Install

```
npx shadcn@latest add https://reactbits.dev/r/ElectricBorder-TS-TW
```

Usage

```
// CREDIT
// Component inspired by @BalintFerenczy on X
// https://codepen.io/BalintFerenczy/pen/KwdoyEN

import ElectricBorder from './ElectricBorder'

<ElectricBorder
  color="#7df9ff"
  speed={1}
  chaos={0.5}
  thickness={2}
  style={{ borderRadius: 16 }}
>
  <div>
    <p style={{ margin: '6px 0 0', opacity: 0.8 }}>
      A glowing, animated border wrapper.
    </p>
  </div>
</ElectricBorder>
```

Code

```
import React, { CSSProperties, PropsWithChildren, useEffect, useId, useLayoutEffect,
useRef } from 'react';

type ElectricBorderProps = PropsWithChildren<{
  color?: string;
  speed?: number;
  chaos?: number;
  thickness?: number;
  className?: string;
  style?: CSSProperties;
}>;

function hexToRgba(hex: string, alpha = 1): string {
  if (!hex) return `rgba(0,0,0,${alpha})`;
```

```
  let h = hex.replace('#', '');
  if (h.length === 3) {
    h = h
      .split('')
      .map(c => c + c)
      .join('');
  }
  const int = parseInt(h, 16);
  const r = (int >> 16) & 255;
  const g = (int >> 8) & 255;
  const b = int & 255;
  return `rgba(${r}, ${g}, ${b}, ${alpha})`;
}

const ElectricBorder: React.FC<ElectricBorderProps> = ({
  children,
  color = '#5227FF',
  speed = 1,
  chaos = 1,
  thickness = 2,
  className,
  style
}) => {
  const rawId = useId().replace(/[:]/g, '');
  const filterId = `turbulent-displace-${rawId}`;
  const svgRef = useRef<SVGSVGElement | null>(null);
  const rootRef = useRef<HTMLDivElement | null>(null);
  const strokeRef = useRef<HTMLDivElement | null>(null);

  const updateAnim = () => {
    const svg = svgRef.current;
    const host = rootRef.current;
    if (!svg || !host) return;

    if (strokeRef.current) {
      strokeRef.current.style.filter = `url(#${filterId})`;
    }

    const width = Math.max(1, Math.round(host.clientWidth ||
host.getBoundingClientRect().width || 0));
    const height = Math.max(1, Math.round(host.clientHeight ||
host.getBoundingClientRect().height || 0));

    const dyAnims = Array.from(svg.querySelectorAll<SVGAnimateElement>('feOffset >
animate[attributeName="dy"]'));
    if (dyAnims.length >= 2) {
      dyAnims[0].setAttribute('values', `${height}; 0`);
      dyAnims[1].setAttribute('values', `0; -${height}`);
```

```
    }

    const dxAnims = Array.from(svg.querySelectorAll<SVGAnimateElement>('feOffset >
animate[attributeName="dx"]'));
    if (dxAnims.length >= 2) {
      dxAnims[0].setAttribute('values', `${width}; 0`);
      dxAnims[1].setAttribute('values', `0; -${width}`);
    }

    const baseDur = 6;
    const dur = Math.max(0.001, baseDur / (speed || 1));
    [...dyAnims, ...dxAnims].forEach(a => a.setAttribute('dur', `${dur}s`));

    const disp = svg.querySelector('feDisplacementMap');
    if (disp) disp.setAttribute('scale', String(30 * (chaos || 1)));

    const filterEl = svg.querySelector<SVGFilterElement>(`#${CSS.escape(filterId)}`);
    if (filterEl) {
      filterEl.setAttribute('x', '-200%');
      filterEl.setAttribute('y', '-200%');
      filterEl.setAttribute('width', '500%');
      filterEl.setAttribute('height', '500%');
    }

    requestAnimationFrame(() => {
      [...dyAnims, ...dxAnims].forEach((a: any) => {
        if (typeof a.beginElement === 'function') {
          try {
            a.beginElement();
          } catch {}
        }
      });
    });
  };

  useEffect(() => {
    updateAnim();
  }, [speed, chaos]);

  useLayoutEffect(() => {
    if (!rootRef.current) return;
    const ro = new ResizeObserver(() => updateAnim());
    ro.observe(rootRef.current);
    updateAnim();
    return () => ro.disconnect();
  }, []);

  const inheritRadius: CSSProperties = {
```

```
    borderRadius: style?.borderRadius ?? 'inherit'
  };

  const strokeStyle: CSSProperties = {
    ...inheritRadius,
    borderWidth: thickness,
    borderStyle: 'solid',
    borderColor: color
  };

  const glow1Style: CSSProperties = {
    ...inheritRadius,
    borderWidth: thickness,
    borderStyle: 'solid',
    borderColor: hexToRgba(color, 0.6),
    filter: `blur(${0.5 + thickness * 0.25}px)`,
    opacity: 0.5
  };

  const glow2Style: CSSProperties = {
    ...inheritRadius,
    borderWidth: thickness,
    borderStyle: 'solid',
    borderColor: color,
    filter: `blur(${2 + thickness * 0.5}px)`,
    opacity: 0.5
  };

  const bgGlowStyle: CSSProperties = {
    ...inheritRadius,
    transform: 'scale(1.08)',
    filter: 'blur(32px)',
    opacity: 0.3,
    zIndex: -1,
    background: `linear-gradient(-30deg, ${hexToRgba(color, 0.8)}, transparent, ${color})`
  };

  return (
    <div ref={rootRef} className={'relative isolate ' + (className ?? '')} style={style}>
      <svg
        ref={svgRef}
        className="fixed -left-[10000px] -top-[10000px] w-[10px] h-[10px] opacity-[0.001]
pointer-events-none"
        aria-hidden
        focusable="false"
      >
        <defs>
```

```
        <filter id={filterId} colorInterpolationFilters="sRGB" x="-20%" y="-20%" width="140%"
height="140%">
          <feTurbulence type="turbulence" baseFrequency="0.02" numOctaves="10"
result="noise1" seed="1" />
          <feOffset in="noise1" dx="0" dy="0" result="offsetNoise1">
            <animate attributeName="dy" values="700; 0" dur="6s" repeatCount="indefinite"
calcMode="linear" />
          </feOffset>

          <feTurbulence type="turbulence" baseFrequency="0.02" numOctaves="10"
result="noise2" seed="1" />
          <feOffset in="noise2" dx="0" dy="0" result="offsetNoise2">
            <animate attributeName="dy" values="0; -700" dur="6s" repeatCount="indefinite"
calcMode="linear" />
          </feOffset>

          <feTurbulence type="turbulence" baseFrequency="0.02" numOctaves="10"
result="noise1" seed="2" />
          <feOffset in="noise1" dx="0" dy="0" result="offsetNoise3">
            <animate attributeName="dx" values="490; 0" dur="6s" repeatCount="indefinite"
calcMode="linear" />
          </feOffset>

          <feTurbulence type="turbulence" baseFrequency="0.02" numOctaves="10"
result="noise2" seed="2" />
          <feOffset in="noise2" dx="0" dy="0" result="offsetNoise4">
            <animate attributeName="dx" values="0; -490" dur="6s" repeatCount="indefinite"
calcMode="linear" />
          </feOffset>

          <feComposite in="offsetNoise1" in2="offsetNoise2" result="part1" />
          <feComposite in="offsetNoise3" in2="offsetNoise4" result="part2" />
          <feBlend in="part1" in2="part2" mode="color-dodge" result="combinedNoise" />
          <feDisplacementMap
            in="SourceGraphic"
            in2="combinedNoise"
            scale="30"
            xChannelSelector="R"
            yChannelSelector="B"
          />
        </filter>
      </defs>
    </svg>

    <div className="absolute inset-0 pointer-events-none" style={inheritRadius}>
      <div ref={strokeRef} className="absolute inset-0 box-border" style={strokeStyle} />
      <div className="absolute inset-0 box-border" style={glow1Style} />
      <div className="absolute inset-0 box-border" style={glow2Style} />
```

```
        <div className="absolute inset-0" style={bgGlowStyle} />
      </div>

      <div className="relative" style={inheritRadius}>
        {children}
      </div>
    </div>
  );
};

export default ElectricBorder;
```