Threads (background effect)

```
import { useEffect, useRef } from 'react';
import { Renderer, Program, Mesh, Triangle, Color } from 'ogl';

import './Threads.css';

const vertexShader = `
attribute vec2 position;
attribute vec2 uv;
varying vec2 vUv;
void main() {
  vUv = uv;
  gl_Position = vec4(position, 0.0, 1.0);
}
`;

const fragmentShader = `
precision highp float;

uniform float iTime;
uniform vec3 iResolution;
uniform vec3 uColor;
uniform float uAmplitude;
uniform float uDistance;
uniform vec2 uMouse;

#define PI 3.1415926538

const int u_line_count = 40;
const float u_line_width = 7.0;
const float u_line_blur = 10.0;

float Perlin2D(vec2 P) {
    vec2 Pi = floor(P);
    vec4 Pf_Pfmin1 = P.xyxy - vec4(Pi, Pi + 1.0);
    vec4 Pt = vec4(Pi.xy, Pi.xy + 1.0);
    Pt = Pt - floor(Pt * (1.0 / 71.0)) * 71.0;
    Pt += vec2(26.0, 161.0).xyxy;
    Pt *= Pt;
    Pt = Pt.xzxz * Pt.yyww;
    vec4 hash_x = fract(Pt * (1.0 / 951.135664));
    vec4 hash_y = fract(Pt * (1.0 / 642.949883));
    vec4 grad_x = hash_x - 0.49999;
    vec4 grad_y = hash_y - 0.49999;
    vec4 grad_results = inversesqrt(grad_x * grad_x + grad_y * grad_y)
        * (grad_x * Pf_Pfmin1.xzxz + grad_y * Pf_Pfmin1.yyww);
    grad_results *= 1.4142135623730950;
    vec2 blend = Pf_Pfmin1.xy * Pf_Pfmin1.xy * Pf_Pfmin1.xy
```

```
            * (Pf_Pfmin1.xy * (Pf_Pfmin1.xy * 6.0 - 15.0) + 10.0);
  vec4 blend2 = vec4(blend, vec2(1.0 - blend));
  return dot(grad_results, blend2.zxzx * blend2.wwyy);
}

float pixel(float count, vec2 resolution) {
  return (1.0 / max(resolution.x, resolution.y)) * count;
}

float lineFn(vec2 st, float width, float perc, float offset, vec2 mouse, float time, float amplitude, float
distance) {
  float split_offset = (perc * 0.4);
  float split_point = 0.1 + split_offset;

  float amplitude_normal = smoothstep(split_point, 0.7, st.x);
  float amplitude_strength = 0.5;
  float finalAmplitude = amplitude_normal * amplitude_strength
                  * amplitude * (1.0 + (mouse.y - 0.5) * 0.2);

  float time_scaled = time / 10.0 + (mouse.x - 0.5) * 1.0;
  float blur = smoothstep(split_point, split_point + 0.05, st.x) * perc;

  float xnoise = mix(
    Perlin2D(vec2(time_scaled, st.x + perc) * 2.5),
    Perlin2D(vec2(time_scaled, st.x + time_scaled) * 3.5) / 1.5,
    st.x * 0.3
  );

  float y = 0.5 + (perc - 0.5) * distance + xnoise / 2.0 * finalAmplitude;

  float line_start = smoothstep(
    y + (width / 2.0) + (u_line_blur * pixel(1.0, iResolution.xy) * blur),
    y,
    st.y
  );

  float line_end = smoothstep(
    y,
    y - (width / 2.0) - (u_line_blur * pixel(1.0, iResolution.xy) * blur),
    st.y
  );

  return clamp(
    (line_start - line_end) * (1.0 - smoothstep(0.0, 1.0, pow(perc, 0.3))),
    0.0,
    1.0
  );
}
```

```
void mainImage(out vec4 fragColor, in vec2 fragCoord) {
    vec2 uv = fragCoord / iResolution.xy;

    float line_strength = 1.0;
    for (int i = 0; i < u_line_count; i++) {
        float p = float(i) / float(u_line_count);
        line_strength *= (1.0 - lineFn(
            uv,
            u_line_width * pixel(1.0, iResolution.xy) * (1.0 - p),
            p,
            (PI * 1.0) * p,
            uMouse,
            iTime,
            uAmplitude,
            uDistance
        ));
    }

    float colorVal = 1.0 - line_strength;
    fragColor = vec4(uColor * colorVal, colorVal);
}

void main() {
    mainImage(gl_FragColor, gl_FragCoord.xy);
}
`;

const Threads = ({ color = [1, 1, 1], amplitude = 1, distance = 0, enableMouseInteraction = false, ...rest })
=> {
  const containerRef = useRef(null);
  const animationFrameId = useRef();

  useEffect(() => {
    if (!containerRef.current) return;
    const container = containerRef.current;

    const renderer = new Renderer({ alpha: true });
    const gl = renderer.gl;
    gl.clearColor(0, 0, 0, 0);
    gl.enable(gl.BLEND);
    gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
    container.appendChild(gl.canvas);

    const geometry = new Triangle(gl);
    const program = new Program(gl, {
      vertex: vertexShader,
      fragment: fragmentShader,
      uniforms: {
        iTime: { value: 0 },
        iResolution: {
          value: new Color(gl.canvas.width, gl.canvas.height, gl.canvas.width / gl.canvas.height)
        },
        uColor: { value: new Color(...color) },
        uAmplitude: { value: amplitude },
        uDistance: { value: distance },
        uMouse: { value: new Float32Array([0.5, 0.5]) }
      }
    });

    const mesh = new Mesh(gl, { geometry, program });

    function resize() {
      const { clientWidth, clientHeight } = container;
      renderer.setSize(clientWidth, clientHeight);
      program.uniforms.iResolution.value.r = clientWidth;
      program.uniforms.iResolution.value.g = clientHeight;
      program.uniforms.iResolution.value.b = clientWidth / clientHeight;
    }
    window.addEventListener('resize', resize);
    resize();

    let currentMouse = [0.5, 0.5];
    let targetMouse = [0.5, 0.5];

    function handleMouseMove(e) {
      const rect = container.getBoundingClientRect();
      const x = (e.clientX - rect.left) / rect.width;
      const y = 1.0 - (e.clientY - rect.top) / rect.height;
      targetMouse = [x, y];
    }
    function handleMouseLeave() {
      targetMouse = [0.5, 0.5];
    }
    if (enableMouseInteraction) {
      container.addEventListener('mousemove', handleMouseMove);
      container.addEventListener('mouseleave', handleMouseLeave);
    }

    function update(t) {
      if (enableMouseInteraction) {
        const smoothing = 0.05;
        currentMouse[0] += smoothing * (targetMouse[0] - currentMouse[0]);
        currentMouse[1] += smoothing * (targetMouse[1] - currentMouse[1]);
        program.uniforms.uMouse.value[0] = currentMouse[0];
        program.uniforms.uMouse.value[1] = currentMouse[1];
```

```
      } else {
        program.uniforms.uMouse.value[0] = 0.5;
        program.uniforms.uMouse.value[1] = 0.5;
      }
      program.uniforms.iTime.value = t * 0.001;

      renderer.render({ scene: mesh });
      animationFrameId.current = requestAnimationFrame(update);
    }
    animationFrameId.current = requestAnimationFrame(update);

    return () => {
      if (animationFrameId.current) cancelAnimationFrame(animationFrameId.current);
      window.removeEventListener('resize', resize);

      if (enableMouseInteraction) {
        container.removeEventListener('mousemove', handleMouseMove);
        container.removeEventListener('mouseleave', handleMouseLeave);
      }
      if (container.contains(gl.canvas)) container.removeChild(gl.canvas);
      gl.getExtension('WEBGL_lose_context')?.loseContext();
    };
  }, [color, amplitude, distance, enableMouseInteraction]);

  return <div ref={containerRef} className="threads-container" {...rest} />;
};

export default Threads;
```

—

```
.threads-container {
  position: relative;
  width: 100%;
  height: 100%;
}
```

Glass Icons

Usage:

```
import GlassIcons from './GlassIcons'

// update with your own icons and colors
const items = [
  { icon: <FiFileText />, color: 'blue', label: 'Files' },
  { icon: <FiBook />, color: 'purple', label: 'Books' },
  { icon: <FiHeart />, color: 'red', label: 'Health' },
  { icon: <FiCloud />, color: 'indigo', label: 'Weather' },
  { icon: <FiEdit />, color: 'orange', label: 'Notes' },
  { icon: <FiBarChart2 />, color: 'green', label: 'Stats' },
];

<div style={{ height: '600px', position: 'relative' }}>
  <GlassIcons items={items} className="custom-class"/>
</div>
```

Code:

```
import './GlassIcons.css';

const gradientMapping = {
  blue: 'linear-gradient(hsl(223, 90%, 50%), hsl(208, 90%, 50%))',
  purple: 'linear-gradient(hsl(283, 90%, 50%), hsl(268, 90%, 50%))',
  red: 'linear-gradient(hsl(3, 90%, 50%), hsl(348, 90%, 50%))',
  indigo: 'linear-gradient(hsl(253, 90%, 50%), hsl(238, 90%, 50%))',
  orange: 'linear-gradient(hsl(43, 90%, 50%), hsl(28, 90%, 50%))',
  green: 'linear-gradient(hsl(123, 90%, 40%), hsl(108, 90%, 40%))'
};

const GlassIcons = ({ items, className }) => {
  const getBackgroundStyle = color => {
    if (gradientMapping[color]) {
      return { background: gradientMapping[color] };
    }
    return { background: color };
  };

  return (
    <div className={`icon-btns ${className || ''}`}>
      {items.map((item, index) => (
        <button key={index} className={`icon-btn ${item.customClass || ''}`} aria-label={item.label}
type="button">
          <span className="icon-btn__back" style={getBackgroundStyle(item.color)}></span>
          <span className="icon-btn__front">
            <span className="icon-btn__icon" aria-hidden="true">
              {item.icon}
            </span>
          </span>
          <span className="icon-btn__label">{item.label}</span>
        </button>
      ))}
    </div>
  );
};

export default GlassIcons;
```

CSS:

```
.icon-btns {
  display: grid;
  grid-gap: 5em;
  grid-template-columns: repeat(2, 1fr);
  margin: auto;
  padding: 3em 0;
  overflow: visible;
}

.icon-btn {
  background-color: transparent;
  outline: none;
  position: relative;
  width: 4.5em;
  height: 4.5em;
  perspective: 24em;
  transform-style: preserve-3d;
  -webkit-tap-highlight-color: transparent;
}

.icon-btn__back,
.icon-btn__front,
.icon-btn__label {
  transition:
    opacity 0.3s cubic-bezier(0.83, 0, 0.17, 1),
    transform 0.3s cubic-bezier(0.83, 0, 0.17, 1);
}

.icon-btn__back,
.icon-btn__front {
  border-radius: 1.25em;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}

.icon-btn__back {
  box-shadow: 0.5em -0.5em 0.75em hsla(223, 10%, 10%, 0.15);
  display: block;
  transform: rotate(15deg);
  transform-origin: 100% 100%;
}

.icon-btn__front {
  background-color: hsla(0, 0%, 100%, 0.15);
  box-shadow: 0 0 0 0.1em hsla(0, 0%, 100%, 0.3) inset;
  backdrop-filter: blur(0.75em);
  -webkit-backdrop-filter: blur(0.75em);
  display: flex;
  transform-origin: 80% 50%;
}

.icon-btn__icon {
  margin: auto;
  width: 1.5em;
  height: 1.5em;
  display: flex;
  align-items: center;
  justify-content: center;
}

.icon-btn__label {
  font-size: 1em;
  white-space: nowrap;
  text-align: center;
  line-height: 2;
  opacity: 0;
  position: absolute;
  top: 100%;
  right: 0;
  left: 0;
  transform: translateY(0);
}

.icon-btn:focus-visible .icon-btn__back,
.icon-btn:hover .icon-btn__back {
  transform: rotate(25deg) translate3d(-0.5em, -0.5em, 0.5em);
}

.icon-btn:focus-visible .icon-btn__front,
.icon-btn:hover .icon-btn__front {
  transform: translateZ(2em);
}

.icon-btn:focus-visible .icon-btn__label,
.icon-btn:hover .icon-btn__label {
  opacity: 1;
  transform: translateY(20%);
}

@media (min-width: 768px) {
  .icon-btns {
    grid-template-columns: repeat(3, 1fr);
  }
}
```

Titled Card:

Usage:

```
import TiltedCard from './TiltedCard';

<TiltedCard
  imageSrc="https://i.scdn.co/image/ab67616d0000b273d9985092cd88bffd97653b58"
  altText="Kendrick Lamar - GNX Album Cover"
  captionText="Kendrick Lamar - GNX"
  containerHeight="300px"
  containerWidth="300px"
  imageHeight="300px"
  imageWidth="300px"
  rotateAmplitude={12}
  scaleOnHover={1.2}
  showMobileWarning={false}
  showTooltip={true}
  displayOverlayContent={true}
  overlayContent={
    <p className="tilted-card-demo-text">
      Kendrick Lamar - GNX
    </p>
  }
/>
```

Code:

```
import { useRef, useState } from 'react';
import { motion, useMotionValue, useSpring } from 'motion/react';
import './TiltedCard.css';

const springValues = {
  damping: 30,
  stiffness: 100,
  mass: 2
};

export default function TiltedCard({
  imageSrc,
  altText = 'Tilted card image',
  captionText = '',
  containerHeight = '300px',
  containerWidth = '100%',
  imageHeight = '300px',
  imageWidth = '300px',
  scaleOnHover = 1.1,
  rotateAmplitude = 14,
  showMobileWarning = true,
  showTooltip = true,
  overlayContent = null,
  displayOverlayContent = false
}) {
  const ref = useRef(null);

  const x = useMotionValue();
  const y = useMotionValue();
  const rotateX = useSpring(useMotionValue(0), springValues);
  const rotateY = useSpring(useMotionValue(0), springValues);
  const scale = useSpring(1, springValues);
  const opacity = useSpring(0);
  const rotateFigcaption = useSpring(0, {
    stiffness: 350,
    damping: 30,
    mass: 1
  });

  const [lastY, setLastY] = useState(0);

  function handleMouse(e) {
    if (!ref.current) return;

    const rect = ref.current.getBoundingClientRect();
    const offsetX = e.clientX - rect.left - rect.width / 2;
    const offsetY = e.clientY - rect.top - rect.height / 2;

    const rotationX = (offsetY / (rect.height / 2)) * -rotateAmplitude;
    const rotationY = (offsetX / (rect.width / 2)) * rotateAmplitude;

    rotateX.set(rotationX);
    rotateY.set(rotationY);

    x.set(e.clientX - rect.left);
    y.set(e.clientY - rect.top);

    const velocityY = offsetY - lastY;
    rotateFigcaption.set(-velocityY * 0.6);
    setLastY(offsetY);
  }

  function handleMouseEnter() {
    scale.set(scaleOnHover);
    opacity.set(1);
  }

  function handleMouseLeave() {
    opacity.set(0);
    scale.set(1);
    rotateX.set(0);
    rotateY.set(0);
    rotateFigcaption.set(0);
  }

  return (
    <figure
      ref={ref}
      className="tilted-card-figure"
      style={{
        height: containerHeight,
        width: containerWidth
      }}
      onMouseMove={handleMouse}
      onMouseEnter={handleMouseEnter}
      onMouseLeave={handleMouseLeave}
    >
      {showMobileWarning && (
        <div className="tilted-card-mobile-alert">This effect is not optimized for mobile. Check on
desktop.</div>
      )}

      <motion.div
        className="tilted-card-inner"
        style={{
          width: imageWidth,
          height: imageHeight,
          rotateX,
          rotateY,
          scale
        }}
      >
        <motion.img
          src={imageSrc}
          alt={altText}
          className="tilted-card-img"
          style={{
            width: imageWidth,
            height: imageHeight
          }}
        />

        {displayOverlayContent && overlayContent && (
          <motion.div className="tilted-card-overlay">{overlayContent}</motion.div>
        )}
      </motion.div>

      {showTooltip && (
        <motion.figcaption
          className="tilted-card-caption"
          style={{
            x,
            y,
            opacity,
            rotate: rotateFigcaption
          }}
        >
          {captionText}
        </motion.figcaption>
      )}
    </figure>
  );
}
```

CSS:

```
.tilted-card-figure {
  position: relative;
  width: 100%;
  height: 100%;
  perspective: 800px;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

.tilted-card-mobile-alert {
  position: absolute;
  top: 1rem;
  text-align: center;
  font-size: 0.875rem;
  display: none;
}

@media (max-width: 640px) {
  .tilted-card-mobile-alert {
    display: block;
  }
  .tilted-card-caption {
    display: none;
  }
}

.tilted-card-inner {
  position: relative;
  transform-style: preserve-3d;
}

.tilted-card-img {
  position: absolute;
  top: 0;
  left: 0;
  object-fit: cover;
  border-radius: 15px;
  will-change: transform;
  transform: translateZ(0);
}

.tilted-card-overlay {
  position: absolute;
  top: 0;
  left: 0;
  z-index: 2;
  will-change: transform;
  transform: translateZ(30px);
}

.tilted-card-caption {
  pointer-events: none;
  position: absolute;
  left: 0;
  top: 0;
  border-radius: 4px;
  background-color: #fff;
  padding: 4px 10px;
  font-size: 10px;
  color: #2d2d2d;
  opacity: 0;
  z-index: 3;
}
```

Stack of images (to scroll through)

Usage:
```
import Stack from './Stack'

const images = [
  { id: 1, img:
"https://images.unsplash.com/photo-1480074568708-e7b720bb3f09?q=80&w=500&auto=format" },
  { id: 2, img:
"https://images.unsplash.com/photo-1449844908441-8829872d2607?q=80&w=500&auto=format" },
  { id: 3, img:
"https://images.unsplash.com/photo-1452626212852-811d58933cae?q=80&w=500&auto=format" },
  { id: 4, img:
"https://images.unsplash.com/photo-1572120360610-d971b9d7767c?q=80&w=500&auto=format" }
];

<Stack
  randomRotation={true}
  sensitivity={180}
  sendToBackOnClick={false}
  cardDimensions={{ width: 200, height: 200 }}
  cardsData={images}
/>
```

Code:
```
import { motion, useMotionValue, useTransform } from 'motion/react';
import { useState } from 'react';
import './Stack.css';

function CardRotate({ children, onSendToBack, sensitivity }) {
  const x = useMotionValue(0);
  const y = useMotionValue(0);
  const rotateX = useTransform(y, [-100, 100], [60, -60]);
  const rotateY = useTransform(x, [-100, 100], [-60, 60]);

  function handleDragEnd(_, info) {
    if (Math.abs(info.offset.x) > sensitivity || Math.abs(info.offset.y) > sensitivity) {
      onSendToBack();
    } else {
      x.set(0);
      y.set(0);
    }
  }

  return (
    <motion.div
      className="card-rotate"
      style={{ x, y, rotateX, rotateY }}
      drag
      dragConstraints={{ top: 0, right: 0, bottom: 0, left: 0 }}
      dragElastic={0.6}
      whileTap={{ cursor: 'grabbing' }}
      onDragEnd={handleDragEnd}
    >
      {children}
    </motion.div>
  );
}

export default function Stack({
  randomRotation = false,
  sensitivity = 200,
  cardDimensions = { width: 208, height: 208 },
  cardsData = [],
  animationConfig = { stiffness: 260, damping: 20 },
  sendToBackOnClick = false
}) {
  const [cards, setCards] = useState(
    cardsData.length
      ? cardsData
      : [
          { id: 1, img:
'https://images.unsplash.com/photo-1480074568708-e7b720bb3f09?q=80&w=500&auto=format' },
          { id: 2, img:
'https://images.unsplash.com/photo-1449844908441-8829872d2607?q=80&w=500&auto=format' },
          { id: 3, img:
'https://images.unsplash.com/photo-1452626212852-811d58933cae?q=80&w=500&auto=format' },
          { id: 4, img:
'https://images.unsplash.com/photo-1572120360610-d971b9d7767c?q=80&w=500&auto=format' }
        ]
  );

  const sendToBack = id => {
    setCards(prev => {
      const newCards = [...prev];
      const index = newCards.findIndex(card => card.id === id);
      const [card] = newCards.splice(index, 1);
      newCards.unshift(card);
      return newCards;
    });
  };

  return (
    <div
      className="stack-container"
      style={{
        width: cardDimensions.width,
        height: cardDimensions.height,
        perspective: 600
      }}
    >
      {cards.map((card, index) => {
        const randomRotate = randomRotation ? Math.random() * 10 - 5 : 0;

        return (
          <CardRotate key={card.id} onSendToBack={() => sendToBack(card.id)} sensitivity={sensitivity}>
            <motion.div
              className="card"
              onClick={() => sendToBackOnClick && sendToBack(card.id)}
              animate={{
                rotateZ: (cards.length - index - 1) * 4 + randomRotate,
                scale: 1 + index * 0.06 - cards.length * 0.06,
                transformOrigin: '90% 90%'
              }}
              initial={false}
              transition={{
                type: 'spring',
                stiffness: animationConfig.stiffness,
                damping: animationConfig.damping
              }}
              style={{
                width: cardDimensions.width,
                height: cardDimensions.height
              }}
            >
              <img src={card.img} alt={`card-${card.id}`} className="card-image" />
            </motion.div>
          </CardRotate>
        );
      })}
    </div>
  );
}

CSS:
.stack-container {
  position: relative;
  perspective: 600px;
}

.card-rotate {
  position: absolute;
  cursor: grab;
}

.card {
  border-radius: 20px;
  border: 5px solid #fff;
  overflow: hidden;
}

.card-image {
  pointer-events: none;
  width: 100%;
  height: 100%;
  object-fit: cover;
}
```

Button Border effect
Usage:
```
import StarBorder from './StarBorder'

<StarBorder
  as="button"
  className="custom-class"
  color="cyan"
  speed="5s"
>
  // content
</StarBorder>
```

Code:
```
import './StarBorder.css';

const StarBorder = ({
  as: Component = 'button',
  className = '',
  color = 'white',
  speed = '6s',
  thickness = 1,
  children,
  ...rest
}) => {
  return (
    <Component
      className={`star-border-container ${className}`}
      style={{
        padding: `${thickness}px 0`,
        ...rest.style
      }}
      {...rest}
    >
      <div
        className="border-gradient-bottom"
        style={{
          background: `radial-gradient(circle, ${color}, transparent 10%)`,
          animationDuration: speed
        }}
      ></div>
      <div
        className="border-gradient-top"
        style={{
          background: `radial-gradient(circle, ${color}, transparent 10%)`,
          animationDuration: speed
        }}
      ></div>
      <div className="inner-content">{children}</div>
    </Component>
  );
};

export default StarBorder;
```

CSS:

```
.star-border-container {
  display: inline-block;
  position: relative;
  border-radius: 20px;
  overflow: hidden;
}

.border-gradient-bottom {
  position: absolute;
  width: 300%;
  height: 50%;
  opacity: 0.7;
  bottom: -12px;
  right: -250%;
  border-radius: 50%;
  animation: star-movement-bottom linear infinite alternate;
  z-index: 0;
}

.border-gradient-top {
  position: absolute;
  opacity: 0.7;
  width: 300%;
  height: 50%;
  top: -12px;
  left: -250%;
  border-radius: 50%;
  animation: star-movement-top linear infinite alternate;
  z-index: 0;
}

.inner-content {
  position: relative;
  border: 1px solid #222;
  background: #000;
  color: white;
  font-size: 16px;
  text-align: center;
  padding: 16px 26px;
  border-radius: 20px;
  z-index: 1;
}

@keyframes star-movement-bottom {
  0% {
    transform: translate(0%, 0%);
    opacity: 1;
  }
  100% {
    transform: translate(-100%, 0%);
    opacity: 0;
  }
}

@keyframes star-movement-top {
  0% {
    transform: translate(0%, 0%);
    opacity: 1;
  }
  100% {
    transform: translate(100%, 0%);
    opacity: 0;
  }
}
```

Metallic paint effect for Logo

Usage:

```
import MetallicPaint, { parseLogoImage } from "./MetallicPaint";
import { useState, useEffect } from 'react';

// replace with your own SVG
// NOTE: your SVG should have a bit of padding around the shape, to keep it from being cut off
// it should also have black fill color, to allow the metallic effect to show through the mask
import logo from '../../assets/logos/react-bits-logo-small-black.svg';
```

```
const Component = () => {
  const [imageData, setImageData] = useState<ImageData | null>(null);

  useEffect(() => {
    async function loadDefaultImage() {
      try {
        const response = await fetch(logo);
        const blob = await response.blob();
        const file = new File([blob], "default.png", { type: blob.type });

        const parsedData = await parseLogoImage(file);
        setImageData(parsedData?.imageData ?? null);

      } catch (err) {
        console.error("Error loading default image:", err);
      }
    }

    loadDefaultImage();
  }, []);

  return (
    <div style={{ width: '100%', height: '100vh' }}>
      <MetallicPaint
        imageData={imageData ?? new ImageData(1, 1)}
        params={{ edge: 2, patternBlur: 0.005, patternScale: 2, refraction: 0.015, speed: 0.3, liquid: 0.07 }}
      />
    </div>
  );
}
```

Code:

```
/* eslint-disable react-hooks/exhaustive-deps */
/* eslint-disable react-refresh/only-export-components */
'use client';

import { useEffect, useRef, useState } from 'react';
import './MetallicPaint.css';

const defaultParams = {
  patternScale: 2,
  refraction: 0.015,
  edge: 1,
  patternBlur: 0.005,
  liquid: 0.07,
  speed: 0.3
};
```

```
export function parseLogoImage(file) {
  const canvas = document.createElement('canvas');
  const ctx = canvas.getContext('2d');

  return new Promise((resolve, reject) => {
    if (!file || !ctx) {
      reject(new Error('Invalid file or context'));
      return;
    }

    const img = new Image();
    img.crossOrigin = 'anonymous';
    img.onload = function () {
      if (file.type === 'image/svg+xml') {
        img.width = 1000;
        img.height = 1000;
      }

      const MAX_SIZE = 1000;
      const MIN_SIZE = 500;
      let width = img.naturalWidth;
      let height = img.naturalHeight;

      if (width > MAX_SIZE || height > MAX_SIZE || width < MIN_SIZE || height < MIN_SIZE) {
        if (width > height) {
          if (width > MAX_SIZE) {
            height = Math.round((height * MAX_SIZE) / width);
            width = MAX_SIZE;
          } else if (width < MIN_SIZE) {
            height = Math.round((height * MIN_SIZE) / width);
            width = MIN_SIZE;
          }
        } else {
          if (height > MAX_SIZE) {
            width = Math.round((width * MAX_SIZE) / height);
            height = MAX_SIZE;
          } else if (height < MIN_SIZE) {
            width = Math.round((width * MIN_SIZE) / height);
            height = MIN_SIZE;
          }
        }
      }

      canvas.width = width;
      canvas.height = height;

      const shapeCanvas = document.createElement('canvas');
      shapeCanvas.width = width;
      shapeCanvas.height = height;
      const shapeCtx = shapeCanvas.getContext('2d');
      shapeCtx.drawImage(img, 0, 0, width, height);

      const shapeImageData = shapeCtx.getImageData(0, 0, width, height);
      const data = shapeImageData.data;
      const shapeMask = new Array(width * height).fill(false);
      for (let y = 0; y < height; y++) {
        for (let x = 0; x < width; x++) {
          const idx4 = (y * width + x) * 4;
          const r = data[idx4];
          const g = data[idx4 + 1];
          const b = data[idx4 + 2];
          const a = data[idx4 + 3];
          shapeMask[y * width + x] = !((r === 255 && g === 255 && b === 255 && a === 255) || a === 0);
        }
      }

      function inside(x, y) {
        if (x < 0 || x >= width || y < 0 || y >= height) return false;
        return shapeMask[y * width + x];
      }

      const boundaryMask = new Array(width * height).fill(false);
      for (let y = 0; y < height; y++) {
        for (let x = 0; x < width; x++) {
          const idx = y * width + x;
          if (!shapeMask[idx]) continue;
          let isBoundary = false;
          for (let ny = y - 1; ny <= y + 1 && !isBoundary; ny++) {
            for (let nx = x - 1; nx <= x + 1 && !isBoundary; nx++) {
              if (!inside(nx, ny)) {
                isBoundary = true;
              }
            }
          }
          if (isBoundary) {
            boundaryMask[idx] = true;
          }
        }
      }

      const interiorMask = new Array(width * height).fill(false);
      for (let y = 1; y < height - 1; y++) {
        for (let x = 1; x < width - 1; x++) {
          const idx = y * width + x;
          if (
            shapeMask[idx] &&
            shapeMask[idx - 1] &&
            shapeMask[idx + 1] &&
            shapeMask[idx - width] &&
            shapeMask[idx + width]
          ) {
            interiorMask[idx] = true;
          }
        }
      }

      const u = new Float32Array(width * height).fill(0);
      const newU = new Float32Array(width * height).fill(0);
      const C = 0.01;
      const ITERATIONS = 300;

      function getU(x, y, arr) {
        if (x < 0 || x >= width || y < 0 || y >= height) return 0;
        if (!shapeMask[y * width + x]) return 0;
        return arr[y * width + x];
      }

      for (let iter = 0; iter < ITERATIONS; iter++) {
        for (let y = 0; y < height; y++) {
          for (let x = 0; x < width; x++) {
            const idx = y * width + x;
            if (!shapeMask[idx] || boundaryMask[idx]) {
              newU[idx] = 0;
              continue;
            }
            const sumN = getU(x + 1, y, u) + getU(x - 1, y, u) + getU(x, y + 1, u) + getU(x, y - 1, u);
            newU[idx] = (C + sumN) / 4;
          }
        }
        u.set(newU);
      }

      let maxVal = 0;
      for (let i = 0; i < width * height; i++) {
        if (u[i] > maxVal) maxVal = u[i];
      }
      const alpha = 2.0;
      const outImg = ctx.createImageData(width, height);
      for (let y = 0; y < height; y++) {
        for (let x = 0; x < width; x++) {
          const idx = y * width + x;
          const px = idx * 4;
          if (!shapeMask[idx]) {
            outImg.data[px] = 255;
```

```
          outImg.data[px + 1] = 255;
          outImg.data[px + 2] = 255;
          outImg.data[px + 3] = 255;
        } else {
          const raw = u[idx] / maxVal;
          const remapped = Math.pow(raw, alpha);
          const gray = 255 * (1 - remapped);
          outImg.data[px] = gray;
          outImg.data[px + 1] = gray;
          outImg.data[px + 2] = gray;
          outImg.data[px + 3] = 255;
        }
      }
    }

    ctx.putImageData(outImg, 0, 0);

    canvas.toBlob(blob => {
      if (!blob) {
        reject(new Error('Failed to create PNG blob'));
        return;
      }
      resolve({
        imageData: outImg,
        pngBlob: blob
      });
    }, 'image/png');
  };

  img.onerror = () => reject(new Error('Failed to load image'));
  img.src = URL.createObjectURL(file);
 });
}

const vertexShaderSource = `#version 300 es
precision mediump float;

in vec2 a_position;
out vec2 vUv;

void main() {
  vUv = .5 * (a_position + 1.);
  gl_Position = vec4(a_position, 0.0, 1.0);
}`;

const liquidFragSource = `#version 300 es
precision mediump float;
```

```
in vec2 vUv;
out vec4 fragColor;

uniform sampler2D u_image_texture;
uniform float u_time;
uniform float u_ratio;
uniform float u_img_ratio;
uniform float u_patternScale;
uniform float u_refraction;
uniform float u_edge;
uniform float u_patternBlur;
uniform float u_liquid;

#define TWO_PI 6.28318530718
#define PI 3.14159265358979323846

vec3 mod289(vec3 x) { return x - floor(x * (1. / 289.)) * 289.; }
vec2 mod289(vec2 x) { return x - floor(x * (1. / 289.)) * 289.; }
vec3 permute(vec3 x) { return mod289(((x*34.)+1.)*x); }
float snoise(vec2 v) {
  const vec4 C = vec4(0.211324865405187, 0.366025403784439, -0.577350269189626,
0.024390243902439);
  vec2 i = floor(v + dot(v, C.yy));
  vec2 x0 = v - i + dot(i, C.xx);
  vec2 i1;
  i1 = (x0.x > x0.y) ? vec2(1., 0.) : vec2(0., 1.);
  vec4 x12 = x0.xyxy + C.xxzz;
  x12.xy -= i1;
  i = mod289(i);
  vec3 p = permute(permute(i.y + vec3(0., i1.y, 1.)) + i.x + vec3(0., i1.x, 1.));
  vec3 m = max(0.5 - vec3(dot(x0, x0), dot(x12.xy, x12.xy), dot(x12.zw, x12.zw)), 0.);
  m = m*m;
  m = m*m;
  vec3 x = 2. * fract(p * C.www) - 1.;
  vec3 h = abs(x) - 0.5;
  vec3 ox = floor(x + 0.5);
  vec3 a0 = x - ox;
  m *= 1.79284291400159 - 0.85373472095314 * (a0*a0 + h*h);
  vec3 g;
  g.x = a0.x * x0.x + h.x * x0.y;
  g.yz = a0.yz * x12.xz + h.yz * x12.yw;
  return 130. * dot(m, g);
}

vec2 get_img_uv() {
  vec2 img_uv = vUv;
  img_uv -= .5;
  if (u_ratio > u_img_ratio) {
      img_uv.x = img_uv.x * u_ratio / u_img_ratio;
  } else {
      img_uv.y = img_uv.y * u_img_ratio / u_ratio;
  }
  float scale_factor = 1.;
  img_uv *= scale_factor;
  img_uv += .5;
  img_uv.y = 1. - img_uv.y;
  return img_uv;
}
vec2 rotate(vec2 uv, float th) {
  return mat2(cos(th), sin(th), -sin(th), cos(th)) * uv;
}
float get_color_channel(float c1, float c2, float stripe_p, vec3 w, float extra_blur, float b) {
  float ch = c2;
  float border = 0.;
  float blur = u_patternBlur + extra_blur;
  ch = mix(ch, c1, smoothstep(.0, blur, stripe_p));
  border = w[0];
  ch = mix(ch, c2, smoothstep(border - blur, border + blur, stripe_p));
  b = smoothstep(.2, .8, b);
  border = w[0] + .4 * (1. - b) * w[1];
  ch = mix(ch, c1, smoothstep(border - blur, border + blur, stripe_p));
  border = w[0] + .5 * (1. - b) * w[1];
  ch = mix(ch, c2, smoothstep(border - blur, border + blur, stripe_p));
  border = w[0] + w[1];
  ch = mix(ch, c1, smoothstep(border - blur, border + blur, stripe_p));
  float gradient_t = (stripe_p - w[0] - w[1]) / w[2];
  float gradient = mix(c1, c2, smoothstep(0., 1., gradient_t));
  ch = mix(ch, gradient, smoothstep(border - blur, border + blur, stripe_p));
  return ch;
}
float get_img_frame_alpha(vec2 uv, float img_frame_width) {
  float img_frame_alpha = smoothstep(0., img_frame_width, uv.x) * smoothstep(1., 1. - img_frame_width,
uv.x);
  img_frame_alpha *= smoothstep(0., img_frame_width, uv.y) * smoothstep(1., 1. - img_frame_width,
uv.y);
  return img_frame_alpha;
}
void main() {
  vec2 uv = vUv;
  uv.y = 1. - uv.y;
  uv.x *= u_ratio;
  float diagonal = uv.x - uv.y;
  float t = .001 * u_time;
  vec2 img_uv = get_img_uv();
  vec4 img = texture(u_image_texture, img_uv);
  vec3 color = vec3(0.);
  float opacity = 1.;
  vec3 color1 = vec3(.98, 0.98, 1.);
  vec3 color2 = vec3(.1, .1, .1 + .1 * smoothstep(.7, 1.3, uv.x + uv.y));
  float edge = img.r;
  vec2 grad_uv = uv;
  grad_uv -= .5;
  float dist = length(grad_uv + vec2(0., .2 * diagonal));
  grad_uv = rotate(grad_uv, (.25 - .2 * diagonal) * PI);
  float bulge = pow(1.8 * dist, 1.2);
  bulge = 1. - bulge;
  bulge *= pow(uv.y, .3);
  float cycle_width = u_patternScale;
  float thin_strip_1_ratio = .12 / cycle_width * (1. - .4 * bulge);
  float thin_strip_2_ratio = .07 / cycle_width * (1. + .4 * bulge);
  float wide_strip_ratio = (1. - thin_strip_1_ratio - thin_strip_2_ratio);
  float thin_strip_1_width = cycle_width * thin_strip_1_ratio;
  float thin_strip_2_width = cycle_width * thin_strip_2_ratio;
  opacity = 1. - smoothstep(.9 - .5 * u_edge, 1. - .5 * u_edge, edge);
  opacity *= get_img_frame_alpha(img_uv, 0.01);
  float noise = snoise(uv - t);
  edge += (1. - edge) * u_liquid * noise;
  float refr = 0.;
  refr += (1. - bulge);
  refr = clamp(refr, 0., 1.);
  float dir = grad_uv.x;
  dir += diagonal;
  dir -= 2. * noise * diagonal * (smoothstep(0., 1., edge) * smoothstep(1., 0., edge));
  bulge *= clamp(pow(uv.y, .1), .3, 1.);
  dir *= (.1 + (1.1 - edge) * bulge);
  dir *= smoothstep(1., .7, edge);
  dir += .18 * (smoothstep(.1, .2, uv.y) * smoothstep(.4, .2, uv.y));
  dir += .03 * (smoothstep(.1, .2, 1. - uv.y) * smoothstep(.4, .2, 1. - uv.y));
  dir *= (.5 + .5 * pow(uv.y, 2.));
  dir *= cycle_width;
  dir -= t;
  float refr_r = refr;
  refr_r += .03 * bulge * noise;
  float refr_b = 1.3 * refr;
  refr_r += 5. * (smoothstep(-.1, .2, uv.y) * smoothstep(.5, .1, uv.y)) * (smoothstep(.4, .6, bulge) *
smoothstep(1., .4, bulge));
  refr_r -= diagonal;
  refr_b += (smoothstep(0., .4, uv.y) * smoothstep(.8, .1, uv.y)) * (smoothstep(.4, .6, bulge) *
smoothstep(.8, .4, bulge));
  refr_b -= .2 * edge;
  refr_r *= u_refraction;
  refr_b *= u_refraction;
  vec3 w = vec3(thin_strip_1_width, thin_strip_2_width, wide_strip_ratio);
  w[1] -= .02 * smoothstep(.0, 1., edge + bulge);
  float stripe_r = mod(dir + refr_r, 1.);
  float r = get_color_channel(color1.r, color2.r, stripe_r, w, 0.02 + .03 * u_refraction * bulge, bulge);
  float stripe_g = mod(dir, 1.);
  float g = get_color_channel(color1.g, color2.g, stripe_g, w, 0.01 / (1. - diagonal), bulge);
  float stripe_b = mod(dir - refr_b, 1.);
  float b = get_color_channel(color1.b, color2.b, stripe_b, w, .01, bulge);
  color = vec3(r, g, b);
  color *= opacity;
  fragColor = vec4(color, opacity);
}
`;

export default function MetallicPaint({ imageData, params = defaultParams }) {
  const canvasRef = useRef(null);
  const [gl, setGl] = useState(null);
  const [uniforms, setUniforms] = useState({});
  const totalAnimationTime = useRef(0);
  const lastRenderTime = useRef(0);

  function updateUniforms() {
  if (!gl || !uniforms) return;
  gl.uniform1f(uniforms.u_edge, params.edge);
  gl.uniform1f(uniforms.u_patternBlur, params.patternBlur);
  gl.uniform1f(uniforms.u_time, 0);
  gl.uniform1f(uniforms.u_patternScale, params.patternScale);
  gl.uniform1f(uniforms.u_refraction, params.refraction);
  gl.uniform1f(uniforms.u_liquid, params.liquid);
  }

  useEffect(() => {
  function initShader() {
    const canvas = canvasRef.current;
    const gl = canvas?.getContext('webgl2', {
      antialias: true,
      alpha: true
    });
    if (!canvas || !gl) {
      return;
    }

    function createShader(gl, sourceCode, type) {
      const shader = gl.createShader(type);
      if (!shader) {
        return null;
      }

      gl.shaderSource(shader, sourceCode);
      gl.compileShader(shader);
      if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('An error occurred compiling the shaders: ' + gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
      }

      return shader;
    }

    const vertexShader = createShader(gl, vertexShaderSource, gl.VERTEX_SHADER);
    const fragmentShader = createShader(gl, liquidFragSource, gl.FRAGMENT_SHADER);
    const program = gl.createProgram();
    if (!program || !vertexShader || !fragmentShader) {
      return;
    }

    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);

    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
      console.error('Unable to initialize the shader program: ' + gl.getProgramInfoLog(program));
      return null;
    }

    function getUniforms(program, gl) {
      let uniforms = {};
      let uniformCount = gl.getProgramParameter(program, gl.ACTIVE_UNIFORMS);
      for (let i = 0; i < uniformCount; i++) {
        let uniformName = gl.getActiveUniform(program, i)?.name;
        if (!uniformName) continue;
        uniforms[uniformName] = gl.getUniformLocation(program, uniformName);
      }
      return uniforms;
    }
    const uniforms = getUniforms(program, gl);
    setUniforms(uniforms);

    const vertices = new Float32Array([-1, -1, 1, -1, -1, 1, 1, 1]);
    const vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    gl.useProgram(program);

    const positionLocation = gl.getAttribLocation(program, 'a_position');
    gl.enableVertexAttribArray(positionLocation);
```

```
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);

    setGl(gl);
  }

  initShader();
  updateUniforms();
}, []);

useEffect(() => {
  if (!gl || !uniforms) return;
  updateUniforms();
}, [gl, params, uniforms]);

useEffect(() => {
  if (!gl || !uniforms) return;

  let renderId;

  function render(currentTime) {
    const deltaTime = currentTime - lastRenderTime.current;
    lastRenderTime.current = currentTime;

    totalAnimationTime.current += deltaTime * params.speed;
    gl.uniform1f(uniforms.u_time, totalAnimationTime.current);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    renderId = requestAnimationFrame(render);
  }

  lastRenderTime.current = performance.now();
  renderId = requestAnimationFrame(render);

  return () => {
    cancelAnimationFrame(renderId);
  };
}, [gl, params.speed]);

useEffect(() => {
  const canvasEl = canvasRef.current;
  if (!canvasEl || !gl || !uniforms) return;

  function resizeCanvas() {
    if (!canvasEl || !gl || !uniforms || !imageData) return;
    const imgRatio = imageData.width / imageData.height;
    gl.uniform1f(uniforms.u_img_ratio, imgRatio);
    const side = 1000;
    canvasEl.width = side * devicePixelRatio;
    canvasEl.height = side * devicePixelRatio;
    gl.viewport(0, 0, canvasEl.height, canvasEl.height);
    gl.uniform1f(uniforms.u_ratio, 1);
    gl.uniform1f(uniforms.u_img_ratio, imgRatio);
  }

  resizeCanvas();
  window.addEventListener('resize', resizeCanvas);

  return () => {
    window.removeEventListener('resize', resizeCanvas);
  };
}, [gl, uniforms, imageData]);

useEffect(() => {
  if (!gl || !uniforms) return;

  const existingTexture = gl.getParameter(gl.TEXTURE_BINDING_2D);
  if (existingTexture) {
    gl.deleteTexture(existingTexture);
  }

  const imageTexture = gl.createTexture();
  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, imageTexture);

  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

  gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);

  try {
    gl.texImage2D(
      gl.TEXTURE_2D,
      0,
      gl.RGBA,
      imageData?.width,
      imageData?.height,
      0,
      gl.RGBA,
      gl.UNSIGNED_BYTE,
      imageData?.data
    );
```

```
      gl.uniform1i(uniforms.u_image_texture, 0);
    } catch (e) {
      console.error('Error uploading texture:', e);
    }

    return () => {
      if (imageTexture) {
        gl.deleteTexture(imageTexture);
      }
    };
  }, [gl, uniforms, imageData]);

  return <canvas ref={canvasRef} className="paint-container" />;
}
```

CSS:

```
.paint-container {
  display: block;
  height: 100%;
  width: 100%;
  object-fit: contain;
}
```

Gradual blur when scrolling

Usage:

```
// Component added by Ansh - github.com/ansh-dhanani

import GradualBlur from './GradualBlur';

<section style={{position: 'relative',height: 500,overflow: 'hidden'}}>
  <div style={{ height: '100%',overflowY: 'auto',padding: '6rem 2rem' }}>
    <!-- Content Here - such as an image or text -->
  </div>

  <GradualBlur
    target="parent"
    position="bottom"
    height="6rem"
    strength={2}
    divCount={5}
    curve="bezier"
    exponential={true}
    opacity={1}
  />
</section>
```

Code:

```
import React, { useEffect, useRef, useState, useMemo } from 'react';
import * as math from 'mathjs';

import './GradualBlur.css';

const DEFAULT_CONFIG = {
  position: 'bottom',
  strength: 2,
  height: '6rem',
  divCount: 5,
  exponential: false,
  zIndex: 1000,
  animated: false,
  duration: '0.3s',
  easing: 'ease-out',
  opacity: 1,
  curve: 'linear',
  responsive: false,
  target: 'parent',
  className: '',
  style: {}
};

const PRESETS = {
  top: { position: 'top', height: '6rem' },
  bottom: { position: 'bottom', height: '6rem' },
  left: { position: 'left', height: '6rem' },
  right: { position: 'right', height: '6rem' },
  subtle: { height: '4rem', strength: 1, opacity: 0.8, divCount: 3 },
  intense: { height: '10rem', strength: 4, divCount: 8, exponential: true },
  smooth: { height: '8rem', curve: 'bezier', divCount: 10 },
  sharp: { height: '5rem', curve: 'linear', divCount: 4 },
  header: { position: 'top', height: '8rem', curve: 'ease-out' },
  footer: { position: 'bottom', height: '8rem', curve: 'ease-out' },
  sidebar: { position: 'left', height: '6rem', strength: 2.5 },
  'page-header': { position: 'top', height: '10rem', target: 'page', strength: 3 },
  'page-footer': { position: 'bottom', height: '10rem', target: 'page', strength: 3 }
};

const CURVE_FUNCTIONS = {
  linear: p => p,
  bezier: p => p * p * (3 - 2 * p),
  'ease-in': p => p * p,
  'ease-out': p => 1 - Math.pow(1 - p, 2),
  'ease-in-out': p => (p < 0.5 ? 2 * p * p : 1 - Math.pow(-2 * p + 2, 2) / 2)
};

const mergeConfigs = (...configs) => configs.reduce((acc, c) => ({ ...acc, ...c }), {});
const getGradientDirection = position =>
```

```
  ({
    top: 'to top',
    bottom: 'to bottom',
    left: 'to left',
    right: 'to right'
  })[position] || 'to bottom';

const debounce = (fn, wait) => {
  let t;
  return (...a) => {
    clearTimeout(t);
    t = setTimeout(() => fn(...a), wait);
  };
};

const useResponsiveDimension = (responsive, config, key) => {
  const [value, setValue] = useState(config[key]);
  useEffect(() => {
    if (!responsive) return;
    const calc = () => {
      const w = window.innerWidth;
      let v = config[key];
      if (w <= 480 && config[`mobile${key[0].toUpperCase() + key.slice(1)}`])
        v = config[`mobile${key[0].toUpperCase() + key.slice(1)}`];
      else if (w <= 768 && config[`tablet${key[0].toUpperCase() + key.slice(1)}`])
        v = config[`tablet${key[0].toUpperCase() + key.slice(1)}`];
      else if (w <= 1024 && config[`desktop${key[0].toUpperCase() + key.slice(1)}`])
        v = config[`desktop${key[0].toUpperCase() + key.slice(1)}`];
      setValue(v);
    };
    const debounced = debounce(calc, 100);
    calc();
    window.addEventListener('resize', debounced);
    return () => window.removeEventListener('resize', debounced);
  }, [responsive, config, key]);
  return responsive ? value : config[key];
};

const useIntersectionObserver = (ref, shouldObserve = false) => {
  const [isVisible, setIsVisible] = useState(!shouldObserve);

  useEffect(() => {
    if (!shouldObserve || !ref.current) return;

    const observer = new IntersectionObserver(([entry]) => setIsVisible(entry.isIntersecting), { threshold: 0.1
});

    observer.observe(ref.current);
    return () => observer.disconnect();
  }, [ref, shouldObserve]);

  return isVisible;
};

function GradualBlur(props) {
  const containerRef = useRef(null);
  const [isHovered, setIsHovered] = useState(false);

  const config = useMemo(() => {
    const presetConfig = props.preset && PRESETS[props.preset] ? PRESETS[props.preset] : {};
    return mergeConfigs(DEFAULT_CONFIG, presetConfig, props);
  }, [props]);

  const responsiveHeight = useResponsiveDimension(config.responsive, config, 'height');
  const responsiveWidth = useResponsiveDimension(config.responsive, config, 'width');

  const isVisible = useIntersectionObserver(containerRef, config.animated === 'scroll');

  const blurDivs = useMemo(() => {
    const divs = [];
    const increment = 100 / config.divCount;
    const currentStrength =
      isHovered && config.hoverIntensity ? config.strength * config.hoverIntensity : config.strength;

    const curveFunc = CURVE_FUNCTIONS[config.curve] || CURVE_FUNCTIONS.linear;

    for (let i = 1; i <= config.divCount; i++) {
      let progress = i / config.divCount;
      progress = curveFunc(progress);

      let blurValue;
      if (config.exponential) {
        blurValue = math.pow(2, progress * 4) * 0.0625 * currentStrength;
      } else {
        blurValue = 0.0625 * (progress * config.divCount + 1) * currentStrength;
      }

      const p1 = math.round((increment * i - increment) * 10) / 10;
      const p2 = math.round(increment * i * 10) / 10;
      const p3 = math.round((increment * i + increment) * 10) / 10;
      const p4 = math.round((increment * i + increment * 2) * 10) / 10;

      let gradient = `transparent ${p1}%, black ${p2}%`;
      if (p3 <= 100) gradient += `, black ${p3}%`;
      if (p4 <= 100) gradient += `, transparent ${p4}%`;
      const direction = getGradientDirection(config.position);

      const divStyle = {
        position: 'absolute',
        inset: '0',
        maskImage: `linear-gradient(${direction}, ${gradient})`,
        WebkitMaskImage: `linear-gradient(${direction}, ${gradient})`,
        backdropFilter: `blur(${blurValue.toFixed(3)}rem)`,
        WebkitBackdropFilter: `blur(${blurValue.toFixed(3)}rem)`,
        opacity: config.opacity,
        transition:
          config.animated && config.animated !== 'scroll'
            ? `backdrop-filter ${config.duration} ${config.easing}`
            : undefined
      };

      divs.push(<div key={i} style={divStyle} />);
    }

    return divs;
  }, [config, isHovered]);

  const containerStyle = useMemo(() => {
    const isVertical = ['top', 'bottom'].includes(config.position);
    const isHorizontal = ['left', 'right'].includes(config.position);
    const isPageTarget = config.target === 'page';

    const baseStyle = {
      position: isPageTarget ? 'fixed' : 'absolute',
      pointerEvents: config.hoverIntensity ? 'auto' : 'none',
      opacity: isVisible ? 1 : 0,
      transition: config.animated ? `opacity ${config.duration} ${config.easing}` : undefined,
      zIndex: isPageTarget ? config.zIndex + 100 : config.zIndex,
      ...config.style
    };

    if (isVertical) {
      baseStyle.height = responsiveHeight;
      baseStyle.width = responsiveWidth || '100%';
      baseStyle[config.position] = 0;
      baseStyle.left = 0;
      baseStyle.right = 0;
    } else if (isHorizontal) {
      baseStyle.width = responsiveWidth || responsiveHeight;
      baseStyle.height = '100%';
      baseStyle[config.position] = 0;
      baseStyle.top = 0;
      baseStyle.bottom = 0;
    }

    return baseStyle;
  }, [config, responsiveHeight, responsiveWidth, isVisible]);

  const { hoverIntensity, animated, onAnimationComplete, duration } = config;

  useEffect(() => {
    if (isVisible && animated === 'scroll' && onAnimationComplete) {
      const ms = parseFloat(duration) * 1000;
      const t = setTimeout(() => onAnimationComplete(), ms);
      return () => clearTimeout(t);
    }
  }, [isVisible, animated, onAnimationComplete, duration]);

  return (
    <div
      ref={containerRef}
      className={`gradual-blur ${config.target === 'page' ? 'gradual-blur-page' : 'gradual-blur-parent'}
${config.className}`}
      style={containerStyle}
      onMouseEnter={hoverIntensity ? () => setIsHovered(true) : undefined}
      onMouseLeave={hoverIntensity ? () => setIsHovered(false) : undefined}
    >
      <div
        className="gradual-blur-inner"
        style={{
          position: 'relative',
          width: '100%',
          height: '100%'
        }}
      >
        {blurDivs}
      </div>
    </div>
  );
}

const GradualBlurMemo = React.memo(GradualBlur);
GradualBlurMemo.displayName = 'GradualBlur';
GradualBlurMemo.PRESETS = PRESETS;
GradualBlurMemo.CURVE_FUNCTIONS = CURVE_FUNCTIONS;
export default GradualBlurMemo;

const injectStyles = () => {
  if (typeof document === 'undefined') return;

  const styleId = 'gradual-blur-styles';
  if (document.getElementById(styleId)) return;

  const styleElement = document.createElement('style');
  styleElement.id = styleId;
  styleElement.textContent = `
  .gradual-blur { pointer-events: none; transition: opacity 0.3s ease-out; }
  .gradual-blur-parent { overflow: hidden; }
  .gradual-blur-inner { pointer-events: none; }`;

  document.head.appendChild(styleElement);
};

if (typeof document !== 'undefined') {
  injectStyles();
}
```

CSS:

```
.gradual-blur-inner {
  position: relative;
  width: 100%;
  height: 100%;
}

.gradual-blur-inner > div {
  -webkit-backdrop-filter: inherit;
  backdrop-filter: inherit;
}

.gradual-blur {
  isolation: isolate;
}

@supports not (backdrop-filter: blur(1px)) {
  .gradual-blur-inner > div {
    background: rgba(0, 0, 0, 0.3);
    opacity: 0.5;
  }
}

.gradual-blur-fixed {
  position: fixed !important;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  pointer-events: none;
  z-index: 1000;
}
```

Glare Hover effect
Usage:
```
import GlareHover from './GlareHover'

<div style={{ height: '600px', position: 'relative' }}>
  <GlareHover
    glareColor="#ffffff"
    glareOpacity={0.3}
    glareAngle={-30}
    glareSize={300}
    transitionDuration={800}
    playOnce={false}
  >
    <h2 style={{ fontSize: '3rem', fontWeight: '900', color: '#333', margin: 0 }}>
      Hover Me
    </h2>
  </GlareHover>
</div>
```

Code:
```
import './GlareHover.css';

const GlareHover = ({
  width = '500px',
  height = '500px',
  background = '#000',
  borderRadius = '10px',
  borderColor = '#333',
  children,
  glareColor = '#ffffff',
  glareOpacity = 0.5,
  glareAngle = -45,
  glareSize = 250,
  transitionDuration = 650,
  playOnce = false,
  className = '',
  style = {}
}) => {
  const hex = glareColor.replace('#', '');
  let rgba = glareColor;
  if (/^[0-9A-Fa-f]{6}$/.test(hex)) {
    const r = parseInt(hex.slice(0, 2), 16);
    const g = parseInt(hex.slice(2, 4), 16);
    const b = parseInt(hex.slice(4, 6), 16);
    rgba = `rgba(${r}, ${g}, ${b}, ${glareOpacity})`;
  } else if (/^[0-9A-Fa-f]{3}$/.test(hex)) {
    const r = parseInt(hex[0] + hex[0], 16);
    const g = parseInt(hex[1] + hex[1], 16);
    const b = parseInt(hex[2] + hex[2], 16);
    rgba = `rgba(${r}, ${g}, ${b}, ${glareOpacity})`;
  }

  const vars = {
    '--gh-width': width,
    '--gh-height': height,
    '--gh-bg': background,
    '--gh-br': borderRadius,
    '--gh-angle': `${glareAngle}deg`,
    '--gh-duration': `${transitionDuration}ms`,
    '--gh-size': `${glareSize}%`,
    '--gh-rgba': rgba,
    '--gh-border': borderColor
  };

  return (
    <div
      className={`glare-hover ${playOnce ? 'glare-hover--play-once' : ''} ${className}`}
      style={{ ...vars, ...style }}
    >
      {children}
    </div>
  );
};

export default GlareHover;
```

CSS:

```
.glare-hover {
  width: var(--gh-width);
  height: var(--gh-height);
  background: var(--gh-bg);
  border-radius: var(--gh-br);
  border: 1px solid var(--gh-border);
  overflow: hidden;
  position: relative;
  display: grid;
  place-items: center;
}

.glare-hover::before {
  content: '';
  position: absolute;
  inset: 0;
  background: linear-gradient(
    var(--gh-angle),
    hsla(0, 0%, 0%, 0) 60%,
    var(--gh-rgba) 70%,
    hsla(0, 0%, 0%, 0),
    hsla(0, 0%, 0%, 0) 100%
  );
  transition: var(--gh-duration) ease;
  background-size:
    var(--gh-size) var(--gh-size),
    100% 100%;
  background-repeat: no-repeat;
  background-position:
    -100% -100%,
    0 0;
}

.glare-hover:hover {
  cursor: pointer;
}

.glare-hover:hover::before {
  background-position:
    100% 100%,
    0 0;
}

.glare-hover--play-once::before {
  transition: none;
}

.glare-hover--play-once:hover::before {
  transition: var(--gh-duration) ease;
  background-position:
    100% 100%,
    0 0;
}
```

Shiny Text
Usage:

```
import ShinyText from './ShinyText';

<ShinyText
  text="Just some shiny text!"
  disabled={false}
  speed={3}
  className='custom-class'
/>
```

Code:
```
import './ShinyText.css';

const ShinyText = ({ text, disabled = false, speed = 5, className = '' }) => {
  const animationDuration = `${speed}s`;

  return (
    <div className={`shiny-text ${disabled ? 'disabled' : ''} ${className}`} style={{ animationDuration }}>
      {text}
    </div>
  );
};

export default ShinyText;
```

CSS:
```
.shiny-text {
  color: #b5b5b5a4; /* Adjust this color to change intensity/style */
  background: linear-gradient(
    120deg,
    rgba(255, 255, 255, 0) 40%,
    rgba(255, 255, 255, 0.8) 50%,
    rgba(255, 255, 255, 0) 60%
  );
  background-size: 200% 100%;
  -webkit-background-clip: text;
  background-clip: text;
  display: inline-block;
  animation: shine 5s linear infinite;
}

@keyframes shine {
  0% {
    background-position: 100%;
  }
  100% {
    background-position: -100%;
  }
}

.shiny-text.disabled {
  animation: none;
}
```

Fade Content

Usage:

```
import FadeContent from './FadeContent'

<FadeContent blur={true} duration={1000} easing="ease-out" initialOpacity={0}>
  {/* Anything placed inside this container will be fade into view */}
</FadeContent>
```

Code:

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

const FadeContent = ({
  children,
  blur = false,
  duration = 1000,
  easing = 'ease-out',
  delay = 0,
  threshold = 0.1,
  initialOpacity = 0,
  className = ''
}) => {
  const [inView, setInView] = useState(false);
  const ref = useRef(null);

  useEffect(() => {
    if (!ref.current) return;

    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          observer.unobserve(ref.current);
          setTimeout(() => {
            setInView(true);
          }, delay);
        }
      },
      { threshold }
    );

    observer.observe(ref.current);

    return () => observer.disconnect();
  }, [threshold, delay]);

  return (
    <div
      ref={ref}
      className={className}
      style={{
        opacity: inView ? 1 : initialOpacity,
        transition: `opacity ${duration}ms ${easing}, filter ${duration}ms ${easing}`,
        filter: blur ? (inView ? 'blur(0px)' : 'blur(10px)') : 'none'
      }}
    >
      {children}
    </div>
  );
};

export default FadeContent;
```

Animated Content:
Usage:

```
import AnimatedContent from './AnimatedContent'

<AnimatedContent
  distance={150}
  direction="horizontal"
  reverse={false}
  duration={1.2}
  ease="bounce.out"
  initialOpacity={0.2}
  animateOpacity
  scale={1.1}
  threshold={0.2}
  delay={0.3}
>
  <div>Content to Animate</div>
</AnimatedContent>
```

Code:

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

gsap.registerPlugin(ScrollTrigger);

const AnimatedContent = ({
  children,
  distance = 100,
  direction = 'vertical',
  reverse = false,
  duration = 0.8,
  ease = 'power3.out',
  initialOpacity = 0,
  animateOpacity = true,
  scale = 1,
  threshold = 0.1,
  delay = 0,
  onComplete
}) => {
  const ref = useRef(null);

  useEffect(() => {
    const el = ref.current;
    if (!el) return;

    const axis = direction === 'horizontal' ? 'x' : 'y';
    const offset = reverse ? -distance : distance;
    const startPct = (1 - threshold) * 100;

    gsap.set(el, {
      [axis]: offset,
      scale,
      opacity: animateOpacity ? initialOpacity : 1
    });

    gsap.to(el, {
      [axis]: 0,
      scale: 1,
      opacity: 1,
      duration,
      ease,
      delay,
      onComplete,
      scrollTrigger: {
        trigger: el,
        start: `top ${startPct}%`,
        toggleActions: 'play none none none',
        once: true
      }
    });

    return () => {
      ScrollTrigger.getAll().forEach(t => t.kill());
      gsap.killTweensOf(el);
    };
  }, [
    distance,
    direction,
    reverse,
    duration,
    ease,
    initialOpacity,
    animateOpacity,
    scale,
    threshold,
    delay,
    onComplete
  ]);

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

export default AnimatedContent;
```

Shiny text:

Usage:

```
import ShinyText from './ShinyText';

<ShinyText
  text="Just some shiny text!"
  disabled={false}
  speed={3}
  className='custom-class'
/>
```

Code:

```
import './ShinyText.css';

const ShinyText = ({ text, disabled = false, speed = 5, className = '' }) => {
  const animationDuration = `${speed}s`;

  return (
    <div className={`shiny-text ${disabled ? 'disabled' : ''} ${className}`} style={{ animationDuration }}>
      {text}
    </div>
  );
};

export default ShinyText;
```

CSS:

```
.shiny-text {
  color: #b5b5b5a4; /* Adjust this color to change intensity/style */
  background: linear-gradient(
    120deg,
    rgba(255, 255, 255, 0) 40%,
    rgba(255, 255, 255, 0.8) 50%,
    rgba(255, 255, 255, 0) 60%
  );
  background-size: 200% 100%;
  -webkit-background-clip: text;
  background-clip: text;
  display: inline-block;
  animation: shine 5s linear infinite;
}

@keyframes shine {
  0% {
    background-position: 100%;
  }
  100% {
    background-position: -100%;
  }
}

.shiny-text.disabled {
  animation: none;
}
```

Text Typed out:

Usage:

```
import TextType from './TextType';

<TextType
  text={["Text typing effect", "for your websites", "Happy coding!"]}
  typingSpeed={75}
  pauseDuration={1500}
  showCursor={true}
  cursorCharacter="|"
/>
```

Code:

```
'use client';

import { useEffect, useRef, useState, createElement, useMemo, useCallback } from 'react';
import { gsap } from 'gsap';
import './TextType.css';

const TextType = ({
  text,
  as: Component = 'div',
  typingSpeed = 50,
  initialDelay = 0,
  pauseDuration = 2000,
  deletingSpeed = 30,
  loop = true,
  className = '',
  showCursor = true,
  hideCursorWhileTyping = false,
  cursorCharacter = '|',
  cursorClassName = '',
  cursorBlinkDuration = 0.5,
  textColors = [],
  variableSpeed,
  onSentenceComplete,
  startOnVisible = false,
  reverseMode = false,
  ...props
}) => {
  const [displayedText, setDisplayedText] = useState('');
  const [currentCharIndex, setCurrentCharIndex] = useState(0);
  const [isDeleting, setIsDeleting] = useState(false);
  const [currentTextIndex, setCurrentTextIndex] = useState(0);
  const [isVisible, setIsVisible] = useState(!startOnVisible);
  const cursorRef = useRef(null);
  const containerRef = useRef(null);

  const textArray = useMemo(() => (Array.isArray(text) ? text : [text]), [text]);

  const getRandomSpeed = useCallback(() => {
    if (!variableSpeed) return typingSpeed;
    const { min, max } = variableSpeed;
    return Math.random() * (max - min) + min;
  }, [variableSpeed, typingSpeed]);

  const getCurrentTextColor = () => {
    if (textColors.length === 0) return;
    return textColors[currentTextIndex % textColors.length];
  };

  useEffect(() => {
    if (!startOnVisible || !containerRef.current) return;

    const observer = new IntersectionObserver(
      entries => {
        entries.forEach(entry => {
          if (entry.isIntersecting) {
            setIsVisible(true);
          }
        });
      },
      { threshold: 0.1 }
    );
  observer.observe(containerRef.current);
  return () => observer.disconnect();
}, [startOnVisible]);

useEffect(() => {
  if (showCursor && cursorRef.current) {
    gsap.set(cursorRef.current, { opacity: 1 });
    gsap.to(cursorRef.current, {
      opacity: 0,
      duration: cursorBlinkDuration,
      repeat: -1,
      yoyo: true,
      ease: 'power2.inOut'
    });
  }
}, [showCursor, cursorBlinkDuration]);

useEffect(() => {
  if (!isVisible) return;

  let timeout;
  const currentText = textArray[currentTextIndex];
  const processedText = reverseMode ? currentText.split('').reverse().join('') : currentText;

  const executeTypingAnimation = () => {
    if (isDeleting) {
      if (displayedText === '') {
        setIsDeleting(false);
        if (currentTextIndex === textArray.length - 1 && !loop) {
          return;
        }

        if (onSentenceComplete) {
          onSentenceComplete(textArray[currentTextIndex], currentTextIndex);
        }

        setCurrentTextIndex(prev => (prev + 1) % textArray.length);
        setCurrentCharIndex(0);
        timeout = setTimeout(() => {}, pauseDuration);
      } else {
        timeout = setTimeout(() => {
          setDisplayedText(prev => prev.slice(0, -1));
        }, deletingSpeed);
      }
    } else {
      if (currentCharIndex < processedText.length) {
        timeout = setTimeout(
          () => {
            setDisplayedText(prev => prev + processedText[currentCharIndex]);
            setCurrentCharIndex(prev => prev + 1);
          },
          variableSpeed ? getRandomSpeed() : typingSpeed
        );
      } else if (textArray.length > 1) {
        timeout = setTimeout(() => {
          setIsDeleting(true);
        }, pauseDuration);
      }
    }
  };

  if (currentCharIndex === 0 && !isDeleting && displayedText === '') {
    timeout = setTimeout(executeTypingAnimation, initialDelay);
  } else {
    executeTypingAnimation();
  }

  return () => clearTimeout(timeout);
  // eslint-disable-next-line react-hooks/exhaustive-deps
}, [
  currentCharIndex,
  displayedText,
  isDeleting,
  typingSpeed,
  deletingSpeed,
  pauseDuration,
  textArray,
  currentTextIndex,
  loop,
  initialDelay,
  isVisible,
  reverseMode,
  variableSpeed,
  onSentenceComplete
]);

const shouldHideCursor =
  hideCursorWhileTyping && (currentCharIndex < textArray[currentTextIndex].length || isDeleting);

return createElement(
  Component,
  {
    ref: containerRef,
    className: `text-type ${className}`,
    ...props
    },
    <span className="text-type__content" style={{ color: getCurrentTextColor() || 'inherit' }}>
      {displayedText}
    </span>,
    showCursor && (
      <span
        ref={cursorRef}
        className={`text-type__cursor ${cursorClassName} ${shouldHideCursor ?
'text-type__cursor--hidden' : ''}`}
      >
        {cursorCharacter}
      </span>
    )
  );
};

export default TextType;
```

CSS:

```
.text-type {
  display: inline-block;
  white-space: pre-wrap;
}

.text-type__cursor {
  margin-left: 0.25rem;
  display: inline-block;
  opacity: 1;
}

.text-type__cursor--hidden {
  display: none;
}
```

Blurred in text:
Usage:

```
import BlurText from "./BlurText";

const handleAnimationComplete = () => {
  console.log('Animation completed!');
};

<BlurText
  text="Isn't this so cool?!"
  delay={150}
  animateBy="words"
  direction="top"
  onAnimationComplete={handleAnimationComplete}
  className="text-2xl mb-8"
/>
```

Code:

```
import { motion } from 'motion/react';
import { useEffect, useRef, useState, useMemo } from 'react';

const buildKeyframes = (from, steps) => {
  const keys = new Set([...Object.keys(from), ...steps.flatMap(s => Object.keys(s))]);

  const keyframes = {};
  keys.forEach(k => {
    keyframes[k] = [from[k], ...steps.map(s => s[k])];
  });
  return keyframes;
};

const BlurText = ({
  text = '',
  delay = 200,
  className = '',
  animateBy = 'words',
  direction = 'top',
  threshold = 0.1,
  rootMargin = '0px',
  animationFrom,
  animationTo,
  easing = t => t,
  onAnimationComplete,
  stepDuration = 0.35
}) => {
  const elements = animateBy === 'words' ? text.split(' ') : text.split('');
  const [inView, setInView] = useState(false);
  const ref = useRef(null);

  useEffect(() => {
    if (!ref.current) return;
    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          setInView(true);
          observer.unobserve(ref.current);
        }
      },
      { threshold, rootMargin }
    );
    observer.observe(ref.current);
    return () => observer.disconnect();
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [threshold, rootMargin]);

  const defaultFrom = useMemo(
    () =>
      direction === 'top' ? { filter: 'blur(10px)', opacity: 0, y: -50 } : { filter: 'blur(10px)', opacity: 0, y: 50 },
    [direction]
  );

  const defaultTo = useMemo(
    () => [
      {
        filter: 'blur(5px)',
        opacity: 0.5,
        y: direction === 'top' ? 5 : -5
      },
      { filter: 'blur(0px)', opacity: 1, y: 0 }
    ],
    [direction]
  );

  const fromSnapshot = animationFrom ?? defaultFrom;
  const toSnapshots = animationTo ?? defaultTo;

  const stepCount = toSnapshots.length + 1;
  const totalDuration = stepDuration * (stepCount - 1);
  const times = Array.from({ length: stepCount }, (_, i) => (stepCount === 1 ? 0 : i / (stepCount - 1)));

  return (
    <p ref={ref} className={className} style={{ display: 'flex', flexWrap: 'wrap' }}>
      {elements.map((segment, index) => {
        const animateKeyframes = buildKeyframes(fromSnapshot, toSnapshots);

        const spanTransition = {
          duration: totalDuration,
          times,
          delay: (index * delay) / 1000
        };
        spanTransition.ease = easing;

        return (
          <motion.span
            className="inline-block will-change-[transform,filter,opacity]"
            key={index}
            initial={fromSnapshot}
            animate={inView ? animateKeyframes : fromSnapshot}
            transition={spanTransition}
            onAnimationComplete={index === elements.length - 1 ? onAnimationComplete : undefined}
          >
            {segment === ' ' ? '\u00A0' : segment}
            {animateBy === 'words' && index < elements.length - 1 && '\u00A0'}
          </motion.span>
        );
      })}
    </p>
  );
};

export default BlurText;
```

Rotating text (maybe for Sleep “nality” “sona” “better” “decoded”)

Usage:

```
import RotatingText from './RotatingText'

<RotatingText
  texts={['React', 'Bits', 'Is', 'Cool!']}
  mainClassName="px-2 sm:px-2 md:px-3 bg-cyan-300 text-black overflow-hidden py-0.5 sm:py-1 md:py-2
justify-center rounded-lg"
  staggerFrom={"last"}
  initial={{ y: "100%" }}
  animate={{ y: 0 }}
  exit={{ y: "-120%" }}
  staggerDuration={0.025}
  splitLevelClassName="overflow-hidden pb-0.5 sm:pb-1 md:pb-1"
  transition={{ type: "spring", damping: 30, stiffness: 400 }}
  rotationInterval={2000}
/>
```

Code:

```
'use client';

import { forwardRef, useCallback, useEffect, useImperativeHandle, useMemo, useState } from 'react';
import { motion, AnimatePresence } from 'motion/react';

import './RotatingText.css';

function cn(...classes) {
  return classes.filter(Boolean).join(' ');
}

const RotatingText = forwardRef((props, ref) => {
  const {
    texts,
    transition = { type: 'spring', damping: 25, stiffness: 300 },
    initial = { y: '100%', opacity: 0 },
    animate = { y: 0, opacity: 1 },
  exit = { y: '-120%', opacity: 0 },
  animatePresenceMode = 'wait',
  animatePresenceInitial = false,
  rotationInterval = 2000,
  staggerDuration = 0,
  staggerFrom = 'first',
  loop = true,
  auto = true,
  splitBy = 'characters',
  onNext,
  mainClassName,
  splitLevelClassName,
  elementLevelClassName,
  ...rest
} = props;

const [currentTextIndex, setCurrentTextIndex] = useState(0);

const splitIntoCharacters = text => {
  if (typeof Intl !== 'undefined' && Intl.Segmenter) {
    const segmenter = new Intl.Segmenter('en', { granularity: 'grapheme' });
    return Array.from(segmenter.segment(text), segment => segment.segment);
  }
  return Array.from(text);
};

const elements = useMemo(() => {
  const currentText = texts[currentTextIndex];
  if (splitBy === 'characters') {
    const words = currentText.split(' ');
    return words.map((word, i) => ({
      characters: splitIntoCharacters(word),
      needsSpace: i !== words.length - 1
    }));
  }
  if (splitBy === 'words') {
    return currentText.split(' ').map((word, i, arr) => ({
      characters: [word],
      needsSpace: i !== arr.length - 1
    }));
  }
  if (splitBy === 'lines') {
    return currentText.split('\n').map((line, i, arr) => ({
      characters: [line],
      needsSpace: i !== arr.length - 1
    }));
  }
```

```
    return currentText.split(splitBy).map((part, i, arr) => ({
      characters: [part],
      needsSpace: i !== arr.length - 1
    }));
  }, [texts, currentTextIndex, splitBy]);

  const getStaggerDelay = useCallback(
    (index, totalChars) => {
      const total = totalChars;
      if (staggerFrom === 'first') return index * staggerDuration;
      if (staggerFrom === 'last') return (total - 1 - index) * staggerDuration;
      if (staggerFrom === 'center') {
        const center = Math.floor(total / 2);
        return Math.abs(center - index) * staggerDuration;
      }
      if (staggerFrom === 'random') {
        const randomIndex = Math.floor(Math.random() * total);
        return Math.abs(randomIndex - index) * staggerDuration;
      }
      return Math.abs(staggerFrom - index) * staggerDuration;
    },
    [staggerFrom, staggerDuration]
  );

  const handleIndexChange = useCallback(
    newIndex => {
      setCurrentTextIndex(newIndex);
      if (onNext) onNext(newIndex);
    },
    [onNext]
  );

  const next = useCallback(() => {
    const nextIndex = currentTextIndex === texts.length - 1 ? (loop ? 0 : currentTextIndex) :
currentTextIndex + 1;
    if (nextIndex !== currentTextIndex) {
      handleIndexChange(nextIndex);
    }
  }, [currentTextIndex, texts.length, loop, handleIndexChange]);

  const previous = useCallback(() => {
    const prevIndex = currentTextIndex === 0 ? (loop ? texts.length - 1 : currentTextIndex) :
currentTextIndex - 1;
    if (prevIndex !== currentTextIndex) {
      handleIndexChange(prevIndex);
    }
  }, [currentTextIndex, texts.length, loop, handleIndexChange]);
  const jumpTo = useCallback(
    index => {
      const validIndex = Math.max(0, Math.min(index, texts.length - 1));
      if (validIndex !== currentTextIndex) {
        handleIndexChange(validIndex);
      }
    },
    [texts.length, currentTextIndex, handleIndexChange]
  );

  const reset = useCallback(() => {
    if (currentTextIndex !== 0) {
      handleIndexChange(0);
    }
  }, [currentTextIndex, handleIndexChange]);

  useImperativeHandle(
    ref,
    () => ({
      next,
      previous,
      jumpTo,
      reset
    }),
    [next, previous, jumpTo, reset]
  );

  useEffect(() => {
    if (!auto) return;
    const intervalId = setInterval(next, rotationInterval);
    return () => clearInterval(intervalId);
  }, [next, rotationInterval, auto]);

  return (
    <motion.span className={cn('text-rotate', mainClassName)} {...rest} layout transition={transition}>
      <span className="text-rotate-sr-only">{texts[currentTextIndex]}</span>
      <AnimatePresence mode={animatePresenceMode} initial={animatePresenceInitial}>
        <motion.span
          key={currentTextIndex}
          className={cn(splitBy === 'lines' ? 'text-rotate-lines' : 'text-rotate')}
          layout
          aria-hidden="true"
        >
          {elements.map((wordObj, wordIndex, array) => {
            const previousCharsCount = array.slice(0, wordIndex).reduce((sum, word) => sum +
word.characters.length, 0);
            return (
              <span key={wordIndex} className={cn('text-rotate-word', splitLevelClassName)}>
```

```
                {wordObj.characters.map((char, charIndex) => (
                  <motion.span
                    key={charIndex}
                    initial={initial}
                    animate={animate}
                    exit={exit}
                    transition={{
                      ...transition,
                      delay: getStaggerDelay(
                        previousCharsCount + charIndex,
                        array.reduce((sum, word) => sum + word.characters.length, 0)
                      )
                    }}
                    className={cn('text-rotate-element', elementLevelClassName)}
                  >
                    {char}
                  </motion.span>
                ))}
                {wordObj.needsSpace && <span className="text-rotate-space"> </span>}
              </span>
            );
          })}
        </motion.span>
      </AnimatePresence>
    </motion.span>
  );
});

RotatingText.displayName = 'RotatingText';
export default RotatingText;
```

CSS:

```
.text-rotate {
  display: flex;
  flex-wrap: wrap;
  white-space: pre-wrap;
  position: relative;
}

.text-rotate-sr-only {
  position: absolute;
  width: 1px;
  height: 1px;
  padding: 0;
  margin: -1px;
  overflow: hidden;
  clip: rect(0, 0, 0, 0);
  white-space: nowrap;
  border: 0;
}

.text-rotate-word {
  display: inline-flex;
}

.text-rotate-lines {
  display: flex;
  flex-direction: column;
  width: 100%;
}

.text-rotate-element {
  display: inline-block;
}

.text-rotate-space {
  white-space: pre;
}
```