La optimización de animaciones en React con librerías como GSAP y Anime.js se centra en lograr transiciones fluidas y eficientes, reduciendo el consumo de recursos del navegador y evitando renderizados innecesarios del componente.  
Las siguientes prácticas son las más relevantes:

| **Técnica** | **Descripción** | **Aplicación práctica** |
| --- | --- | --- |
| **1. Uso de useRef() en lugar de useState()** | Las referencias (ref) permiten almacenar valores o acceder a elementos del DOM sin provocar re-renderizados. | Se utilizan para guardar el estado de la animación o las referencias a los elementos animados. |
| **2. Ejecución de animaciones dentro de useEffect()** | Garantiza que el DOM esté completamente montado antes de inicializar una animación. | Se crea el timeline de GSAP una sola vez dentro de useEffect. |
| **3. Evitar renderizados innecesarios** | Cada actualización de estado vuelve a renderizar el componente, lo cual puede afectar la fluidez de la animación. | Se evita usar useState en animaciones de cada frame; en su lugar, se usa un ref para almacenar si la animación está activa. |
| **4. Utilizar requestAnimationFrame** | Permite que el navegador sincronice la animación con su tasa de refresco, obteniendo un movimiento más suave y eficiente. | Se emplea para crear un bucle de animación controlado para la caja azul. |
| **5. Reutilizar timelines** | Crear el timeline una sola vez y reutilizarlo mejora el rendimiento y evita reinicializaciones costosas. | El timeline de GSAP se guarda en un ref y se controla (reiniciar, pausar, detener) sin recrearlo. |
| **6. Limpieza de efectos** | Detener animaciones al desmontar el componente previene fugas de memoria y errores futuros. | Se usa el retorno de useEffect para llamar a .kill() sobre el timeline de GSAP. |

**Explicación del código hecho:**

| **Elemento** | **Función** |
| --- | --- |
| **useRef()** | Guarda referencias a los elementos del DOM (cajas) y al timeline de GSAP, así como un valor booleano (isAnimatingRef) que indica si las animaciones están activas. Este valor no causa re-renderizados, lo que optimiza el rendimiento. |
| **useState()** | Solo se usa para actualizar el texto del botón (iniciar/detener), sin afectar la lógica interna de animación. |
| **useEffect()** | Se ejecuta una sola vez al montar el componente, creando y configurando el timeline de GSAP. También limpia la animación al desmontar. |
| **requestAnimationFrame()** | Controla el movimiento de la caja azul mediante un bucle sincronizado con los cuadros del navegador, lo que garantiza animaciones suaves y sin saltos. |
| **gsap.timeline()** | Administra las animaciones de la caja verde en secuencia (posición, tamaño y color) de forma fluida y optimizada. |
| **isAnimatingRef** | Permite controlar las animaciones sin depender del estado de React, evitando que la animación se detenga al producirse un re-render. |
| **Limpieza del bucle** | Cuando el usuario detiene las animaciones, se detiene el timeline de GSAP, se cancela el bucle requestAnimationFrame y se restablecen las posiciones iniciales. |

**Resultado esperado del código hecho:**

* El componente muestra dos cajas y un botón central.
* La caja verde (izquierda) se anima con GSAP, cambiando suavemente de posición, tamaño y color mediante un timeline optimizado.
* La caja azul (derecha) se mueve horizontalmente con un movimiento oscilante continuo, controlado mediante requestAnimationFrame.
* Al hacer clic en “Iniciar animaciones”, ambas animaciones empiezan de manera fluida y sincronizada.
* Al presionar nuevamente el botón (“Detener animaciones”), ambas cajas regresan a su posición original y las animaciones se detienen sin errores ni consumo innecesario de CPU.
* Gracias al uso de useRef, requestAnimationFrame y la limpieza del ciclo de vida, las animaciones se ejecutan de forma eficiente, evitando renderizados innecesarios y manteniendo una excelente performance general.