The Unofficial Guide to GreenSock (GSAP) Animation

A Comprehensive Technical Manual for
Modern Web Animation

Prepared for Developers and Designers

Note on Terminology:
This book addresses ”GreenSock” (GSAP), the industry-standard JavaScript animation library.
This is often the intended subject when the term ”Green Stock” is used in an animation context.

December 5, 2025

Contents

Chapter 1

Introduction to GSAP

1.1 What is GSAP?

The GreenSock Animation Platform (GSAP) is a robust JavaScript toolset that turns developers into animation superheroes. It solves the biggest problem in web animation: compatibility. While CSS animations are great for simple transitions, they fail when you need complex sequencing, precise timing control, or compatibility with older browsers.

GSAP is framework-agnostic. Whether you use React, Vue, Angular, or vanilla JavaScript, GSAP can animate any DOM element, canvas object, or generic JavaScript object.

1.2 Why ”GreenSock”?

The name comes from the company’s origins. While users sometimes mistakenly search for ”Green Stock Animation,” the correct term is GreenSock. It has evolved from a Flash-based animation tool to the standard for the modern web, powering over 11 million sites, including award-winning experiences by Apple, Google, and Sony.

1.3 Core Features

- **Speed:** GSAP is highly optimized for performance, often outperforming CSS transitions in complex scenarios.
- **Robustness:** It handles browser inconsistencies automatically (e.g., SVG transform bugs).
- **Plugins:** A rich ecosystem including ScrollTrigger, MorphSVG, and Draggable.
- **Timeline:** The ability to sequence animations is GSAP’s superpower.

Chapter 2

Getting Started

2.1 Installation

There are several ways to include GSAP in your project.

2.1.1 CDN (Content Delivery Network)

The quickest way to start is by adding the script tag to your HTML file.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.12.2/gsap.min.js"></script>
```

2.1.2 NPM / Yarn

For modern build workflows (React, Vue, etc.):

```
npm install gsap
# or
yarn add gsap
```

Then import it in your JavaScript file:

```
import gsap from "gsap";
```

2.2 Your First Tween

A ”Tween” is a single animation. It creates an intermediate state between a start value and an end value.

```
// Moves the element with class "box" to x-position 200
gsap.to(".box", {
    x: 200,
    duration: 2
});
```

In this example:

- **Target:** `".box"` (uses ‘document.querySelectorAll‘ under the hood).
- **Vars Object:** `{ x: 200, duration: 2 }`. This defines *what* to animate and *how*.

Chapter 3

The Core: Tweens

3.1 The Three Main Methods

GSAP provides three primary methods for creating animations.

3.1.1 gsap.to()

This is the most common method. It animates an element *from* its current state *to* the values you define.

```
gsap.to(".circle", {
    opacity: 0.5,
    x: 100
});
```

3.1.2 gsap.from()

This animates an element *from* the values you define *to* its current state. This is incredibly useful for "intro" animations where elements fly onto the screen.

```
// The element starts at opacity 0 and y 100, then moves to its CSS defaults
gsap.from(".hero-text", {
    opacity: 0,
    y: 100,
    duration: 1
});
```

3.1.3 gsap.fromTo()

This allows you to define both the starting and ending values, giving you complete control. This is necessary if you need to reset an animation to a specific state before playing it.

```
gsap.fromTo(".box",
    { x: 0, opacity: 0 }, // From values
    { x: 200, opacity: 1, duration: 2 } // To values
);
```

3.2 Common Properties

GSAP can animate almost any CSS property, but it uses specific shorthand for transforms to ensure performance.

- `x`: translateX (pixels or %)
- `y`: translateY (pixels or %)
- `rotation`: rotate (degrees)
- `scale`: scale (multiplier, 1 is normal)
- `opacity`: opacity (0 to 1)
- `backgroundColor`: Background color (accepts hex, rgb, hsl)

Note: Always prefer animating transforms (`x`, `y`, `scale`, `rotation`) and `opacity` over properties like `top`, `left`, or `margin`, as transforms are hardware-accelerated and do not trigger browser layout reflows.

3.3 Easing

Easing determines the "feel" of the animation. Does it start slow and speed up? Does it bounce?

```
gsap.to(".box", {
    x: 300,
    ease: "power1.in",  // Start slow, speed up
    duration: 2
});

gsap.to(".box", {
    y: 300,
    ease: "bounce.out", // Bounces at the end
    duration: 2
});
```

Common eases: `none`, `power1`, `power2`, `power3`, `power4`, `back`, `elastic`, `bounce`, `circ`, `expo`, `sine`. Each ease has `.in`, `.out`, or `.inOut` variants.

Chapter 4

Sequencing with Timelines

4.1 The Power of Timelines

Without timelines, creating a complex sequence involves calculating delays manually. If you change the duration of the first animation, you have to adjust the delay of every subsequent animation.

A `Timeline` solves this. It acts as a container for tweens.

4.2 Creating a Basic Timeline

```
// Create a timeline instance
let tl = gsap.timeline();

// Chain animations
tl.to(".box1", { x: 100, duration: 1 })
  .to(".box2", { y: 50, duration: 1 })  // Starts after box1 finishes
  .to(".box3", { rotation: 180, duration: 1 }); // Starts after box2 finishes
```

4.3 The Position Parameter

The secret weapon of timelines is the Position Parameter. It allows you to control exactly when an animation starts relative to the timeline.

```
let tl = gsap.timeline();

tl.to(".box1", { x: 100 })
  .to(".box2", { x: 100 }, "-=0.5") // Starts 0.5s BEFORE box1 ends
  .to(".box3", { x: 100 }, "<")     // Starts at the SAME TIME as box2
  .to(".box4", { x: 100 }, "+=1");  // Starts 1s AFTER box3 ends
```

4.4 Timeline Defaults

You can set default properties for all tweens in a timeline to avoid repetition.

```
let tl = gsap.timeline({
    defaults: {
        duration: 1,
        ease: "power2.out",
        opacity: 0
    }
});

tl.from(".header", { y: -50 }) // Inherits defaults
  .from(".sidebar", { x: -50 }) // Inherits defaults
  .from(".content", { y: 50 }); // Inherits defaults
```

Chapter 5

ScrollTrigger

5.1 Introduction to ScrollTrigger

ScrollTrigger is the most popular GSAP plugin. It enables scroll-driven animations with minimal code. It can trigger animations when elements enter the viewport, or link animation progress directly to the scrollbar (scrubbing).

5.2 Basic Setup

First, you must register the plugin (if using a build tool).

```
gsap.registerPlugin(ScrollTrigger);

gsap.to(".box", {
    scrollTrigger: ".box", // Starts when ".box" enters viewport
    x: 500,
    duration: 3
});
```

5.3 Toggle Actions

You can control what happens when the user scrolls in and out of the trigger area using `toggleActions`.

```
// toggleActions: "onEnter onLeave onEnterBack onLeaveBack"
gsap.to(".box", {
    scrollTrigger: {
        trigger: ".box",
        toggleActions: "play pause resume reset"
    },
    x: 500
});
```

Common actions: `play`, `pause`, `resume`, `reverse`, `restart`, `reset`, `complete`, `none`.

5.4 Scrubbing

Scrubbing links the animation playhead directly to the scroll position. The animation doesn't play over time; it plays over pixels scrolled.

```
gsap.to(".box", {
    scrollTrigger: {
        trigger: ".container",
        start: "top center", // When top of container hits center of viewport
        end: "bottom top",   // When bottom of container hits top of viewport
        scrub: true,         // Smooth scrubbing
        // scrub: 1          // Takes 1 second to catch up (smoother)
    },
    rotation: 360
});
```

5.5 Pinning

Pinning fixes an element in place while the scroll continues.

```
ScrollTrigger.create({
    trigger: ".gallery",
    start: "top top",
    end: "+=2000", // Pin for 2000px of scrolling
    pin: true
});
```

Chapter 6

Advanced Techniques

6.1 Staggers

Staggers allow you to animate a group of elements with a delay between each one, creating a wave effect.

```
gsap.from(".menu-item", {
    y: 50,
    opacity: 0,
    duration: 0.5,
    stagger: 0.1 // 0.1s delay between each item
});

// Grid staggering
gsap.to(".grid-box", {
    scale: 0.1,
    y: 60,
    stagger: {
        grid: [5, 10], // rows, columns
        from: "center", // start from the center
        amount: 1.5
    }
});
```

6.2 Callbacks

GSAP provides hooks to run custom logic at specific points in an animation.

```
gsap.to(".box", {
    x: 100,
    onStart: () => console.log("Started"),
    onUpdate: () => console.log("Animating..."),
    onComplete: () => console.log("Finished!"),
    onReverseComplete: () => console.log("Back to start")
});
```

6.3 Controlling Animations

You can assign an animation to a variable and control it later.

```
let tween = gsap.to(".box", { x: 100, paused: true });

document.querySelector("#playBtn").addEventListener("click", () => tween.play());
document.querySelector("#pauseBtn").addEventListener("click", () => tween.pause());
document.querySelector("#reverseBtn").addEventListener("click", () => tween.reverse());
```

Chapter 7

Using GSAP with React

7.1 The useGSAP Hook

React's strict mode and cleanup cycles can make animations tricky. GSAP introduced the `useGSAP` hook to handle cleanup automatically.

```
import { useRef } from 'react';
import gsap from 'gsap';
import { useGSAP } from '@gsap/react';

export default function App() {
  const container = useRef();

  useGSAP(() => {
    // gsap code here...
    gsap.to(".box", { rotation: 360 });
  }, { scope: container }); // Scope selector to this component

  return (
    <div ref={container} className="app">
      <div className="box">Hello</div>
    </div>
  );
}
```

Chapter 8

Common Mistakes & Best Practices

8.1 Mistakes to Avoid

1. **Animating Layout Properties:** Avoid animating `width`, `height`, `top`, or `left`. These trigger layout recalculations. Use `scale`, `x`, and `y` instead.
2. **Forgetting Cleanup in Frameworks:** If using React/Vue without `useGSAP` or `gsap.context()`, your animations might double up or cause memory leaks.
3. **Not Using String Quotes for Selectors:** `gsap.to(box, ...)` will fail if `box` isn't a defined variable. Use `gsap.to(".box", ...)`.

8.2 Performance Tips

- Use `will-change: transform` in CSS for elements that animate frequently.
- Use the `force3D: true` property (GSAP does this automatically mostly) to push elements to the GPU.
- Don't create new timelines on every render loop.