Install

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

Usage

```
import Particles from './Particles';

<div style={{ width: '100%', height: '600px', position: 'relative' }}>
  <Particles
    particleColors={['#ffffff', '#ffffff']}
    particleCount={200}
    particleSpread={10}
    speed={0.1}
    particleBaseSize={100}
    moveParticlesOnHover={true}
    alphaParticles={false}
    disableRotation={false}
  />
</div>
```

Code

```
import React, { useEffect, useRef } from 'react';
import { Renderer, Camera, Geometry, Program, Mesh } from 'ogl';

interface ParticlesProps {
  particleCount?: number;
  particleSpread?: number;
  speed?: number;
  particleColors?: string[];
  moveParticlesOnHover?: boolean;
  particleHoverFactor?: number;
  alphaParticles?: boolean;
  particleBaseSize?: number;
  sizeRandomness?: number;
  cameraDistance?: number;
  disableRotation?: boolean;
  className?: string;
}
```

```
const defaultColors: string[] = ['#ffffff', '#ffffff', '#ffffff'];

const hexToRgb = (hex: string): [number, number, number] => {
  hex = hex.replace(/^#/, '');
  if (hex.length === 3) {
    hex = hex
      .split('')
      .map(c => c + c)
      .join('');
  }
  const int = parseInt(hex, 16);
  const r = ((int >> 16) & 255) / 255;
  const g = ((int >> 8) & 255) / 255;
  const b = (int & 255) / 255;
  return [r, g, b];
};

const vertex = /* glsl */ `
  attribute vec3 position;
  attribute vec4 random;
  attribute vec3 color;

  uniform mat4 modelMatrix;
  uniform mat4 viewMatrix;
  uniform mat4 projectionMatrix;
  uniform float uTime;
  uniform float uSpread;
  uniform float uBaseSize;
  uniform float uSizeRandomness;

  varying vec4 vRandom;
  varying vec3 vColor;

  void main() {
    vRandom = random;
    vColor = color;

    vec3 pos = position * uSpread;
    pos.z *= 10.0;

    vec4 mPos = modelMatrix * vec4(pos, 1.0);
    float t = uTime;
    mPos.x += sin(t * random.z + 6.28 * random.w) * mix(0.1, 1.5, random.x);
    mPos.y += sin(t * random.y + 6.28 * random.x) * mix(0.1, 1.5, random.w);
    mPos.z += sin(t * random.w + 6.28 * random.y) * mix(0.1, 1.5, random.z);

    vec4 mvPos = viewMatrix * mPos;
```

```
    if (uSizeRandomness == 0.0) {
      gl_PointSize = uBaseSize;
    } else {
      gl_PointSize = (uBaseSize * (1.0 + uSizeRandomness * (random.x - 0.5))) /
length(mvPos.xyz);
    }

    gl_Position = projectionMatrix * mvPos;
    gl_Position = projectionMatrix * mvPos;
  }
`;

const fragment = /* glsl */ `
  precision highp float;

  uniform float uTime;
  uniform float uAlphaParticles;
  varying vec4 vRandom;
  varying vec3 vColor;

  void main() {
    vec2 uv = gl_PointCoord.xy;
    float d = length(uv - vec2(0.5));

    if(uAlphaParticles < 0.5) {
      if(d > 0.5) {
        discard;
      }
      gl_FragColor = vec4(vColor + 0.2 * sin(uv.yxx + uTime + vRandom.y * 6.28), 1.0);
    } else {
      float circle = smoothstep(0.5, 0.4, d) * 0.8;
      gl_FragColor = vec4(vColor + 0.2 * sin(uv.yxx + uTime + vRandom.y * 6.28), circle);
    }
  }
`;

const Particles: React.FC<ParticlesProps> = ({
  particleCount = 200,
  particleSpread = 10,
  speed = 0.1,
  particleColors,
  moveParticlesOnHover = false,
  particleHoverFactor = 1,
  alphaParticles = false,
  particleBaseSize = 100,
  sizeRandomness = 1,
  cameraDistance = 20,
  disableRotation = false,
```

```
  className
}) => {
  const containerRef = useRef<HTMLDivElement>(null);
  const mouseRef = useRef<{ x: number; y: number }>({ x: 0, y: 0 });

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

    const renderer = new Renderer({ depth: false, alpha: true });
    const gl = renderer.gl;
    container.appendChild(gl.canvas);
    gl.clearColor(0, 0, 0, 0);

    const camera = new Camera(gl, { fov: 15 });
    camera.position.set(0, 0, cameraDistance);

    const resize = () => {
      const width = container.clientWidth;
      const height = container.clientHeight;
      renderer.setSize(width, height);
      camera.perspective({ aspect: gl.canvas.width / gl.canvas.height });
    };
    window.addEventListener('resize', resize, false);
    resize();

    const handleMouseMove = (e: MouseEvent) => {
      const rect = container.getBoundingClientRect();
      const x = ((e.clientX - rect.left) / rect.width) * 2 - 1;
      const y = -(((e.clientY - rect.top) / rect.height) * 2 - 1);
      mouseRef.current = { x, y };
    };

    if (moveParticlesOnHover) {
      container.addEventListener('mousemove', handleMouseMove);
    }

    const count = particleCount;
    const positions = new Float32Array(count * 3);
    const randoms = new Float32Array(count * 4);
    const colors = new Float32Array(count * 3);
    const palette = particleColors && particleColors.length > 0 ? particleColors : defaultColors;

    for (let i = 0; i < count; i++) {
      let x: number, y: number, z: number, len: number;
      do {
        x = Math.random() * 2 - 1;
        y = Math.random() * 2 - 1;
```

```
    z = Math.random() * 2 - 1;
    len = x * x + y * y + z * z;
  } while (len > 1 || len === 0);
  const r = Math.cbrt(Math.random());
  positions.set([x * r, y * r, z * r], i * 3);
  randoms.set([Math.random(), Math.random(), Math.random(), Math.random()], i * 4);
  const col = hexToRgb(palette[Math.floor(Math.random() * palette.length)]);
  colors.set(col, i * 3);
}

const geometry = new Geometry(gl, {
  position: { size: 3, data: positions },
  random: { size: 4, data: randoms },
  color: { size: 3, data: colors }
});

const program = new Program(gl, {
  vertex,
  fragment,
  uniforms: {
    uTime: { value: 0 },
    uSpread: { value: particleSpread },
    uBaseSize: { value: particleBaseSize },
    uSizeRandomness: { value: sizeRandomness },
    uAlphaParticles: { value: alphaParticles ? 1 : 0 }
  },
  transparent: true,
  depthTest: false
});

const particles = new Mesh(gl, { mode: gl.POINTS, geometry, program });

let animationFrameId: number;
let lastTime = performance.now();
let elapsed = 0;

const update = (t: number) => {
  animationFrameId = requestAnimationFrame(update);
  const delta = t - lastTime;
  lastTime = t;
  elapsed += delta * speed;

  program.uniforms.uTime.value = elapsed * 0.001;

  if (moveParticlesOnHover) {
    particles.position.x = -mouseRef.current.x * particleHoverFactor;
    particles.position.y = -mouseRef.current.y * particleHoverFactor;
  } else {
```

```
        particles.position.x = 0;
        particles.position.y = 0;
      }

      if (!disableRotation) {
        particles.rotation.x = Math.sin(elapsed * 0.0002) * 0.1;
        particles.rotation.y = Math.cos(elapsed * 0.0005) * 0.15;
        particles.rotation.z += 0.01 * speed;
      }

      renderer.render({ scene: particles, camera });
    };

    animationFrameId = requestAnimationFrame(update);

    return () => {
      window.removeEventListener('resize', resize);
      if (moveParticlesOnHover) {
        container.removeEventListener('mousemove', handleMouseMove);
      }
      cancelAnimationFrame(animationFrameId);
      if (container.contains(gl.canvas)) {
        container.removeChild(gl.canvas);
      }
    };
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, [
    particleCount,
    particleSpread,
    speed,
    moveParticlesOnHover,
    particleHoverFactor,
    alphaParticles,
    particleBaseSize,
    sizeRandomness,
    cameraDistance,
    disableRotation
  ]);

  return <div ref={containerRef} className={`relative w-full h-full ${className}`} />;
};

export default Particles;
```