GSAP Cinematic Animation Masterclass for Svelte

> *Transform static Svelte interfaces into cinematic experiences with GSAP (GreenSock Animation Platform).*

Table of Contents

Chapter 1: Introduction to GSAP

What is GSAP?

GSAP (GreenSock Animation Platform) is the industry-standard JavaScript animation library. It animates anything JavaScript can touch — CSS properties, SVG attributes, canvas, WebGL, even custom object properties — with unmatched performance and cross-browser consistency.

Why GSAP Over CSS Animations?

Feature	CSS Animations	GSAP
Sequencing	Manual delays	Timeline orchestration
Scroll-driven	Limited	ScrollTrigger plugin
Easing	cubic-bezier only	30+ built-in eases
Runtime control	None	Play, pause, reverse, seek
Performance	Good	Excellent (requestAnimationFrame)
Complex choreography	Painful	Natural

Installation

```
pnpm add gsap
```

Your First GSAP Animation

```
import gsap from 'gsap';

// Animate an element with class "box"
gsap.to('.box', {
  x: 200,
  rotation: 360,
  duration: 1,
  ease: 'power2.out'
});
```

GSAP Core Concepts

GSAP operates on three fundamental primitives:

1. **Tweens** — Single animation operations (to, from, fromTo)
2. **Timelines** — Containers that sequence tweens
3. **Plugins** — Extensions like ScrollTrigger, Draggable, MorphSVG

The GSAP Object Model

```
// gsap is the global entry point
gsap.to(target, vars);       // Animate FROM current state TO new state
gsap.from(target, vars);     // Animate FROM specified state TO current
gsap.fromTo(target, fromVars, toVars); // Full control of start and end
gsap.set(target, vars);      // Instantly set properties (0-duration tween)
```

Understanding Targets

```
// CSS selector string
gsap.to('.my-element', { x: 100 });

// DOM element reference
const el = document.querySelector('.box');
gsap.to(el, { x: 100 });

// Array of elements
gsap.to(['.box1', '.box2', '.box3'], { x: 100 });

// Svelte bound element
let myDiv;
// <div bind:this={myDiv}>
gsap.to(myDiv, { x: 100 });
```

Chapter 2: Tweens — The Building Blocks

gsap.to() — The Workhorse

`gsap.to()` animates elements from their current state to new values.

```
gsap.to('.card', {
  x: 300,           // translateX (pixels)
  y: -50,           // translateY (pixels)
  rotation: 15,     // degrees
  scale: 1.2,       // uniform scale
  opacity: 0.8,
  duration: 0.8,
  ease: 'power3.out'
});
```

gsap.from() — Entrance Animations

`gsap.from()` animates from specified values back to the element's natural state.

```
// Element fades in and slides up from 60px below
gsap.from('.hero-title', {
  y: 60,
  opacity: 0,
  duration: 1,
  ease: 'power2.out'
});
```

gsap.fromTo() — Full Control

When you need explicit start and end values:

```
gsap.fromTo('.progress-bar',
  { width: '0%' },
  { width: '100%', duration: 2, ease: 'none' }
);
```

gsap.set() — Instant Property Setting

Sets properties immediately with no animation (equivalent to a zero-duration tween):

```
// Prepare elements before animating
gsap.set('.card', { opacity: 0, y: 30 });

// Later, animate them in
gsap.to('.card', { opacity: 1, y: 0, duration: 0.6 });
```

Transform Shorthand Properties

GSAP provides convenient shorthand for CSS transforms:

```
gsap.to('.element', {
  x: 100,          // transform: translateX(100px)
  y: 50,           // transform: translateY(50px)
  xPercent: -50,   // transform: translateX(-50%)
  yPercent: -50,   // transform: translateY(-50%)
  rotation: 45,    // transform: rotate(45deg)
  rotationX: 30,   // transform: rotateX(30deg)
  rotationY: 60,   // transform: rotateY(60deg)
  scale: 1.5,      // transform: scale(1.5)
  scaleX: 2,       // transform: scaleX(2)
  scaleY: 0.5,     // transform: scaleY(0.5)
  skewX: 15,       // transform: skewX(15deg)
  skewY: 10,       // transform: skewY(10deg)
  transformOrigin: '50% 50%'
});
```

Tween Special Properties

```
gsap.to('.element', {
  x: 200,
  duration: 1,           // Animation length in seconds
  delay: 0.5,            // Wait before starting
  repeat: 3,             // Repeat 3 additional times (4 total)
  repeatDelay: 0.2,      // Pause between repeats
  yoyo: true,            // Reverse on alternate repeats
  ease: 'elastic.out',   // Easing function
  overwrite: 'auto',     // How to handle conflicting tweens
  onStart: () => console.log('Started'),
  onUpdate: () => console.log('Updating'),
  onComplete: () => console.log('Done'),
  onRepeat: () => console.log('Repeating')
});
```

Controlling Tweens

```
const tween = gsap.to('.box', { x: 500, duration: 2 });

tween.play();          // Play forward
tween.pause();         // Pause at current position
tween.resume();        // Resume from paused position
tween.reverse();       // Play in reverse
tween.restart();       // Jump to start and play
tween.seek(0.5);       // Jump to 0.5 seconds
tween.progress(0.75);  // Jump to 75% progress
tween.timeScale(2);    // Double speed
tween.kill();          // Destroy the tween
```

Animating CSS Properties

```
gsap.to('.element', {
  // Layout
  width: 300,
  height: 200,
  padding: 20,
  borderRadius: '50%',

  // Colors
  color: '#ff6600',
  backgroundColor: 'rgba(0, 0, 0, 0.8)',
  borderColor: '#333',

  // Typography
  fontSize: 24,
  letterSpacing: 2,
  lineHeight: 1.5,

  // Misc
  boxShadow: '0 10px 40px rgba(0,0,0,0.3)',
  clipPath: 'circle(50%)',
  duration: 1
});
```

Chapter 3: Timelines — Orchestrating Sequences

Why Timelines?

Individual tweens are useful, but real animation requires choreography. GSAP timelines let you sequence, overlap, and orchestrate multiple animations with precision.

Creating a Timeline

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

tl.to('.title', { opacity: 1, y: 0, duration: 0.6 })
  .to('.subtitle', { opacity: 1, y: 0, duration: 0.4 })
  .to('.cta-button', { opacity: 1, scale: 1, duration: 0.3 });
```

Each tween starts after the previous one completes — no manual delay calculations.

Position Parameter — Precision Placement

The position parameter is the secret weapon of GSAP timelines:

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

// Absolute time (seconds)
tl.to('.a', { x: 100, duration: 1 }, 0);       // Starts at 0s
tl.to('.b', { x: 100, duration: 1 }, 0.5);     // Starts at 0.5s
tl.to('.c', { x: 100, duration: 1 }, 1);       // Starts at 1s

// Relative to previous tween's end
tl.to('.d', { x: 100, duration: 1 }, '+=0.2'); // 0.2s after previous ends
tl.to('.e', { x: 100, duration: 1 }, '-=0.3'); // 0.3s before previous ends

// Relative to previous tween's start
tl.to('.f', { x: 100, duration: 1 }, '<');      // Same start as previous
tl.to('.g', { x: 100, duration: 1 }, '<0.1');   // 0.1s after previous starts
tl.to('.h', { x: 100, duration: 1 }, '<+=0.5'); // 0.5s after previous starts

// Labels
tl.add('midpoint');
tl.to('.i', { x: 100, duration: 1 }, 'midpoint');
tl.to('.j', { x: 100, duration: 1 }, 'midpoint+=0.3');
```

Timeline Defaults

Apply shared properties to all child tweens:

```
const tl = gsap.timeline({
  defaults: {
    duration: 0.6,
    ease: 'power2.out',
    opacity: 0,
    y: 30
  }
});

// Each tween inherits defaults — override only what differs
tl.from('.title', { y: 60 })       // y overridden to 60
  .from('.subtitle', {})            // uses all defaults
  .from('.cta', { scale: 0.8 });   // adds scale, keeps defaults
```

Nested Timelines

Break complex sequences into reusable pieces:

```
function createHeaderAnimation() {
  const tl = gsap.timeline();
  tl.from('.logo', { opacity: 0, x: -30, duration: 0.5 })
    .from('.nav-links li', { opacity: 0, y: -20, stagger: 0.1 });
  return tl;
}

function createHeroAnimation() {
  const tl = gsap.timeline();
  tl.from('.hero-title', { opacity: 0, y: 60, duration: 0.8 })
    .from('.hero-description', { opacity: 0, y: 40, duration: 0.6 }, '-=0.3')
    .from('.hero-cta', { opacity: 0, scale: 0.8, duration: 0.4 }, '-=0.2');
  return tl;
}

// Master timeline
const master = gsap.timeline();
master
  .add(createHeaderAnimation())
  .add(createHeroAnimation(), '-=0.3');
```

Timeline Control

```
const tl = gsap.timeline({ paused: true });

tl.to('.box', { x: 300, duration: 1 })
  .to('.box', { rotation: 360, duration: 0.5 });

// Control programmatically
tl.play();
tl.pause();
tl.reverse();
tl.restart();
tl.progress(0.5);  // Jump to 50%
tl.timeScale(0.5); // Half speed

// Control from events
button.addEventListener('click', () => tl.reversed(!tl.reversed()));
```

Timeline Callbacks

```
const tl = gsap.timeline({
  onStart: () => console.log('Timeline started'),
  onUpdate: () => console.log(`Progress: ${tl.progress()}`),
  onComplete: () => console.log('Timeline complete'),
  onReverseComplete: () => console.log('Reverse complete'),
  repeat: -1,        // Infinite repeat
  repeatDelay: 1,    // 1s gap between repeats
  yoyo: true         // Alternate direction each repeat
});
```

Chapter 4: Easing — The Soul of Motion

Why Easing Matters

Easing defines the rate of change over time. Linear animation feels mechanical and lifeless. Proper easing creates natural, organic motion that follows real-world physics.

Ease Types

Every ease type comes in three flavors:

- `.in` — Starts slow, accelerates (use for exits)
- `.out` — Starts fast, decelerates (use for entrances)
- `.inOut` — Slow start and end (use for continuous motion)

Built-in Eases

```
// Power eases (most common)
gsap.to('.a', { x: 300, ease: 'none' });         // Linear
gsap.to('.b', { x: 300, ease: 'power1.out' });   // Subtle deceleration
gsap.to('.c', { x: 300, ease: 'power2.out' });   // Moderate deceleration
gsap.to('.d', { x: 300, ease: 'power3.out' });   // Strong deceleration
gsap.to('.e', { x: 300, ease: 'power4.out' });   // Extreme deceleration

// Character eases
gsap.to('.f', { x: 300, ease: 'back.out(1.7)' });    // Overshoots then settles
gsap.to('.g', { x: 300, ease: 'elastic.out(1, 0.3)' }); // Springy bounce
gsap.to('.h', { x: 300, ease: 'bounce.out' });        // Bouncing ball
gsap.to('.i', { x: 300, ease: 'circ.out' });          // Circular motion
gsap.to('.j', { x: 300, ease: 'expo.out' });          // Exponential
gsap.to('.k', { x: 300, ease: 'sine.out' });          // Gentle sine wave

// Steps
gsap.to('.l', { x: 300, ease: 'steps(5)' });    // 5 discrete steps
```

Easing Recommendations by Context

```
// UI Entrances — fast start, gentle landing
const entrance = 'power2.out';    // or power3.out

// UI Exits — gentle start, fast finish
const exit = 'power2.in';         // or power3.in

// Hover effects — snappy and responsive
```

```
const hover = 'power1.out';

// Modal/overlay transitions
const modalIn = 'power3.out';
const modalOut = 'power2.in';

// Scroll-driven reveals
const scrollReveal = 'power2.out';

// Playful/fun interactions
const playful = 'back.out(1.7)';
const bouncy = 'elastic.out(1, 0.5)';

// Loading spinners — constant speed
const spinner = 'none';

// Page transitions
const pageEnter = 'power4.out';
const pageExit = 'power2.in';
```

Custom Eases with gsap.registerEase()

```
// Register a custom ease
gsap.registerEase('myCustom', (progress) => {
  return progress * progress * (3 - 2 * progress); // Smooth step
});

gsap.to('.box', { x: 300, ease: 'myCustom' });
```

Ease Visualization Tip

```
// Quick visual test for easing
function testEase(easeName) {
  gsap.fromTo('.test-box',
    { x: 0 },
    { x: 400, duration: 2, ease: easeName, repeat: -1, repeatDelay: 0.5 }
  );
}

testEase('power3.out');
testEase('elastic.out(1, 0.3)');
testEase('back.out(2)');
```

Chapter 5: ScrollTrigger — Scroll-Driven Animation

Setup

```
import gsap from 'gsap';
import { ScrollTrigger } from 'gsap/ScrollTrigger';

gsap.registerPlugin(ScrollTrigger);
```

Basic ScrollTrigger

```
gsap.to('.section-title', {
  scrollTrigger: '.section-title', // Trigger element
  x: 300,
  duration: 1
});
```

Full ScrollTrigger Configuration

```
gsap.to('.animated-element', {
  scrollTrigger: {
    trigger: '.animated-element',   // Element that triggers
    scroller: window,               // Scrollable ancestor (default: window)
    start: 'top 80%',              // trigger-start scroller-start
    end: 'bottom 20%',             // trigger-end scroller-end
    toggleActions: 'play none none reverse',
    // onEnter onLeave onEnterBack onLeaveBack
    // Options: play, pause, resume, reset, restart, complete, reverse, none

    markers: true,                  // Debug markers (remove in production)
    scrub: true,                   // Link animation to scroll position
    pin: true,                     // Pin element during animation
    pinSpacing: true,              // Add spacing for pinned element
    anticipatePin: 1,              // Reduce pin jitter
    once: false,                   // Only trigger once
    id: 'myTrigger',              // For debugging

    onEnter: () => console.log('Entered'),
    onLeave: () => console.log('Left'),
    onEnterBack: () => console.log('Entered back'),
    onLeaveBack: () => console.log('Left back'),
    onUpdate: (self) => console.log('Progress:', self.progress),
    onToggle: (self) => console.log('Active:', self.isActive),
    onRefresh: (self) => console.log('Refreshed')
  },
  x: 300,
  opacity: 1,
  duration: 1
});
```

Scrub — Scroll-Linked Animation

```
// Boolean scrub: directly tied to scroll
gsap.to('.parallax-bg', {
  scrollTrigger: {
    trigger: '.parallax-section',
    start: 'top bottom',
    end: 'bottom top',
    scrub: true
  },
  y: -200
});
```

```
// Numeric scrub: smoothed with catch-up time (seconds)
gsap.to('.smooth-element', {
  scrollTrigger: {
    trigger: '.section',
    start: 'top center',
    end: 'bottom center',
    scrub: 0.5  // 0.5 second catch-up
  },
  x: 500,
  rotation: 360
});
```

Pinning Sections

```
// Pin an element while scrolling through content
ScrollTrigger.create({
  trigger: '.pinned-section',
  start: 'top top',
  end: '+=1000',          // Pin for 1000px of scrolling
  pin: true,
  pinSpacing: true
});
```

Timeline + ScrollTrigger

```
const tl = gsap.timeline({
  scrollTrigger: {
    trigger: '.feature-section',
    start: 'top center',
    end: 'bottom center',
    scrub: 1
  }
});

tl.from('.feature-icon', { scale: 0, rotation: -180, duration: 0.5 })
  .from('.feature-title', { opacity: 0, x: -50, duration: 0.3 }, '-=0.2')
  .from('.feature-text', { opacity: 0, y: 30, duration: 0.3 }, '-=0.1')
  .from('.feature-image', { opacity: 0, scale: 0.8, duration: 0.5 }, '-=0.2');
```

Batch Animations

```
// Animate elements as they scroll into view, in batches
ScrollTrigger.batch('.card', {
  onEnter: (elements) => {
    gsap.from(elements, {
      opacity: 0,
      y: 60,
      stagger: 0.15,
      duration: 0.6,
      ease: 'power2.out'
    });
  },
  start: 'top 85%'
});
```

ScrollTrigger.matchMedia() — Responsive

```
ScrollTrigger.matchMedia({
  // Desktop
  '(min-width: 1024px)': function () {
    gsap.to('.hero-image', {
      scrollTrigger: {
        trigger: '.hero',
        start: 'top top',
        end: 'bottom top',
        scrub: true
      },
      y: -100,
      scale: 1.1
    });
  },

  // Mobile
  '(max-width: 1023px)': function () {
    gsap.from('.hero-image', {
      scrollTrigger: '.hero-image',
      opacity: 0,
      y: 30,
      duration: 0.6
    });
  }
});
```

Chapter 6: Stagger Animations — Coordinated Motion

Basic Stagger

```
gsap.from('.card', {
  opacity: 0,
  y: 50,
  duration: 0.6,
  stagger: 0.15,       // 0.15s delay between each element
  ease: 'power2.out'
});
```

Advanced Stagger Object

```
gsap.from('.grid-item', {
  opacity: 0,
  scale: 0,
  rotation: -15,
  duration: 0.5,
  ease: 'back.out(1.7)',
  stagger: {
    amount: 1.2,        // Total time distributed across all elements
    from: 'center',     // Start from center and radiate outward
    // from options: 'start', 'end', 'center', 'edges', 'random', or index number
    grid: [4, 6],       // Treat elements as a 4x6 grid
    axis: null,         // null (both), 'x', or 'y'
    ease: 'power2.in'   // Ease the stagger distribution itself
  }
});
```

Stagger from Patterns

```
// From start (default) — left to right / top to bottom
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 'start', each: 0.1 } });

// From end — right to left / bottom to top
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 'end', each: 0.1 } });

// From center — radiate outward
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 'center', each: 0.1 } });

// From edges — collapse inward
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 'edges', each: 0.1 } });

// Random order
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 'random', each: 0.1 } });

// From specific index
gsap.from('.item', { y: 30, opacity: 0, stagger: { from: 3, each: 0.1 } });
```

Grid Stagger — 2D Ripple Effects

```
// Grid elements ripple from top-left corner
gsap.from('.grid-cell', {
  opacity: 0,
  scale: 0,
  duration: 0.4,
  ease: 'back.out(1.7)',
  stagger: {
    grid: 'auto',       // GSAP auto-detects grid dimensions
    from: 0,            // Top-left corner
    amount: 0.8,
    axis: null           // Radiate in both X and Y
  }
});

// Grid elements ripple from center
gsap.from('.grid-cell', {
  opacity: 0,
  y: 20,
  duration: 0.3,
  stagger: {
    grid: [5, 8],
    from: 'center',
    amount: 1
  }
});
```

Stagger in Timelines

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

tl.from('.section-title', { opacity: 0, y: 40, duration: 0.6 })
  .from('.card', {
    opacity: 0,
    y: 60,
    duration: 0.5,
    stagger: 0.12
  }, '-=0.2')
  .from('.footer-link', {
    opacity: 0,
    x: -20,
    duration: 0.3,
    stagger: 0.08
  }, '-=0.3');
```

Dynamic Stagger with Functions

```
gsap.from('.item', {
  opacity: 0,
  y: (index, target) => {
    // Each element gets a different y offset
    return 30 + (index * 10);
  },
  rotation: (index) => index % 2 === 0 ? -10 : 10,
  duration: 0.5,
  stagger: {
    each: 0.1,
    from: 'random'
  }
});
```

Chapter 7: GSAP in Svelte — Integration Patterns

The Svelte Lifecycle Connection

GSAP animations must be created after the DOM is available and cleaned up when components are destroyed.

Svelte 5 Pattern with $effect

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';

  gsap.registerPlugin(ScrollTrigger);

  let container = $state(null);
  $effect(() => {
    if (!container) return;

    const ctx = gsap.context(() => {
      gsap.from('.hero-title', {
        opacity: 0,
        y: 60,
        duration: 0.8,
        ease: 'power3.out'
      });

      gsap.from('.hero-subtitle', {
        opacity: 0,
        y: 40,
        duration: 0.6,
        delay: 0.3,
        ease: 'power2.out'
      });
    }, container);

    return () => ctx.revert();
  });
</script>

<div bind:this={container}>
  <h1 class="hero-title">Welcome</h1>
  <p class="hero-subtitle">Animated with GSAP</p>
</div>
```

gsap.context() — The Cleanup Pattern

`gsap.context()` scopes all animations to a container and provides a single `.revert()` call for cleanup:

```
<script>
  import gsap from 'gsap';

  let wrapper = $state(null);

  $effect(() => {
    if (!wrapper) return;

    const ctx = gsap.context(() => {
      // All selectors scoped to wrapper
      gsap.from('.title', { opacity: 0, y: 30, duration: 0.5 });
      gsap.from('.card', { opacity: 0, y: 50, stagger: 0.1 });
    }, wrapper);

    // Cleanup: kills all animations, reverts all changes
    return () => ctx.revert();
  });
</script>

<section bind:this={wrapper}>
  <h2 class="title">Features</h2>
  <div class="card">Card 1</div>
  <div class="card">Card 2</div>
  <div class="card">Card 3</div>
</section>
```

Creating a Reusable Animation Action

```
// src/lib/actions/gsapReveal.js
import gsap from 'gsap';
import { ScrollTrigger } from 'gsap/ScrollTrigger';

gsap.registerPlugin(ScrollTrigger);

export function gsapReveal(node, params = {}) {
  const {
    y = 40,
    opacity = 0,
    duration = 0.6,
    ease = 'power2.out',
    start = 'top 85%',
    delay = 0
  } = params;

  gsap.set(node, { opacity, y });

  const tween = gsap.to(node, {
    opacity: 1,
    y: 0,
    duration,
    ease,
    delay,
    scrollTrigger: {
      trigger: node,
      start,
      toggleActions: 'play none none reverse'
    }
  });

  return {
    destroy() {
      tween.scrollTrigger?.kill();
      tween.kill();
    }
  };
}
```

Usage in a component:

```
<script>
  import { gsapReveal } from '$lib/actions/gsapReveal';
</script>

<h2 use:gsapReveal>Fade In on Scroll</h2>
<p use:gsapReveal={{ y: 60, duration: 0.8, delay: 0.2 }}>
  Custom parameters
</p>
```

Reusable Timeline Factory

```
// src/lib/animations/sectionReveal.js
import gsap from 'gsap';

export function createSectionReveal(container) {
  const tl = gsap.timeline({ paused: true });

  const ctx = gsap.context(() => {
    tl.from('.section-badge', {
      opacity: 0, scale: 0.8, duration: 0.4, ease: 'back.out(1.7)'
    })
    .from('.section-heading', {
      opacity: 0, y: 40, duration: 0.6, ease: 'power3.out'
    }, '-=0.2')
    .from('.section-description', {
      opacity: 0, y: 30, duration: 0.5
    }, '-=0.3')
    .from('.section-card', {
      opacity: 0, y: 50, stagger: 0.12, duration: 0.5
    }, '-=0.2');
  }, container);

  return {
    timeline: tl,
    play: () => tl.play(),
    revert: () => ctx.revert()
  };
}
```

Reactive Animations with $effect

```
<script>
  import gsap from 'gsap';

  let isOpen = $state(false);
  let menuRef = $state(null);

  $effect(() => {
    if (!menuRef) return;

    if (isOpen) {
      gsap.to(menuRef, {
        height: 'auto',
        opacity: 1,
        duration: 0.4,
        ease: 'power2.out'
      });
    } else {
      gsap.to(menuRef, {
        height: 0,
        opacity: 0,
        duration: 0.3,
        ease: 'power2.in'
      });
    }
  });
</script>

<button onclick={() => isOpen = !isOpen}>Toggle Menu</button>
<nav bind:this={menuRef} style="overflow: hidden; height: 0; opacity: 0;">
  <a href="/about">About</a>
  <a href="/work">Work</a>
  <a href="/contact">Contact</a>
</nav>
```

SSR Safety

GSAP depends on the DOM. In SvelteKit with SSR, guard all GSAP code:

```
<script>
  import { browser } from '$app/environment';

  $effect(() => {
    if (!browser) return;

    // Safe to use GSAP here
    import('gsap').then(({ default: gsap }) => {
      gsap.from('.title', { opacity: 0, y: 30, duration: 0.6 });
    });
  });
</script>
```

Or use dynamic imports in a utility:

```
// src/lib/utils/gsap.js
export async function loadGSAP() {
  const { default: gsap } = await import('gsap');
  const { ScrollTrigger } = await import('gsap/ScrollTrigger');
  gsap.registerPlugin(ScrollTrigger);
  return { gsap, ScrollTrigger };
}
```

Chapter 8: Hero Animations — First Impressions

The Cinematic Hero Entrance

```
<script>
  import gsap from 'gsap';

  let hero = $state(null);

  $effect(() => {
    if (!hero) return;

    const ctx = gsap.context(() => {
      const tl = gsap.timeline({
        defaults: { ease: 'power3.out' }
      });

      // Set initial states
      gsap.set('.hero-line', { overflow: 'hidden' });
      gsap.set('.hero-line span', { yPercent: 100 });
      gsap.set('.hero-subtitle', { opacity: 0, y: 20 });
      gsap.set('.hero-cta', { opacity: 0, y: 20 });
      gsap.set('.hero-visual', { opacity: 0, scale: 0.9 });

      // Sequence the reveal
      tl.to('.hero-line span', {
        yPercent: 0,
        duration: 0.8,
        stagger: 0.12
      })
      .to('.hero-subtitle', {
        opacity: 1, y: 0, duration: 0.6
      }, '-=0.3')
      .to('.hero-cta', {
        opacity: 1, y: 0, duration: 0.5
      }, '-=0.2')
      .to('.hero-visual', {
        opacity: 1, scale: 1, duration: 0.8
      }, '-=0.4');
    }, hero);

    return () => ctx.revert();
  });
</script>

<section bind:this={hero} class="hero">
  <div class="hero-content">
    <h1>
      <div class="hero-line"><span>Build Beautiful</span></div>
      <div class="hero-line"><span>Web Experiences</span></div>
      <div class="hero-line"><span>That Convert</span></div>
    </h1>
    <p class="hero-subtitle">
      Modern tools for modern makers.
    </p>
    <button class="hero-cta">Get Started</button>
  </div>
  <div class="hero-visual">
    <img src="/hero-illustration.svg" alt="Hero" />
  </div>
</section>

<style>
  .hero-line {
    overflow: hidden;
    line-height: 1.1;
  }
  .hero-line span {
    display: inline-block;
  }
</style>
```

Split Text Hero Animation

```
// src/lib/animations/splitText.js
export function splitTextIntoSpans(element) {
  const text = element.textContent;
  element.innerHTML = '';

  return text.split('').map((char) => {
    const span = document.createElement('span');
    span.textContent = char === ' ' ? '\u00A0' : char;
    span.style.display = 'inline-block';
    element.appendChild(span);
    return span;
  });
}
```

```
<script>
  import gsap from 'gsap';
  import { splitTextIntoSpans } from '$lib/animations/splitText';

  let heading = $state(null);

  $effect(() => {
    if (!heading) return;

    const chars = splitTextIntoSpans(heading);

    gsap.from(chars, {
      opacity: 0,
      y: 80,
      rotationX: -90,
      stagger: 0.02,
      duration: 0.6,
      ease: 'back.out(1.7)'
    });

    return () => {
      // Restore original text
      heading.textContent = heading.dataset.text;
    };
  });
</script>

<h1 bind:this={heading} data-text="ShipForge">ShipForge</h1>
```

Parallax Hero with Depth Layers

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';
  gsap.registerPlugin(ScrollTrigger);

  let heroSection = $state(null);

  $effect(() => {
    if (!heroSection) return;

    const ctx = gsap.context(() => {
      // Different scroll speeds create depth
      gsap.to('.layer-bg', {
        y: 200,
        scrollTrigger: {
          trigger: heroSection,
          start: 'top top',
          end: 'bottom top',
          scrub: true
        }
      });

      gsap.to('.layer-mid', {
        y: 100,
        scrollTrigger: {
          trigger: heroSection,
          start: 'top top',
          end: 'bottom top',
          scrub: true
        }
      });

      gsap.to('.layer-fg', {
        y: 30,
        scrollTrigger: {
          trigger: heroSection,
          start: 'top top',
          end: 'bottom top',
          scrub: true
        }
      });

      // Text fades out as user scrolls
      gsap.to('.hero-text', {
        opacity: 0,
        y: -80,
        scrollTrigger: {
          trigger: heroSection,
          start: 'top top',
          end: '40% top',
          scrub: true
        }
      });
    }, heroSection);

    return () => ctx.revert();
  });
</script>
```

```
<section bind:this={heroSection} class="hero-parallax">
  <div class="layer-bg"><!-- Background layer --></div>
  <div class="layer-mid"><!-- Middle layer --></div>
  <div class="layer-fg"><!-- Foreground layer --></div>
  <div class="hero-text">
    <h1>Scroll to Explore</h1>
  </div>
</section>
```

Counter Animation

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';

  gsap.registerPlugin(ScrollTrigger);

  let statsSection = $state(null);
  let counter1 = $state(0);
  let counter2 = $state(0);
  let counter3 = $state(0);

  const stats = { val1: 0, val2: 0, val3: 0 };

  $effect(() => {
    if (!statsSection) return;

    const ctx = gsap.context(() => {
      gsap.to(stats, {
        val1: 2500,
        val2: 98,
        val3: 150,
        duration: 2,
        ease: 'power2.out',
        scrollTrigger: {
          trigger: statsSection,
          start: 'top 70%',
          toggleActions: 'play none none reverse'
        },
        onUpdate: () => {
          counter1 = Math.round(stats.val1);
          counter2 = Math.round(stats.val2);
          counter3 = Math.round(stats.val3);
        }
      });
    }, statsSection);

    return () => ctx.revert();
  });
</script>

<section bind:this={statsSection} class="stats">
  <div class="stat">
    <span class="stat-number">{counter1.toLocaleString()}+</span>
    <span class="stat-label">Users</span>
  </div>
  <div class="stat">
    <span class="stat-number">{counter2}%</span>
    <span class="stat-label">Satisfaction</span>
  </div>
  <div class="stat">
    <span class="stat-number">{counter3}+</span>
    <span class="stat-label">Projects</span>
  </div>
</section>
```

Chapter 9: Scroll-Driven Storytelling

Horizontal Scroll Section

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';

  gsap.registerPlugin(ScrollTrigger);

  let horizontalSection = $state(null);
  let track = $state(null);

  $effect(() => {
    if (!horizontalSection || !track) return;

    const ctx = gsap.context(() => {
      const panels = gsap.utils.toArray('.panel');
      const totalWidth = panels.length * window.innerWidth;

      gsap.to(track, {
        x: -(totalWidth - window.innerWidth),
        ease: 'none',
        scrollTrigger: {
          trigger: horizontalSection,
          start: 'top top',
          end: () => `+=${totalWidth}`,
          pin: true,
          scrub: 1,
          anticipatePin: 1,
          invalidateOnRefresh: true
        }
      });
    }, horizontalSection);

    return () => ctx.revert();
  });
</script>

<section bind:this={horizontalSection} class="horizontal-scroll">
  <div bind:this={track} class="track">
    <div class="panel">
```

```
      <div class="panel-content">
        <h2>Step 1: Design</h2>
        <p>Craft your vision with modern design tools.</p>
      </div>
    </div>
    <div class="panel">
      <div class="panel-content">
        <h2>Step 2: Develop</h2>
        <p>Build with cutting-edge frameworks.</p>
      </div>
    </div>
    <div class="panel">
      <div class="panel-content">
        <h2>Step 3: Deploy</h2>
        <p>Ship to production with confidence.</p>
      </div>
    </div>
    <div class="panel">
      <div class="panel-content">
        <h2>Step 4: Scale</h2>
        <p>Grow without limits.</p>
      </div>
    </div>
  </div>
</section>

<style>
  .horizontal-scroll {
    overflow: hidden;
  }
  .track {
    display: flex;
    width: max-content;
  }
  .panel {
    width: 100vw;
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
  }
</style>
```

Progress-Driven Story Sections

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';

  gsap.registerPlugin(ScrollTrigger);

  let storyContainer = $state(null);
  let progress = $state(0);

  $effect(() => {
    if (!storyContainer) return;

    const ctx = gsap.context(() => {
      const tl = gsap.timeline({
        scrollTrigger: {
          trigger: storyContainer,
          start: 'top top',
          end: '+=3000',
          pin: true,
          scrub: 1,
          onUpdate: (self) => {
            progress = Math.round(self.progress * 100);
          }
        }
      });

      // Chapter 1: Fade in the problem
      tl.from('.story-chapter-1', { opacity: 0, y: 100, duration: 1 })
        .to('.story-chapter-1', { opacity: 0, y: -100, duration: 0.5 }, '+=0.5');

      // Chapter 2: Reveal the solution
      tl.from('.story-chapter-2', { opacity: 0, scale: 0.8, duration: 1 })
        .to('.story-chapter-2', { opacity: 0, scale: 1.2, duration: 0.5 }, '+=0.5');

      // Chapter 3: Show the result
      tl.from('.story-chapter-3', { opacity: 0, x: 200, duration: 1 })
        .from('.story-chapter-3 .stat', {
          innerText: 0,
          snap: { innerText: 1 },
          stagger: 0.2,
          duration: 0.5
        }, '-=0.5');

    }, storyContainer);

    return () => ctx.revert();
  });
</script>

<section bind:this={storyContainer} class="story">
  <div class="progress-bar" style="width: {progress}%"></div>

  <div class="story-chapter-1">
    <h2>The Problem</h2>
    <p>Building modern web apps is hard.</p>
  </div>

  <div class="story-chapter-2">
    <h2>The Solution</h2>
    <p>ShipForge makes it effortless.</p>
  </div>

  <div class="story-chapter-3">
    <h2>The Results</h2>
    <div class="stats-row">
      <span class="stat">0</span>% faster
      <span class="stat">0</span>% easier
    </div>
  </div>
</section>
```

Scroll-Triggered SVG Path Drawing

```
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';

  gsap.registerPlugin(ScrollTrigger);

  let svgSection = $state(null);

  $effect(() => {
    if (!svgSection) return;

    const ctx = gsap.context(() => {
      const path = svgSection.querySelector('.draw-path');
      const length = path.getTotalLength();

      gsap.set(path, {
        strokeDasharray: length,
        strokeDashoffset: length
      });

      gsap.to(path, {
        strokeDashoffset: 0,
        ease: 'none',
        scrollTrigger: {
          trigger: svgSection,
          start: 'top center',
          end: 'bottom center',
          scrub: 1
        }
      });
    }, svgSection);

    return () => ctx.revert();
  });
</script>

<section bind:this={svgSection} class="svg-draw-section">
  <svg viewBox="0 0 800 400" xmlns="http://www.w3.org/2000/svg">
    <path
      class="draw-path"
      d="M 50 200 C 150 50, 350 50, 400 200 S 650 350, 750 200"
      fill="none"
      stroke="currentColor"
      stroke-width="3"
    />
  </svg>
</section>
```

Image Reveal on Scroll

```js
// src/lib/animations/imageReveal.js
import gsap from 'gsap';
import { ScrollTrigger } from 'gsap/ScrollTrigger';

gsap.registerPlugin(ScrollTrigger);

export function imageReveal(node) {
  const overlay = document.createElement('div');
  overlay.style.cssText =
    'position: absolute; inset: 0; background: var(--color-primary, #6366f1); transform-origin: left; z-index: 1;';
  node.style.position = 'relative';
  node.style.overflow = 'hidden';
  node.appendChild(overlay);

  const img = node.querySelector('img');
  gsap.set(img, { scale: 1.3 });

  const tl = gsap.timeline({
    scrollTrigger: {
      trigger: node,
      start: 'top 75%',
      toggleActions: 'play none none reverse'
    }
  });

  tl.to(overlay, {
    scaleX: 0,
    transformOrigin: 'right',
    duration: 0.8,
    ease: 'power3.inOut'
  })
  .to(img, {
    scale: 1,
    duration: 1.2,
    ease: 'power2.out'
  }, '-=0.4');

  return {
    destroy() {
      tl.scrollTrigger?.kill();
      tl.kill();
      overlay.remove();
    }
  };
}
```

Chapter 10: Performance — Silky Smooth 60fps

The Golden Rules

1. **Animate transforms and opacity only** — these are GPU-composited
2. **Avoid layout triggers** — width, height, top, left, margin, padding

3. **Use `will-change` sparingly** — only on elements about to animate
4. **Kill animations on cleanup** — prevent memory leaks
5. **Prefer `scrub` over scroll events** — GSAP handles throttling

Transform vs Layout Properties

```
// GOOD — GPU-accelerated, no layout recalculation
gsap.to('.element', {
  x: 100,           // transform: translateX
  y: 50,            // transform: translateY
  scale: 1.2,       // transform: scale
  rotation: 45,     // transform: rotate
  opacity: 0.8      // opacity
});

// BAD — Triggers layout recalculation every frame
gsap.to('.element', {
  left: 100,        // Layout trigger
  top: 50,          // Layout trigger
  width: 200,       // Layout trigger
  height: 100,      // Layout trigger
  marginLeft: 20    // Layout trigger
});
```

will-change Strategy

```
<script>
  import gsap from 'gsap';

  let box = $state(null);

  $effect(() => {
    if (!box) return;

    // Add will-change before animation
    gsap.set(box, { willChange: 'transform, opacity' });

    const tween = gsap.to(box, {
      x: 300,
      opacity: 0.5,
      duration: 1,
      onComplete: () => {
        // Remove will-change after animation
        gsap.set(box, { willChange: 'auto' });
      }
    });

    return () => tween.kill();
  });
</script>

<div bind:this={box} class="animated-box">Performant</div>
```

Lazy Initialization

```
// Only create ScrollTriggers when elements are near the viewport
function lazyScrollTrigger(trigger, animationFn) {
  ScrollTrigger.create({
    trigger,
    start: 'top bottom+=200', // 200px before entering viewport
    once: true,
    onEnter: () => animationFn()
  });
}

lazyScrollTrigger('.features-section', () => {
  gsap.from('.feature-card', {
    opacity: 0,
    y: 60,
    stagger: 0.15,
    duration: 0.6,
    scrollTrigger: {
      trigger: '.features-section',
      start: 'top 80%'
    }
  });
});
```

Debounced Refresh

```
// Batch ScrollTrigger refreshes
let refreshTimeout;
function debouncedRefresh() {
  clearTimeout(refreshTimeout);
  refreshTimeout = setTimeout(() => {
    ScrollTrigger.refresh();
  }, 200);
}

// Call after dynamic content loads
window.addEventListener('resize', debouncedRefresh);
```

Animation Budgeting

```
// Limit simultaneous animations on lower-end devices
const isLowEnd = navigator.hardwareConcurrency <= 4
  || navigator.deviceMemory <= 4;

function animateCards(cards) {
  if (isLowEnd) {
    // Simple fade only
    gsap.from(cards, { opacity: 0, duration: 0.3, stagger: 0.05 });
  } else {
    // Full cinematic entrance
    gsap.from(cards, {
      opacity: 0,
      y: 60,
      rotation: -5,
      scale: 0.9,
      duration: 0.6,
      stagger: {
        each: 0.12,
        from: 'center'
      },
      ease: 'back.out(1.7)'
    });
  }
}
```

Reduced Motion Respect

```
// src/lib/utils/motion.js
export function prefersReducedMotion() {
  return window.matchMedia('(prefers-reduced-motion: reduce)').matches;
}

export function safeAnimate(target, vars) {
  if (prefersReducedMotion()) {
    // Instant state change, no animation
    gsap.set(target, {
      opacity: vars.opacity ?? 1,
      x: vars.x ?? 0,
      y: vars.y ?? 0,
      scale: vars.scale ?? 1,
      rotation: vars.rotation ?? 0
    });
    return null;
  }
  return gsap.to(target, vars);
}
```

Memory Leak Prevention Checklist

```
// Pattern 1: Always use gsap.context()
$effect(() => {
  const ctx = gsap.context(() => {
    // All animations here
  }, containerRef);
  return () => ctx.revert(); // Kills everything
});

// Pattern 2: Kill individual tweens
$effect(() => {
  const tween = gsap.to('.box', { x: 100 });
  return () => tween.kill();
});

// Pattern 3: Kill ScrollTriggers explicitly
$effect(() => {
  const st = ScrollTrigger.create({
    trigger: '.section',
    start: 'top center',
    onEnter: () => { /* ... */ }
  });
  return () => st.kill();
});

// Pattern 4: Kill all ScrollTriggers on page navigation
// In +layout.svelte
import { beforeNavigate } from '$app/navigation';
beforeNavigate(() => {
  ScrollTrigger.getAll().forEach(st => st.kill());
});
```

Chapter 11: Production Patterns — Ship It

Animation Registry Pattern

```
// src/lib/animations/registry.js
import gsap from 'gsap';
import { ScrollTrigger } from 'gsap/ScrollTrigger';
import { prefersReducedMotion } from '$lib/utils/motion';

gsap.registerPlugin(ScrollTrigger);

// Centralized animation definitions
const animations = {
  fadeUp: (target, vars = {}) => ({
    opacity: 0,
    y: vars.y ?? 40,
    duration: vars.duration ?? 0.6,
    ease: vars.ease ?? 'power2.out',
    delay: vars.delay ?? 0,
    stagger: vars.stagger ?? 0
  }),

  scaleIn: (target, vars = {}) => ({
    opacity: 0,
    scale: vars.scale ?? 0.8,
    duration: vars.duration ?? 0.5,
    ease: vars.ease ?? 'back.out(1.7)',
    delay: vars.delay ?? 0
  }),

  slideInLeft: (target, vars = {}) => ({
    opacity: 0,
    x: vars.x ?? -60,
    duration: vars.duration ?? 0.6,
    ease: vars.ease ?? 'power3.out',
    delay: vars.delay ?? 0
  }),

  slideInRight: (target, vars = {}) => ({
    opacity: 0,
    x: vars.x ?? 60,
    duration: vars.duration ?? 0.6,
    ease: vars.ease ?? 'power3.out',
    delay: vars.delay ?? 0
  })
};

export function animate(target, animationName, vars = {}) {
  if (prefersReducedMotion()) {
    gsap.set(target, { opacity: 1, x: 0, y: 0, scale: 1 });
    return null;
  }

  const animVars = animations[animationName]?.(target, vars);
  if (!animVars) {
    console.warn(`Animation "${animationName}" not found in registry`);
    return null;
  }

  return gsap.from(target, animVars);
}

export function scrollAnimate(target, animationName, triggerVars = {}, animVars = {}) {
  if (prefersReducedMotion()) {
    gsap.set(target, { opacity: 1, x: 0, y: 0, scale: 1 });
    return null;
  }

  const animation = animations[animationName]?.(target, animVars);
  if (!animation) return null;

  return gsap.from(target, {
    ...animation,
    scrollTrigger: {
      trigger: triggerVars.trigger ?? target,
      start: triggerVars.start ?? 'top 85%',
      toggleActions: triggerVars.toggleActions ?? 'play none none reverse',
      ...triggerVars
    }
  });
}
```

Page Transition System

```
<!-- src/lib/components/PageTransition.svelte -->
<script>
  import gsap from 'gsap';
  import { beforeNavigate, afterNavigate } from '$app/navigation';

  let { children } = $props();
  let overlay = $state(null);

  beforeNavigate((navigation) => {
    if (navigation.from?.url.pathname === navigation.to?.url.pathname) return;

    return new Promise((resolve) => {
      gsap.to(overlay, {
        scaleY: 1,
        transformOrigin: 'bottom',
        duration: 0.4,
        ease: 'power3.inOut',
        onComplete: resolve
      });
    });
  });

  afterNavigate(() => {
    gsap.fromTo(overlay,
      { scaleY: 1, transformOrigin: 'top' },
      {
        scaleY: 0,
        duration: 0.4,
        ease: 'power3.inOut',
        delay: 0.1
      }
    );
  });
</script>

<div
  bind:this={overlay}
  class="page-transition-overlay"
  style="
    position: fixed;
    inset: 0;
    background: var(--color-primary);
    z-index: 9999;
    transform: scaleY(0);
    pointer-events: none;
  "
></div>

{@render children()}
```

Component Animation Presets

```
<!-- src/lib/components/AnimatedSection.svelte -->
<script>
  import gsap from 'gsap';
  import { ScrollTrigger } from 'gsap/ScrollTrigger';
  import { browser } from '$app/environment';

  gsap.registerPlugin(ScrollTrigger);

  let {
    animation = 'fadeUp',
    stagger = 0.1,
    duration = 0.6,
    delay = 0,
    start = 'top 85%',
    children
  } = $props();
  let section = $state(null);

  const presets = {
    fadeUp: { opacity: 0, y: 40 },
    fadeDown: { opacity: 0, y: -40 },
    fadeLeft: { opacity: 0, x: -40 },
    fadeRight: { opacity: 0, x: 40 },
    scaleUp: { opacity: 0, scale: 0.85 },
    rotateIn: { opacity: 0, rotation: -10, y: 30 }
  };

  $effect(() => {
    if (!browser || !section) return;

    const ctx = gsap.context(() => {
      const targets = section.querySelectorAll('[data-animate]');
      if (targets.length === 0) return;

      gsap.from(targets, {
        ...presets[animation],
        duration,
        delay,
        stagger,
        ease: 'power2.out',
        scrollTrigger: {
          trigger: section,
          start,
          toggleActions: 'play none none reverse'
        }
      });
    }, section);

    return () => ctx.revert();
  });
</script>

<section bind:this={section}>
  {@render children()}
</section>
```

Usage:

```
<AnimatedSection animation="fadeUp" stagger={0.15}>
  <h2 data-animate>Our Features</h2>
  <div data-animate>Feature 1</div>
  <div data-animate>Feature 2</div>
  <div data-animate>Feature 3</div>
</AnimatedSection>
```

Debug Mode for Development

```
// src/lib/animations/debug.js
import { ScrollTrigger } from 'gsap/ScrollTrigger';
import gsap from 'gsap';
import { dev } from '$app/environment';

export function setupAnimationDebug() {
  if (!dev) return;

  // Show all ScrollTrigger markers
  ScrollTrigger.defaults({ markers: true });

  // Global animation speed control
  window.gsapSpeed = (speed) => {
    gsap.globalTimeline.timeScale(speed);
    console.log(`GSAP global speed: ${speed}x`);
  };

  // Pause all animations
  window.gsapPause = () => {
    gsap.globalTimeline.pause();
    console.log('GSAP paused');
  };

  // Resume all animations
  window.gsapResume = () => {
    gsap.globalTimeline.resume();
    console.log('GSAP resumed');
  };

  // List all active ScrollTriggers
  window.gsapTriggers = () => {
    const triggers = ScrollTrigger.getAll();
    console.table(triggers.map(st => ({
      id: st.vars.id || 'unnamed',
      trigger: st.trigger,
      start: st.start,
      end: st.end,
      progress: st.progress.toFixed(2)
    })));
  };

  console.log('GSAP Debug mode active. Available: gsapSpeed(n), gsapPause(), gsapResume(),
gsapTriggers()');
}
```

Production Build Checklist

```
// Before shipping, ensure:

// 1. Remove all debug markers
ScrollTrigger.defaults({ markers: false }); // or just omit markers

// 2. All animations have cleanup
$effect(() => {
  const ctx = gsap.context(() => { /* ... */ }, ref);
  return () => ctx.revert(); // ALWAYS
});
```

```
// 3. Reduced motion is respected
// if (prefersReducedMotion()) { /* Skip or simplify */ }

// 4. ScrollTrigger.refresh() called after dynamic content
// afterNavigate(() => {
//   requestAnimationFrame(() => { ScrollTrigger.refresh(); });
// });

// 5. No layout-triggering properties animated
// Use x/y instead of left/top
// Use scale instead of width/height

// 6. Tree-shake unused plugins
// Only import what you use:
// import gsap from 'gsap';
// import { ScrollTrigger } from 'gsap/ScrollTrigger';
// NOT: import 'gsap/all';

// 7. Test on real devices
// Chrome DevTools Performance tab
// Aim for < 16ms per frame (60fps)
```

Final Project Structure

```
src/lib/
├── actions/
│   ├── gsapReveal.js          # Svelte action for scroll reveals
│   └── imageReveal.js         # Svelte action for image reveals
├── animations/
│   ├── registry.js            # Centralized animation definitions
│   ├── sectionReveal.js       # Reusable section timeline
│   ├── splitText.js           # Text splitting utility
│   ├── cleanup.js             # Cleanup patterns
│   └── debug.js               # Development debug tools
├── components/
│   ├── AnimatedSection.svelte # Declarative scroll animations
│   └── PageTransition.svelte  # Route transition overlay
└── utils/
    ├── gsap.js                # Lazy GSAP loader
    └── motion.js              # Reduced motion detection
```

Summary

Chapter	Key Takeaway
1. GSAP Intro	GSAP animates anything JS can touch
2. Tweens	to, from, fromTo are your core tools
3. Timelines	Position parameter is the choreography secret
4. Easing	power2.out for entrances, power2.in for exits
5. ScrollTrigger	scrub + pin = scroll-driven magic
6. Staggers	grid + from patterns create visual waves

7. Svelte Integration	gsap.context() + $effect = clean lifecycle
8. Hero Animations	First impression = split text + parallax + counters
9. Scroll Storytelling	Pin + timeline = narrative sections
10. Performance	Transforms only, will-change, reduced motion
11. Production	Registry pattern, page transitions, debug tools

---

End of GSAP Cinematic Animation Masterclass for Svelte