Advanced Notes (React + GSAP)

Use useGSAP() instead of useEffect() refer https://gsap.com/resources/React/ for latest updates

1. Introduction to GSAP

GSAP (GreenSock Animation Platform) is a powerful and high-performance JavaScript library used for creating rich animations. It's optimized for speed and efficiency, offering a smooth, consistent experience even with complex, high-volume animations.

Why GSAP over CSS for animations?

- ***Performance***: GSAP provides smoother animations with better performance than CSS animations.
- ***Flexibility***: It offers more control over animation sequencing, timing, and easing.
- ***Advanced Features***: GSAP can handle complex animations, including physics-based animation, SVG animations, 3D transforms, and scroll-based animations.
- ***Cross-Browser Compatibility***: Works seamlessly across different browsers, even older versions of Internet Explorer.

2. Setting Up GSAP in React

Before you start, you need to install GSAP. Use npm or yarn to install it in your React project:

```
# Install the GSAP library
npm install gsap

# Install the GSAP React package
npm install @gsap/react
```

For advanced features like ScrollTrigger or ScrollSmoother, you'll need to install the gsap version that supports plugins:

```
npm install gsap@3.12.2
```

3. Advanced GSAP Methods in React

GSAP's flexibility comes from its comprehensive API. Let's go deeper into its key methods and advanced functionalities.

3.1. gsap.to() and gsap.from()

- gsap.to(): Used to animate an element to a set of values.
- gsap.from(): Used to animate an element from a set of values to its current state.

Example 1: gsap.to() with Advanced Easing and Callbacks

```
import React, { useRef, useEffect } from 'react';
import { gsap } from 'gsap';

const AnimationExample = () => {
  const boxRef = useRef(null);

  useEffect(() => {
    gsap.to(boxRef.current, {
      x: 500,
      rotation: 180,
      duration: 2,
      ease: "power4.out", // Custom easing for smoothness
      onComplete: () => {
        console.log("Animation Complete!");
      },
      onStart: () => {
        console.log("Animation Started!");
      }
    });
  }, []);

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

using useGSAP()

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

const AnimationExample = () => {
  const boxRef = useRef(null);

  useGSAP(() => {
    gsap.to(boxRef.current, {
      x: 500,
      rotation: 180,
      duration: 2,
      ease: "power4.out", // Custom easing for smoothness
      onComplete: () => {
```

```
      console.log("Animation Complete!");
    },
    onStart: () => {
      console.log("Animation Started!");
    }
  });
 }, []); // Empty dependency array for running once

 return <div ref={boxRef} className="box">Animate Me!</div>;
};

export default AnimationExample;
```

In this example, the box moves horizontally (x: 500) and rotates (rotation: 180) over 2 seconds with a "power4" easing curve.

Advanced Easing: GSAP supports various easing functions like power2.in, bounce.out, elastic.inOut, etc., to control the animation's timing.

Callbacks: onStart, onUpdate, onComplete, and onRepeat can be used to handle events during the animation lifecycle.

3.2. gsap.fromTo()

- gsap.fromTo(): Allows you to define both starting and ending values for an animation.

Example 2: gsap.fromTo() with Delays

```
useEffect(() => {
 gsap.fromTo(
   boxRef.current,
   { opacity: 0, scale: 0 }, // Initial state
   { opacity: 1, scale: 1, duration: 2, delay: 0.5, ease: "back.out(1.7)" } // Final state with delay
 );
}, []);
```

using useGSAP()

```
import { useGSAP } from "@gsap/react";
useGSAP(() => {
 gsap.fromTo(
   boxRef.current,
   { opacity: 0, scale: 0 }, // Initial state
   { opacity: 1, scale: 1, duration: 2, delay: 0.5, ease: "back.out(1.7)" } // Final state with delay
 );
});
```

In this example, the element starts at opacity: 0 and scale: 0 and animates to opacity: 1 and scale: 1. The delay: 0.5 adds a half-second delay before the animation begins.

4. Creating Complex Animations with GSAP Timelines

GSAP's Timeline feature allows you to sequence multiple animations in a structured way. This is crucial for creating complex animations that involve multiple elements.

4.1. Timeline Basics

gsap.timeline(): A timeline object that lets you chain animations together. Timelines provide control over playback and synchronization.

Example 3: Using gsap.timeline() for Sequential Animations

```
useEffect(() => {
  const tl = gsap.timeline();
  tl.to(boxRef1.current, { x: 100, duration: 1 })
    .to(boxRef2.current, { y: 100, duration: 1 })
    .to(boxRef1.current, { rotation: 360, duration: 1 })
    .to(boxRef2.current, { scale: 1.5, duration: 1 });
}, []);
```

using useGSAP()

```
import { useGSAP } from "@gsap/react";
useGSAP(() => {
  const tl = gsap.timeline();
  tl.to(boxRef1.current, { x: 100, duration: 1 })
    .to(boxRef2.current, { y: 100, duration: 1 })
    .to(boxRef1.current, { rotation: 360, duration: 1 })
    .to(boxRef2.current, { scale: 1.5, duration: 1 });
});
```

Chaining: Each to() or from() method in the timeline runs sequentially.

stagger: GSAP's stagger option is used to create staggered effects across multiple elements.

4.2. Controlling Timeline Playback

You can control a timeline's playback programmatically:

tl.play(): Starts or resumes the timeline.
tl.pause(): Pauses the timeline.
tl.reverse(): Reverses the timeline's direction.
tl.seek(time): Jumps to a specific point in the timeline.

Example 4: Interactive Timeline Control

```
useEffect(() => {
  const tl = gsap.timeline({ paused: true });
  tl.to(boxRef.current, { x: 200, duration: 1 });

  // Play animation on a button click
  document.getElementById("playBtn").addEventListener("click", () => {
    tl.play();
  });
}, []);
```

using useGSAP()

```
import { useGSAP } from "@gsap/react";
useGSAP(() => {
  const tl = gsap.timeline({ paused: true });
  tl.to(boxRef.current, { x: 200, duration: 1 });

  // Play animation on a button click
  document.getElementById("playBtn").addEventListener("click", () => {
    tl.play();
  });
});
```

5. Advanced GSAP Features

5.1. GSAP Plugins

GSAP's plugins are powerful tools that expand the library's functionality. Below are some advanced plugins that can be used in React:

ScrollTrigger: Creates animations that are triggered when an element scrolls into view.
Scroll Smoother: Provides smooth scrolling effects.
MotionPathPlugin: Makes it easy to animate elements along a motion path (SVG or other).

Example 5: ScrollTrigger for Scroll-based Animation

```
import { gsap } from 'gsap';
import { ScrollTrigger } from 'gsap/ScrollTrigger';
gsap.registerPlugin(ScrollTrigger);
useEffect(() => {
  gsap.to(boxRef.current, {
    scrollTrigger: {
      trigger: boxRef.current,
      start: "top 80%",  // When the top of the element reaches 80% of the viewport
      end: "bottom 20%",
      scrub: true,  // Smooth scrolling effect
      markers: true  // Show scroll markers for debugging
    },
    x: 500,
    rotation: 360,
    duration: 3
  });
}, []);
```

using useGSAP()

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

gsap.registerPlugin(ScrollTrigger);

useGSAP(() => {
  gsap.to(boxRef.current, {
    scrollTrigger: {
      trigger: boxRef.current,
      start: "top 80%",  // When the top of the element reaches 80% of the viewport
      end: "bottom 20%",
      scrub: true,  // Smooth scrolling effect
      markers: true  // Show scroll markers for debugging
    },
    x: 500,
    rotation: 360,
    duration: 3
  });
});
```

In this example, the animation is triggered when the element scrolls into view. The scrub property ensures the animation is tied to the scroll position.

5.2. Staggered Animations

To create staggered animations, GSAP provides the stagger feature, which allows you to animate multiple elements with a delay between each.

Example 6: Staggering Animations on Multiple Elements

```
useEffect(() => {
  gsap.to('.box', {
    x: 300,
    duration: 1,
    stagger: 0.2, // Each element will be animated with a 0.2s delay
    ease: 'bounce.out'
  });
}, []);
```

using useGSAP()

```
import { useGSAP } from "@gsap/react";
useGSAP(() => {
  gsap.to('.box', {
    x: 300,
    duration: 1,
    stagger: 0.2, // Each element will be animated with a 0.2s delay
    ease: 'bounce.out'
  });
});
```

This example moves all .box elements 300px along the x-axis with a bounce easing effect, with each element's animation staggered by 0.2 seconds.

6. Advanced React-GSAP Integration

6.1. Using GSAP with React's useState and useEffect

For dynamic animations in React components, you can use state and props to control the animation's trigger points.

Example 7: React State-based Animation

```
import React, { useState, useEffect, useRef } from 'react';
import { gsap } from 'gsap';

const App = () => {
  const [triggerAnimation, setTriggerAnimation] = useState(false);
  const boxRef = useRef(null);

  useEffect(() => {
    if (triggerAnimation) {
      gsap.to(boxRef.current, { x: 500, rotation: 360, duration: 2 });
    }
  }, [triggerAnimation]);

  return (
    <div>
      <button onClick={() => setTriggerAnimation(!triggerAnimation)}>
        Trigger Animation
      </button>
      <div ref={boxRef} className="box">Animate Me!</div>
    </div>
  );
};
export default App;
```

using useGSAP()

```
import React, { useState, useRef } from 'react';
import { gsap } from 'gsap';
import { useGSAP } from "@gsap/react";

const App = () => {
  const [triggerAnimation, setTriggerAnimation] = useState(false);
  const boxRef = useRef(null);

  // Create a timeline ref to control the animation
  const tl = useRef(null);

  useGSAP(() => {
    // Initialize the timeline
    tl.current = gsap.to(boxRef.current, {
      x: 500,
      rotation: 360,
      duration: 2,
      paused: true // Start paused
    });
  });
```

```
  // Watch for triggerAnimation changes
  useGSAP(() => {
    if (triggerAnimation) {
      tl.current.play();
    } else {
      tl.current.reverse();
    }
  }, { dependencies: [triggerAnimation] });

  return (
    <div>
      <button onClick={() => setTriggerAnimation(!triggerAnimation)}>
        Trigger Animation
      </button>
      <div ref={boxRef} className="box">Animate Me!</div>
    </div>
  );
};

export default App;
```

using a single useGSAP with context

```
import React, { useState, useRef } from 'react';
import { gsap } from 'gsap';
import { useGSAP } from "@gsap/react";

const App = () => {
  const [triggerAnimation, setTriggerAnimation] = useState(false);
  const boxRef = useRef(null);
  const containerRef = useRef(null);

  useGSAP(() => {
    const ctx = gsap.context(() => {
      if (triggerAnimation) {
        gsap.to(boxRef.current, { x: 500, rotation: 360, duration: 2 });
      } else {
        gsap.to(boxRef.current, { x: 0, rotation: 0, duration: 2 });
      }
    }, containerRef);

    return () => ctx.revert(); // Cleanup
  }, { dependencies: [triggerAnimation], scope: containerRef });

  return (
    <div ref={containerRef}>
      <button onClick={() => setTriggerAnimation(!triggerAnimation)}>
        Trigger Animation
      </button>
      <div ref={boxRef} className="box">Animate Me!</div>
    </div>
  );
};

export default App;
```

using a click handler

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

const App = () => {
  const boxRef = useRef(null);
  const containerRef = useRef(null);

  useGSAP(() => {
    // Initialize animation if needed
  }, { scope: containerRef });

  const handleClick = () => {
    gsap.to(boxRef.current, {
      x: gsap.getProperty(boxRef.current, "x") === 0 ? 500 : 0,
      rotation: gsap.getProperty(boxRef.current, "rotation") === 0 ? 360 : 0,
      duration: 2
    });
  };

  return (
    <div ref={containerRef}>
      <button onClick={handleClick}>
        Trigger Animation
      </button>
      <div ref={boxRef} className="box">Animate Me!</div>
    </div>
  );
};

export default App;
```

Dynamic Triggering: Here, the setTriggerAnimation function toggles the animation by setting state, and useEffect watches that state to trigger GSAP animations when the state changes.

6.2. Animating in React with React Router

If you're using React Router for navigation, you can use GSAP to animate page transitions.

Example 8: Page Transitions with React Router and GSAP

```
import { useHistory } from 'react-router-dom';
import { gsap } from 'gsap';

const PageTransition = () => {
 const history = useHistory();

 const handleNavigation = (path) => {
  gsap.to('.page', { opacity: 0, duration: 0.5, onComplete: () => {
   history.push(path);
   gsap.to('.page', { opacity: 1, duration: 0.5 });
  } });
 };

 return (
  <div>
   <button onClick={() => handleNavigation('/newPage')}>Go to New Page</button>
   <div className="page">Current Page</div>
  </div>
 );
};
```

using useGSAP()

```
import { useNavigate } from 'react-router-dom'; // Using modern React Router
import { gsap } from 'gsap';
import { useGSAP } from "@gsap/react";
import { useRef } from 'react';

const PageTransition = () => {
 const navigate = useNavigate(); // Modern replacement for useHistory
 const containerRef = useRef(null);
 const pageRef = useRef(null);

 // Initialize GSAP context
 useGSAP(() => {
  // Initial setup if needed
 }, { scope: containerRef });

 const handleNavigation = (path) => {
  const ctx = gsap.context(() => {
   gsap.to(pageRef.current, {
    opacity: 0,
    duration: 0.5,
    onComplete: () => {
     navigate(path);
     gsap.to(pageRef.current, {
      opacity: 1,
      duration: 0.5
     });
    }
```

```
    });
  }, containerRef);

  return () => ctx.revert();
 };

 return (
  <div ref={containerRef}>
   <button onClick={() => handleNavigation('/newPage')}>
    Go to New Page
   </button>
   <div ref={pageRef} className="page">
    Current Page
   </div>
  </div>
 );
};
```

In this example, the page fades out before the route change happens and fades back in after the route has been updated.

7. Best Practices

1. Avoid Direct DOM Manipulation: Use useRef to target DOM elements in React, and avoid manipulating the DOM directly within useEffect.

2. Use GSAP Timelines for Complex Animations: Manage and control multiple animations with timelines for cleaner code and easier animation sequencing.

3. Performance Considerations: Use will-change CSS property or GPU acceleration for animations that are highly performance-sensitive.

4. Reuse Animations with GSAP's stagger and repeat: For animations that need to be reused multiple times (e.g., list animations), define them in reusable functions or components.

5. Optimize for Mobile: Test your animations on different screen sizes and adjust for mobile devices where performance may vary.

By combining GSAP's advanced features with React's component-based architecture, you can create rich, interactive animations that are both performant and highly customizable. With timelines, plugins like ScrollTrigger, and advanced methods like fromTo and stagger, the possibilities are vast for creating engaging user experiences.