
Created 2026-04-19 19:36:16 UTC by art10m
Updated 2026-04-19 20:59:36 UTC by art10m