GSAP x Vite React.js - Complete Documentation 2025

Table of Contents

---

1. Introduction & Setup {#introduction}

What is GSAP?

GSAP (GreenSock Animation Platform) is the industry-standard JavaScript animation library. All GSAP animations, ScrollTriggers, Draggables, and SplitText instances created when the useGSAP() hook executes will be reverted automatically when the component unmounts.

Why GSAP with React?

- **Framework-agnostic**: Works seamlessly across any project
- **High performance**: Optimized for 60fps animations
- **Powerful features**: Timeline control, scroll animations, morphing
- **React integration**: Official `@gsap/react` hook for cleanup

2025 Updates

Thanks to Webflow's sponsorship, GSAP is now 100% FREE including ALL bonus plugins like SplitText, MorphSVG, ScrollSmoother, and others that were previously exclusive to Club GreenSock members.

---

2. Installation & Configuration {#installation}

Create Vite React Project

```bash
# Create new Vite project with React
npm create vite@latest my-gsap-app -- --template react

# Navigate to directory
cd my-gsap-app

# Install dependencies
npm install
```

Install GSAP

```bash
# Install GSAP core and React hook
npm install gsap @gsap/react

# All plugins are now FREE (as of 2025)
# No additional packages needed!
```

Project Structure

```
my-gsap-app/
├── src/
│   ├── components/
│   │   └── AnimatedComponent.jsx
│   ├── hooks/
│   │   └── useScrollAnimation.js
│   ├── utils/
│   │   └── animations.js
│   ├── plugins/
│   │   └── index.js
│   ├── App.jsx
│   └── main.jsx
├── package.json
└── vite.config.js
```

Central Plugin Configuration

Create `src/plugins/index.js`:

```javascript
import { gsap } from "gsap";
import { ScrollTrigger } from "gsap/ScrollTrigger";
import { ScrollSmoother } from "gsap/ScrollSmoother";
import { SplitText } from "gsap/SplitText";
import { MorphSVG } from "gsap/MorphSVG";
import { Draggable } from "gsap/Draggable";

// Register all plugins once
gsap.registerPlugin(
  ScrollTrigger,
  ScrollSmoother,
  SplitText,
  MorphSVG,
  Draggable
);

// Export for use throughout app
export {
  gsap,
  ScrollTrigger,
  ScrollSmoother,
  SplitText,
  MorphSVG,
  Draggable
};
```

---

3. Core Concepts {#core-concepts}

GSAP Methods

gsap.to()

Animates FROM current state TO new values

```javascript
gsap.to(".box", { x: 100, duration: 1 });
```

gsap.from()

Animates FROM specified values TO current state

```javascript
gsap.from(".box", { opacity: 0, y: 50, duration: 1 });
```

gsap.fromTo()

Complete control over start and end values

```javascript
gsap.fromTo(".box",
  { x: 0, opacity: 0 },  // from
  { x: 100, opacity: 1, duration: 1 } // to
);
```

gsap.set()

Immediately sets properties without animation

```javascript
gsap.set(".box", { x: 100, opacity: 0 });
```

Animation Properties

Transform Properties (Best Performance)

```javascript
gsap.to(".element", {
  x: 100,          // translateX
  y: 50,           // translateY
  rotation: 45,    // rotate in degrees
  scale: 1.5,      // scale uniformly
  scaleX: 2,       // scale horizontally
  scaleY: 0.5,     // scale vertically
  skewX: 10,       // skew horizontally
  skewY: 5,        // skew vertically
  transformOrigin: "center center"
});
```

Common CSS Properties

javascript

```javascript
gsap.to(".element", {
  opacity: 0.5,
  backgroundColor: "#ff0000",
  color: "#ffffff",
  borderRadius: "50%",
  width: "200px",
  height: "200px"
});
```

Easing Functions

javascript

```javascript
gsap.to(".box", {
  x: 100,
  ease: "power2.out",  // Various easing options:
  // "power1", "power2", "power3", "power4"
  // "back", "elastic", "bounce", "circ", "expo"
  // Add .in, .out, or .inOut
  // Examples: "power2.inOut", "elastic.out", "bounce.in"
});
```

Duration & Delay

javascript

```javascript
gsap.to(".box", {
  x: 100,
  duration: 2,      // Animation length in seconds
  delay: 0.5,       // Wait before starting
  repeat: 3,        // Repeat 3 times (4 total plays)
  repeatDelay: 1,   // Delay between repeats
  yoyo: true        // Reverse on alternate iterations
});
```

---

4. Basic to Advanced Animations {#animations}

Level 1: Basic Animations

Simple Fade In

```javascript
import { useGSAP } from "@gsap/react";
import { gsap } from "gsap";

function FadeIn() {
  const boxRef = useRef();

  useGSAP(() => {
    gsap.from(boxRef.current, {
      opacity: 0,
      duration: 1,
      ease: "power2.out"
    });
  });

  return <div ref={boxRef}>Hello World</div>;
}
```

Slide Animation

```javascript
function SlideIn() {
  const elementRef = useRef();

  useGSAP(() => {
    gsap.from(elementRef.current, {
      x: -100,
      opacity: 0,
      duration: 1,
      ease: "power3.out"
    });
  });

  return <div ref={elementRef}>Sliding Content</div>;
}
```

Level 2: Stagger Animations

List Items Stagger

javascript

```javascript
function StaggerList() {
  const listRef = useRef();

  useGSAP(() => {
    gsap.from(listRef.current.children, {
      y: 50,
      opacity: 0,
      duration: 0.6,
      stagger: 0.1, // 0.1s delay between each item
      ease: "back.out"
    });
  }, { scope: listRef });

  return (
    <ul ref={listRef}>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
      <li>Item 4</li>
    </ul>
  );
}
```

Grid Stagger (Advanced)

javascript

```
function GridAnimation() {
  const gridRef = useRef();

  useGSAP(() => {
    gsap.from(gridRef.current.children, {
      scale: 0,
      opacity: 0,
      duration: 0.5,
      stagger: {
        amount: 1.5,      // Total stagger duration
        from: "center",   // "start", "center", "end", "edges", "random"
        grid: [4, 4],     // Grid layout [rows, columns]
        ease: "power2.inOut"
      }
    });
  }, { scope: gridRef });

  return (
    <div ref={gridRef} className="grid">
      {Array(16).fill(0).map((_, i) => (
        <div key={i} className="grid-item">Item {i + 1}</div>
      ))}
    </div>
  );
}
```

Level 3: Timeline Sequences

Basic Timeline

javascript

```javascript
function TimelineSequence() {
  const containerRef = useRef();

  useGSAP(() => {
    const tl = gsap.timeline();

    tl.from(".title", {
      y: -50,
      opacity: 0,
      duration: 0.6
    })
    .from(".subtitle", {
      y: 20,
      opacity: 0,
      duration: 0.6
    }, "-=0.3") // Overlap by 0.3s
    .from(".button", {
      scale: 0,
      duration: 0.4
    });
  }, { scope: containerRef });

  return (
    <div ref={containerRef}>
      <h1 className="title">Welcome</h1>
      <p className="subtitle">To GSAP Animations</p>
      <button className="button">Get Started</button>
    </div>
  );
}
```

Advanced Timeline with Labels

javascript

```
function AdvancedTimeline() {
  const ref = useRef();

  useGSAP(() => {
    const tl = gsap.timeline({
      repeat: -1,  // Infinite loop
      yoyo: true
    });

    tl.addLabel("start")
      .to(".box", { x: 200, duration: 1 })
      .to(".box", { y: 200, duration: 1 })
      .addLabel("corner")
      .to(".box", { rotation: 360, duration: 1 }, "corner")
      .to(".box", { scale: 2, duration: 0.5 });

    // Control timeline
    // tl.pause();
    // tl.play("corner"); // Jump to label
    // tl.timeScale(2); // 2x speed
  }, { scope: ref });

  return <div ref={ref}><div className="box">Box</div></div>;
}
```

Level 4: Complex Animations

Morphing Path Animation

javascript

```
import { MorphSVG } from "../plugins";

function MorphAnimation() {
  const svgRef = useRef();

  useGSAP(() => {
    gsap.to("#circle", {
      morphSVG: "#star",
      duration: 2,
      ease: "power2.inOut",
      repeat: -1,
      yoyo: true
    });
  }, { scope: svgRef });

  return (
    <svg ref={svgRef} viewBox="0 0 100 100">
      <circle id="circle" cx="50" cy="50" r="30" fill="blue"/>
      <path id="star" d="M50,10 L60,40 L90,45 L65,65 L70,95 L50,80 L30,95 L35,65 L10,45 L40,40 Z" opacity="0"/>
    </svg>
  );
}
```

Text Split Animation

javascript

```
import { SplitText } from "../plugins";

function TextAnimation() {
  const textRef = useRef();

  useGSAP(() => {
    const split = new SplitText(textRef.current, { type: "chars,words" });

    gsap.from(split.chars, {
      opacity: 0,
      y: 50,
      rotateX: -90,
      stagger: 0.02,
      duration: 0.5,
      ease: "back.out"
    });
  });

  return <h1 ref={textRef}>Amazing Text Animation</h1>;
}
```

Parallax Effect

javascript

```
function ParallaxSection() {
  const sectionRef = useRef();

  useGSAP(() => {
    gsap.to(".layer-1", {
      y: -100,
      scrollTrigger: {
        trigger: sectionRef.current,
        start: "top bottom",
        end: "bottom top",
        scrub: 1 // Smooth scrubbing
      }
    });

    gsap.to(".layer-2", {
      y: -200,
      scrollTrigger: {
        trigger: sectionRef.current,
        start: "top bottom",
        end: "bottom top",
        scrub: 1
      }
    });
  }, { scope: sectionRef });

  return (
    <section ref={sectionRef}>
      <div className="layer-1">Slow Layer</div>
      <div className="layer-2">Fast Layer</div>
    </section>
  );
}
```

---

5. React-Specific Patterns {#react-patterns}

useGSAP Hook (Recommended)

useGSAP() is a drop-in replacement for useEffect() or useLayoutEffect() that automatically handles cleanup using gsap.context().

Basic Usage

javascript

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

gsap.registerPlugin(useGSAP);

function Component() {
  const container = useRef();

  useGSAP(() => {
    // All animations here are automatically cleaned up
    gsap.to(".box", { rotation: 360 });
  }, { scope: container });

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

With Dependencies

javascript

```javascript
function DependentAnimation({ endX }) {
  const container = useRef();

  useGSAP(() => {
    gsap.to(".box", { x: endX, duration: 1 });
  }, {
    dependencies: [endX], // Re-run when endX changes
    scope: container
  });

  return <div ref={container}><div className="box">Box</div></div>;
}
```

With Revert on Update

javascript

```
function RevertableAnimation({ color }) {
  const container = useRef();

  useGSAP(() => {
    gsap.to(".box", { backgroundColor: color, duration: 1 });
  }, {
    dependencies: [color],
    revertOnUpdate: true,  // Revert previous animation
    scope: container
  });

  return <div ref={container}><div className="box">Box</div></div>;
}
```

Event Handlers with contextSafe

javascript

```
function InteractiveAnimation() {
  const container = useRef();
  const boxRef = useRef();

  const { contextSafe } = useGSAP({ scope: container });

  // Wrap event handlers with contextSafe
  const handleClick = contextSafe(() => {
    gsap.to(boxRef.current, {
      rotation: "+=360",
      duration: 1
    });
  });

  return (
    <div ref={container}>
      <div ref={boxRef} onClick={handleClick} className="box">
        Click Me
      </div>
    </div>
  );
}
```

State-Driven Animations

javascript

```
function StateAnimation() {
  const [isOpen, setIsOpen] = useState(false);
  const menuRef = useRef();

  useGSAP(() => {
    gsap.to(menuRef.current, {
      x: isOpen ? 0 : -300,
      duration: 0.5,
      ease: "power3.out"
    });
  }, { dependencies: [children] });

  return <div ref={contentRef}>{children}</div>;
}
```

Loading Screen

javascript

```
function LoadingScreen({ onComplete }) {
  const loaderRef = useRef();

  useGSAP(() => {
    const tl = gsap.timeline({
      onComplete: () => onComplete?.()
    });

    tl.to(loaderRef.current, {
      scale: 1.2,
      duration: 0.5,
      ease: "power2.inOut",
      yoyo: true,
      repeat: 1
    })
    .to(loaderRef.current, {
      opacity: 0,
      scale: 0,
      duration: 0.5,
      ease: "back.in"
    });
  });

  return (
    <div className="loader-container">
      <div ref={loaderRef} className="loader">Loading...</div>
    </div>
  );
}
```

Hero Section Animation

javascript

```
function HeroSection() {
  const containerRef = useRef();

  useGSAP(() => {
    const tl = gsap.timeline();

    tl.from(".hero-title", {
      y: 100,
      opacity: 0,
      duration: 1,
      ease: "power3.out"
    })
    .from(".hero-subtitle", {
      y: 50,
      opacity: 0,
      duration: 0.8,
      ease: "power3.out"
    }, "-=0.5")
    .from(".hero-button", {
      scale: 0,
      opacity: 0,
      duration: 0.5,
      ease: "back.out"
    }, "-=0.3")
    .from(".hero-image", {
      scale: 0.8,
      opacity: 0,
      duration: 1,
      ease: "power2.out"
    }, "-=0.8");
  }, { scope: containerRef });

  return (
    <section ref={containerRef} className="hero">
      <h1 className="hero-title">Welcome to Our Site</h1>
      <p className="hero-subtitle">Amazing experiences await</p>
      <button className="hero-button">Get Started</button>
      <img className="hero-image" src="/hero.jpg" alt="Hero" />
    </section>
  );
}
```

Card Hover Animation

javascript

```
function AnimatedCard({ title, description }) {
  const cardRef = useRef();

  const { contextSafe } = useGSAP({ scope: cardRef });

  const handleMouseEnter = contextSafe(() => {
    gsap.to(cardRef.current, {
      scale: 1.05,
      boxShadow: "0 20px 40px rgba(0,0,0,0.2)",
      duration: 0.3,
      ease: "power2.out"
    });

    gsap.to(".card-image", {
      scale: 1.1,
      duration: 0.3,
      ease: "power2.out"
    });
  });

  const handleMouseLeave = contextSafe(() => {
    gsap.to(cardRef.current, {
      scale: 1,
      boxShadow: "0 5px 15px rgba(0,0,0,0.1)",
      duration: 0.3,
      ease: "power2.out"
    });

    gsap.to(".card-image", {
      scale: 1,
      duration: 0.3,
      ease: "power2.out"
    });
  });

  return (
    <div
      ref={cardRef}
      className="card"
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    >
      <img className="card-image" src="/placeholder.jpg" alt={title} />
      <h3>{title}</h3>
      <p>{description}</p>
    </div>
  );
}
```

Reveal on Scroll

javascript

```javascript
function RevealOnScroll() {
  const containerRef = useRef();

  useGSAP(() => {
    const sections = gsap.utils.toArray(".reveal-section");

    sections.forEach((section) => {
      gsap.from(section, {
        y: 100,
        opacity: 0,
        duration: 1,
        scrollTrigger: {
          trigger: section,
          start: "top 80%",
          end: "top 20%",
          toggleActions: "play none none reverse"
        }
      });
    });
  }, { scope: containerRef });

  return (
    <div ref={containerRef}>
      <section className="reveal-section">Section 1</section>
      <section className="reveal-section">Section 2</section>
      <section className="reveal-section">Section 3</section>
    </div>
  );
}
```

Magnetic Button

javascript

```
function MagneticButton({ children }) {
  const buttonRef = useRef();

  const { contextSafe } = useGSAP({ scope: buttonRef });

  const handleMouseMove = contextSafe((e) => {
    const button = buttonRef.current;
    const rect = button.getBoundingClientRect();
    const x = e.clientX - rect.left - rect.width / 2;
    const y = e.clientY - rect.top - rect.height / 2;

    gsap.to(button, {
      x: x * 0.3,
      y: y * 0.3,
      duration: 0.3,
      ease: "power2.out"
    });
  });

  const handleMouseLeave = contextSafe(() => {
    gsap.to(buttonRef.current, {
      x: 0,
      y: 0,
      duration: 0.5,
      ease: "elastic.out(1, 0.5)"
    });
  });

  return (
    <button
      ref={buttonRef}
      onMouseMove={handleMouseMove}
      onMouseLeave={handleMouseLeave}
      className="magnetic-button"
    >
      {children}
    </button>
  );
}
```

Infinite Marquee

javascript

```
function InfiniteMarquee({ items }) {
  const marqueeRef = useRef();

  useGSAP(() => {
    const marquee = marqueeRef.current;
    const marqueeContent = marquee.querySelector(".marquee-content");
    const marqueeWidth = marqueeContent.offsetWidth;

    gsap.to(marqueeContent, {
      x: -marqueeWidth / 2,
      duration: 20,
      ease: "none",
      repeat: -1
    });
  });

  return (
    <div ref={marqueeRef} className="marquee">
      <div className="marquee-content">
        {[...items, ...items].map((item, i) => (
          <span key={i}>{item}</span>
        ))}
      </div>
    </div>
  );
}
```

Image Gallery with Lightbox

```
javascript
```

```
function ImageGallery({ images }) {
  const [selectedImage, setSelectedImage] = useState(null);
  const galleryRef = useRef();
  const lightboxRef = useRef();

  useGSAP(() => {
    gsap.from(".gallery-item", {
      scale: 0,
      opacity: 0,
      duration: 0.5,
      stagger: {
        amount: 1,
        from: "random"
      },
      ease: "back.out"
    });
  }, { scope: galleryRef });

  useGSAP(() => {
    if (selectedImage) {
      gsap.fromTo(lightboxRef.current,
        { opacity: 0, scale: 0.8 },
        { opacity: 1, scale: 1, duration: 0.3 }
      );
    }
  }, { dependencies: [selectedImage] });

  const handleClose = () => {
    gsap.to(lightboxRef.current, {
      opacity: 0,
      scale: 0.8,
      duration: 0.3,
      onComplete: () => setSelectedImage(null)
    });
  };

  return (
    <>
      <div ref={galleryRef} className="gallery">
        {images.map((img, i) => (
          <div
            key={i}
            className="gallery-item"
            onClick={() => setSelectedImage(img)}
          >
            <img src={img} alt={`Gallery ${i}`} />
          </div>
        ))}
      </div>

      {selectedImage && (
        <div className="lightbox" onClick={handleClose}>
          <div ref={lightboxRef}>
            <img src={selectedImage} alt="Selected" />
          </div>
        </div>
      )}
    </>
  );
}
```

Typewriter Effect

javascript

```
function TypewriterText({ text }) {
  const textRef = useRef();

  useGSAP(() => {
    const chars = text.split("");
    textRef.current.innerHTML = "";

    chars.forEach((char, i) => {
      const span = document.createElement("span");
      span.textContent = char;
      span.style.opacity = "0";
      textRef.current.appendChild(span);
    });

    gsap.to(textRef.current.children, {
      opacity: 1,
      duration: 0.05,
      stagger: 0.05,
      ease: "none"
    });
  }, { dependencies: [text] });

  return <div ref={textRef}></div>;
}
```

Drawer/Sidebar Animation

javascript

```
function AnimatedDrawer({ isOpen, onClose, children }) {
  const drawerRef = useRef();
  const overlayRef = useRef();

  useGSAP(() => {
    if (isOpen) {
      gsap.to(overlayRef.current, {
        opacity: 1,
        duration: 0.3,
        pointerEvents: "auto"
      });
      gsap.to(drawerRef.current, {
        x: 0,
        duration: 0.3,
        ease: "power2.out"
      });
    } else {
      gsap.to(overlayRef.current, {
        opacity: 0,
        duration: 0.3,
        pointerEvents: "none"
      });
      gsap.to(drawerRef.current, {
        x: "100%",
        duration: 0.3,
        ease: "power2.in"
      });
    }
  }, { dependencies: [isOpen] });

  return (
    <>
      <div
        ref={overlayRef}
        className="overlay"
        onClick={onClose}
        style={{ opacity: 0, pointerEvents: "none" }}
      />
      <div
        ref={drawerRef}
        className="drawer"
        style={{ transform: "translateX(100%)" }}
      >
        {children}
      </div>
    </>
  );
}
```

Number Counter Animation

javascript

```javascript
function CounterAnimation({ target, duration = 2 }) {
  const counterRef = useRef();
  const [count, setCount] = useState(0);

  useGSAP(() => {
    ScrollTrigger.create({
      trigger: counterRef.current,
      start: "top 80%",
      onEnter: () => {
        gsap.to(counterRef.current, {
          textContent: target,
          duration: duration,
          snap: { textContent: 1 },
          ease: "power1.out",
          onUpdate: function() {
            setCount(Math.floor(this.targets()[0].textContent));
          }
        });
      },
      once: true
    });
  });

  return (
    <div ref={counterRef} className="counter">
      {count}
    </div>
  );
}
```

10. What to Use / Avoid {#dos-donts}

Modern 2025 Recommendations

✅ ALWAYS USE

1. **useGSAP Hook**
   - Automatic cleanup
   - Better React integration
   - Prevents memory leaks
2. **Transform Properties**
   - `x`, `y`, `rotation`, `scale`
   - GPU-accelerated
   - 60fps performance
3. **ScrollTrigger for Scroll Animations**
   - Industry standard
   - Performant
   - Feature-rich
4. **Refs for Element Selection**
   - React-friendly
   - Type-safe
   - Prevents DOM conflicts
5. **contextSafe for Event Handlers**
   - Proper cleanup
   - Scoped animations
   - Memory efficient
6. **Timeline for Complex Sequences**
   - Better control
   - Easier to maintain
   - Reusable
7. **Stagger for Lists**

- Better UX
- Performant
- Professional look

8. **Force3D for Hardware Acceleration**
   - Smoother animations
   - Better performance
   - Mobile optimization

❌ NEVER USE

1. **Direct DOM Manipulation in React**

```javascript
// ❌ Don't do this
document.querySelector(".box").style.left = "100px";

// ✅ Use refs and GSAP
gsap.to(boxRef.current, { x: 100 });
```

2. **Layout-Triggering Properties**
   - Avoid: `width`, `height`, `top`, `left`, `margin`, `padding`
   - Use: `scale`, `x`, `y`, `opacity`

3. **Expensive CSS Properties**

```javascript
// ❌ Slow
gsap.to(".box", {
  filter: "blur(10px)",
  boxShadow: "0 10px 20px rgba(0,0,0,0.5)"
});

// ✅ Fast
gsap.to(".box", {
  opacity: 0.5,
  scale: 1.1
});
```

4. **useEffect for Animations** (use useGSAP instead)

```javascript
// ❌ Manual cleanup required
useEffect(() => {
  const tween = gsap.to(".box", { x: 100 });
  return () => tween.kill();
}, []);

// ✅ Automatic cleanup
useGSAP(() => {
  gsap.to(".box", { x: 100 });
});
```

5. **Multiple IDs for Same Elements**

```javascript
// ❌ Invalid HTML
<div id="item">1</div>
<div id="item">2</div>

// ✅ Use classes
<div className="item">1</div>
<div className="item">2</div>
```

6. **Creating Animations Without Cleanup**
   - Always use useGSAP hook
   - Always scope animations
   - Always clean up event listeners
7. **Animating Too Many Elements**
   - Limit to visible elements
   - Use IntersectionObserver
   - Implement pagination
8. **Blocking the Main Thread**
   - Don't animate during heavy computations
   - Use `requestAnimationFrame`

- Debounce resize handlers

🎯 BEST FOR 2025

1. **Composition Pattern**

javascript

```javascript
// Create reusable animation hooks
function useFadeIn(ref, options = {}) {
  useGSAP(() => {
    gsap.from(ref.current, {
      opacity: 0,
      y: 20,
      duration: 0.6,
      ...options
    });
  });
}

// Use in components
function MyComponent() {
  const ref = useRef();
  useFadeIn(ref);
  return <div ref={ref}>Content</div>;
}
```

2. **Custom Hooks for Complex Animations**

javascript

```
function useScrollAnimation() {
  const ref = useRef();

  useGSAP(() => {
    gsap.from(ref.current, {
      scrollTrigger: {
        trigger: ref.current,
        start: "top 80%",
        toggleActions: "play none none reverse"
      },
      y: 50,
      opacity: 0,
      duration: 1
    });
  }, { scope: ref });

  return ref;
}
```

3. **Animation Library Structure**

javascript

```
// utils/animations.js
export const fadeIn = (element, options = {}) => {
  return gsap.from(element, {
    opacity: 0,
    y: 20,
    duration: 0.6,
    ease: "power2.out",
    ...options
  });
};

export const slideIn = (element, direction = "left") => {
  const xValue = direction === "left" ? -100 : 100;
  return gsap.from(element, {
    x: xValue,
    opacity: 0,
    duration: 0.8,
    ease: "power3.out"
  });
};
```

4. **Accessibility Considerations**

javascript

```javascript
function AccessibleAnimation() {
  const prefersReducedMotion = window.matchMedia(
    "(prefers-reduced-motion: reduce)"
  ).matches;

  useGSAP(() => {
    gsap.to(".box", {
      x: 100,
      duration: prefersReducedMotion ? 0 : 1,
      ease: prefersReducedMotion ? "none" : "power2.out"
    });
  });
}
```

5. **Performance Monitoring**

javascript

```
  useGSAP(() => {
    const tl = gsap.timeline({
      onStart: () => console.log("Animation started"),
      onComplete: () => console.log("Animation completed"),
      onUpdate: function() {
        console.log("Progress:", this.progress());
      }
    });

    tl.to(".box", { x: 100 });
  });
```

Advanced Techniques

Custom Eases

javascript

```
// Register custom ease
import { CustomEase } from "gsap/CustomEase";
gsap.registerPlugin(CustomEase);

CustomEase.create("myEase", "M0,0 C0.5,0 0.5,1 1,1");

// Use custom ease
gsap.to(".box", {
  x: 100,
  ease: "myEase"
});
```

Dynamic Timelines

javascript

```
function DynamicTimeline({ items }) {
  const containerRef = useRef();

  useGSAP(() => {
    const tl = gsap.timeline();

    items.forEach((item, i) => {
      tl.from(`.item-${i}`, {
        x: -100,
        opacity: 0,
        duration: 0.5
      }, i * 0.1);
    });
  }, { dependencies: [items], scope: containerRef });

  return (
    <div ref={containerRef}>
      {items.map((item, i) => (
        <div key={i} className={`item-${i}`}>{item}</div>
      ))}
    </div>
  );
}
```

3D Transforms

javascript

```javascript
function Transform3D() {
  const boxRef = useRef();

  useGSAP(() => {
    gsap.set(boxRef.current, {
      transformStyle: "preserve-3d",
      perspective: 1000
    });

    gsap.to(boxRef.current, {
      rotationY: 360,
      rotationX: 360,
      duration: 3,
      repeat: -1,
      ease: "none"
    });
  });

  return <div ref={boxRef} className="box-3d">3D Box</div>;
}
```

Draggable Elements

javascript

```
import { Draggable } from "../plugins";

function DraggableBox() {
  const boxRef = useRef();

  useGSAP(() => {
    Draggable.create(boxRef.current, {
      type: "x,y",
      bounds: ".container",
      inertia: true,
      onDrag: function() {
        console.log("Dragging:", this.x, this.y);
      },
      onDragEnd: function() {
        console.log("Drag ended");
      }
    });
  });

  return (
    <div className="container">
      <div ref={boxRef} className="draggable-box">
        Drag Me
      </div>
    </div>
  );
}
```

Path Animation

javascript

```
import { MotionPathPlugin } from "gsap/MotionPathPlugin";
gsap.registerPlugin(MotionPathPlugin);

function PathAnimation() {
  const ballRef = useRef();

  useGSAP(() => {
    gsap.to(ballRef.current, {
      duration: 5,
      repeat: -1,
      ease: "none",
      motionPath: {
        path: "#path",
        align: "#path",
        autoRotate: true,
        alignOrigin: [0.5, 0.5]
      }
    });
  });

  return (
    <svg viewBox="0 0 500 500">
      <path
        id="path"
        d="M 50,250 Q 250,50 450,250 Q 250,450 50,250"
        fill="none"
        stroke="#ccc"
      />
      <circle ref={ballRef} r="10" fill="blue" />
    </svg>
  );
}
```

Production Checklist

Before Deployment

- [ ] Remove all `markers: true` from ScrollTriggers
- [ ] Set `force3D: true` for animated elements
- [ ] Add `will-change` to frequently animated elements
- [ ] Remove console.logs from animations

- [ ] Test on mobile devices
- [ ] Check for memory leaks (cleanup in useGSAP)
- [ ] Verify accessibility (reduced motion)
- [ ] Optimize images and assets
- [ ] Test ScrollTrigger.refresh() on resize
- [ ] Validate performance with Chrome DevTools

Performance Budget

- Keep animation duration < 1s for micro-interactions
- Limit concurrent animations to 5-10 elements
- Use `stagger` instead of individual animations
- Debounce scroll/resize handlers (250ms)
- Lazy load off-screen animations

Resources & Links

Official Documentation

- GSAP Docs: https://gsap.com/docs/v3/
- React Integration: https://gsap.com/resources/React/
- ScrollTrigger: https://gsap.com/docs/v3/Plugins/ScrollTrigger/
- Timeline: https://gsap.com/docs/v3/GSAP/Timeline/

Community

- GSAP Forums: https://gsap.com/community/
- CodePen Examples: https://codepen.io/GreenSock/
- GitHub: https://github.com/greensock/GSAP

Learning

- Easing Visualizer: https://gsap.com/docs/v3/Eases/
- Cheat Sheet: https://gsap.com/cheatsheet/

Final Tips

1. **Start Simple**: Master basic animations before complex timelines
2. **Use DevTools**: Chrome Performance tab is your friend
3. **Think in Sequences**: Break complex animations into timelines
4. **Scope Everything**: Always use `scope` in useGSAP
5. **Mobile First**: Test on actual devices, not just desktop
6. **Accessibility**: Always respect `prefers-reduced-motion`
7. **Document**: Comment complex animations for future you
8. **Iterate**: Animation is an art - refine until it feels right

Remember: The best animation is one that serves the user experience, not just looks cool. Less is often more!

Last Updated: November 2025 GSAP Version: 3.12+ (All plugins now FREE): [isOpen] });

return (<> <button onClick={() => setIsOpen(!isOpen)}>Toggle</button> <div ref={menuRef} className="menu">Menu Content</div> </>); }

## 6. ScrollTrigger Mastery {#scrolltrigger}

### Basic ScrollTrigger
```javascript
import { ScrollTrigger } from "../plugins";

function ScrollAnimation() {
  const boxRef = useRef();

  useGSAP(() => {
    gsap.to(boxRef.current, {
      x: 400,
      scrollTrigger: {
        trigger: boxRef.current,
        start: "top center",    // When top of element hits center of viewport
        end: "bottom center",   // When bottom of element hits center of viewport
        scrub: true,            // Smooth scrubbing
        markers: true           // Debug markers (remove in production)
      }
    });
  });

  return <div ref={boxRef} className="box">Scroll Me</div>;
}
```

### ScrollTrigger Start/End Values
```javascript
// Format: "triggerPosition viewportPosition"

start: "top top"       // Element top hits viewport top
start: "top center"    // Element top hits viewport center
start: "top bottom"    // Element top hits viewport bottom
start: "center center" // Element center hits viewport center
start: "bottom top"    // Element bottom hits viewport top

// With offsets
start: "top top+=100"  // 100px below top
start: "top top-=100"  // 100px above top
```

// Absolute positions
end: "+=500"   // 500px of scrolling

### Pin Elements
```javascript
function PinAnimation() {
  const sectionRef = useRef();

  useGSAP(() => {
    ScrollTrigger.create({
      trigger: sectionRef.current,
      start: "top top",
      end: "+=2000",        // Pin for 2000px of scrolling
      pin: true,            // Pin the element
      pinSpacing: true,     // Add spacing below
      anticipatePin: 1      // Smooth pinning
    });
  });

  return <section ref={sectionRef}>Pinned Content</section>;
}
```

### Timeline with ScrollTrigger
```javascript
function ScrollTimeline() {
  const containerRef = useRef();

  useGSAP(() => {
    const tl = gsap.timeline({
      scrollTrigger: {
        trigger: containerRef.current,
        start: "top top",
        end: "+=3000",
        pin: true,
        scrub: 1,
        anticipatePin: 1
      }
    });

    tl.from(".slide-1", { xPercent: 100 })
      .from(".slide-2", { xPercent: 100 })
      .from(".slide-3", { xPercent: 100 });
  }, { scope: containerRef });

  return (
    <div ref={containerRef} className="container">
      <div className="slide-1">Slide 1</div>
      <div className="slide-2">Slide 2</div>
      <div className="slide-3">Slide 3</div>
    </div>
  );
}
```

### Horizontal Scroll
```javascript
function HorizontalScroll() {
  const containerRef = useRef();
  const scrollRef = useRef();

  useGSAP(() => {
    const sections = gsap.utils.toArray(".panel");

    gsap.to(sections, {
      xPercent: -100 * (sections.length - 1),
      ease: "none",
      scrollTrigger: {
        trigger: scrollRef.current,
        pin: true,
        scrub: 1,
        snap: 1 / (sections.length - 1),
        end: () => "+=" + scrollRef.current.offsetWidth
      }
    });
  }, { scope: containerRef });

  return (
    <div ref={containerRef}>
      <div ref={scrollRef} className="scroll-container">
        <div className="panel">Panel 1</div>
        <div className="panel">Panel 2</div>
        <div className="panel">Panel 3</div>
        <div className="panel">Panel 4</div>
      </div>
    </div>
  );
}
```

### Scroll-Triggered Actions
```javascript
function ScrollActions() {
  const [count, setCount] = useState(0);
  const counterRef = useRef();

  useGSAP(() => {
    ScrollTrigger.create({
      trigger: counterRef.current,
      start: "top center",
      onEnter: () => {
        // Animate counter
        gsap.to(counterRef.current, {
          textContent: 1000,
          duration: 2,
          snap: { textContent: 1 },
          onUpdate: function() {
            setCount(Math.floor(this.targets()[0].textContent));
          }
        });
      },
      once: true // Only trigger once
    });
  });

  return <div ref={counterRef}>{count}</div>;
}
```

### Batch Animations
```javascript
function BatchScrollAnimations() {
  const containerRef = useRef();

  useGSAP(() => {
    ScrollTrigger.batch(".fade-in", {
      onEnter: (elements) => {
        gsap.from(elements, {
          opacity: 0,
          y: 50,
          stagger: 0.1,
```

          duration: 1
        });
      },
      start: "top 80%",
      once: true
    });
  }, { scope: containerRef });

  return (
    <div ref={containerRef}>
      <div className="fade-in">Item 1</div>
      <div className="fade-in">Item 2</div>
      <div className="fade-in">Item 3</div>
    </div>
  );
}

## 7. Performance Optimization {#performance}

### Best Performing Properties

By animating the CSS property, you resize the element, which entails recalculating the Layout, redrawing the Paint, and composing the Composite.

#### ✅ USE (GPU-Accelerated)
```javascript
// These are FAST - use transform properties
gsap.to(".element", {
  x: 100,          // translateX
  y: 100,          // translateY
  rotation: 45,    // rotate
  scale: 1.5,      // scale
  opacity: 0.5     // opacity
});
```

#### ❌ AVOID (CPU-Heavy)
```javascript
// These are SLOW - trigger layout recalculation
gsap.to(".element", {
  top: "100px",    // Use y instead
```

  left: "100px",   // Use x instead
  width: "200px",   // Triggers reflow
  height: "200px", // Triggers reflow
  margin: "20px",   // Triggers reflow
  padding: "10px"   // Triggers reflow
});

### will-change Property

The will-change CSS property lets browsers know in advance which element is going to be changed and in what way so that they can set up their optimizations before the element actually gets changed.
```css
/* Add to elements you'll animate */
.animated-element {
  will-change: transform, opacity;
}

/* Remove after animation */
.animated-element.done {
  will-change: auto;
}
```
```javascript
// Or set programmatically
useGSAP(() => {
  const element = elementRef.current;

  // Before animation
  gsap.set(element, { willChange: "transform, opacity" });

  gsap.to(element, {
    x: 100,
    opacity: 0,
    onComplete: () => {
      // After animation
      gsap.set(element, { willChange: "auto" });
    }
  });
});
```

### Force3D
```javascript
```

```
// Enable hardware acceleration
gsap.config({ force3D: true });

// Or per animation
gsap.to(".box", {
  x: 100,
  force3D: true
});
```

### Lag Smoothing

gsap.ticker.lagSmoothing(1000, 16); lets you avoid animation lags when the CPU starts to freeze.
```javascript
// Adjust lag smoothing threshold
gsap.ticker.lagSmoothing(1000, 16);
```

### Reduce Redraws
```javascript
// Bad: Multiple redraws
gsap.to(".box", { x: 100 });
gsap.to(".box", { y: 100 });
gsap.to(".box", { opacity: 0.5 });

// Good: Single redraw
gsap.to(".box", {
  x: 100,
  y: 100,
  opacity: 0.5
});
```

### Stagger for Better Performance

The stagger parameter starts your animations at the specified time interval. Even the smallest playback delay greatly affects the animation, making it slow and twitchy.
```javascript
// Instead of animating all at once
gsap.to(".items", { opacity: 0 });

// Use stagger for better performance
gsap.to(".items", {
  opacity: 0,
```

  stagger: 0.01 // Small delay between each
});

### Optimize ScrollTrigger
```javascript
// Invalidate on resize for better performance
useGSAP(() => {
  ScrollTrigger.create({
    trigger: ".element",
    onRefresh: () => {
      // Recalculate positions
    }
  });

  // Refresh on window resize (debounced)
  let resizeTimer;
  window.addEventListener("resize", () => {
    clearTimeout(resizeTimer);
    resizeTimer = setTimeout(() => {
      ScrollTrigger.refresh();
    }, 250);
  });
});
```

## 8. Best Practices 2025 {#best-practices}

### ✅ DO's

#### 1. Use useGSAP Hook
```javascript
// ✅ Automatic cleanup
useGSAP(() => {
  gsap.to(".box", { x: 100 });
}, { scope: container });
```

#### 2. Centralize Plugin Registration
```javascript
// plugins/index.js
import { gsap } from "gsap";
```

~~~~
import { ScrollTrigger } from "gsap/ScrollTrigger";

gsap.registerPlugin(ScrollTrigger);
export { gsap, ScrollTrigger };
```

#### 3. Use Refs for Scoping
```javascript
const containerRef = useRef();

useGSAP(() => {
  // Animations scoped to container
  gsap.to(".box", { x: 100 });
}, { scope: containerRef });
```

#### 4. Use Transform Properties
```javascript
// ✅ GPU-accelerated
gsap.to(".box", { x: 100, y: 50, rotation: 45 });
```

#### 5. Clean Up Properly
```javascript
useGSAP(() => {
  const tl = gsap.timeline();
  // animations...

  return () => {
    // useGSAP handles this automatically
    // but you can add custom cleanup
    tl.kill();
  };
});
```

#### 6. Use contextSafe for Events
```javascript
const { contextSafe } = useGSAP({ scope: container });

const handleClick = contextSafe(() => {
  gsap.to(".box", { rotation: 360 });
});
```
~~~~

#### 7. Batch Similar Animations
```javascript
ScrollTrigger.batch(".item", {
  onEnter: (elements) => {
    gsap.from(elements, {
      y: 50,
      opacity: 0,
      stagger: 0.1
    });
  }
});
```

#### 8. Use Labels in Timelines
```javascript
const tl = gsap.timeline();
tl.addLabel("start")
  .to(".box", { x: 100 })
  .addLabel("middle")
  .to(".box", { y: 100 });
```

### ❌ DON'Ts

#### 1. Don't Animate Layout Properties
```javascript
// ❌ Slow - triggers reflow
gsap.to(".box", {
  width: "200px",
  height: "200px",
  top: "100px",
  left: "100px"
});

// ✅ Fast - uses transforms
gsap.to(".box", {
  scale: 2,
  x: 100,
  y: 100
});
```

#### 2. Don't Forget Dependencies

```javascript
// ❌ Won't update when props change
useGSAP(() => {
  gsap.to(".box", { x: endX });
});

// ✅ Updates when endX changes
useGSAP(() => {
  gsap.to(".box", { x: endX });
}, { dependencies: [endX] });
```

#### 3. Don't Use Multiple IDs
```javascript
// ❌ Invalid HTML
<div id="box">1</div>
<div id="box">2</div>

// ✅ Use classes
<div className="box">1</div>
<div className="box">2</div>
```

#### 4. Don't Animate Everything
You should not animate everything. Also you should only animate properties like transformations or opacity and avoid animating properties like filter or boxShadow as they consume a lot of CPU in the browsers.
```javascript
// ❌ Too much animation
gsap.to(".box", {
  filter: "blur(10px)",  // Expensive
  boxShadow: "0 10px 20px rgba(0,0,0,0.5)" // Expensive
});

// ✅ Performant alternatives
gsap.to(".box", {
  opacity: 0.5,
  scale: 1.1
});
```

#### 5. Don't Create Animations in Loops Without Scoping
```javascript
// ❌ Memory leak potential
items.forEach(item => {
```

~~~
  gsap.to(item, { x: 100 });
});

// ✅ Use context or batch
useGSAP(() => {
  gsap.to(".item", { x: 100 });
}, { scope: container });
```

#### 6. Don't Ignore Mobile Performance
```javascript
// ✅ Reduce animations on mobile
const isMobile = window.innerWidth < 768;

useGSAP(() => {
  gsap.to(".box", {
    x: 100,
    duration: isMobile ? 0.5 : 1,
    ease: isMobile ? "power1.out" : "power3.out"
  });
});
```

---

## 9. Common Patterns & Examples {#examples}

### Page Transitions
```javascript
function PageTransition({ children }) {
  const contentRef = useRef();

  useGSAP(() => {
    gsap.from(contentRef.current, {
      opacity: 0,
      y: 20,
      duration: 0.6,
      ease: "power2.out"
    });
  }, { dependencies
~~~