```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>应力集中现象演示 - 圆孔应力流线</title>
    <style>
        :root {
            --bg: #f5f0eb;
            --panel-bg: #ffffff;
            --text: #2c3e50;
            --accent: #c0392b;
            --accent2: #2980b9;
            --border: #d5cfc6;
            --shadow: 0 4px 20px rgba(0, 0, 0, 0.08);
            --slider-track: #ddd;
            --input-bg: #fafaf7;
        }

        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: 'Segoe UI', 'PingFang SC', 'Microsoft YaHei', 'Noto Sans SC', sans-serif;
            background: linear-gradient(135deg, #e8e0d8 0%, #f0ebe3 30%, #e5ddd4 100%);
            min-height: 100vh;
            display: flex;
            justify-content: center;
            align-items: center;
            padding: 15px;
        }

        .main-container {
            width: 100%;
            max-width: 920px;
            background: var(--panel-bg);
            border-radius: 18px;
            box-shadow: var(--shadow), 0 1px 3px rgba(0, 0, 0, 0.04);
            overflow: hidden;
            display: flex;
            flex-direction: column;
```

```
    border: 1px solid var(--border);
}

/* 标题栏 */
.header {
    background: linear-gradient(180deg, #fdfdfc 0%, #f8f6f2 100%);
    padding: 16px 24px 12px;
    border-bottom: 1px solid var(--border);
    text-align: center;
}
.header h1 {
    font-size: 1.45rem;
    font-weight: 700;
    color: #1a1a1a;
    letter-spacing: 0.03em;
    margin: 0 0 4px;
}
.header .subtitle {
    font-size: 0.85rem;
    color: #7a7368;
    font-weight: 400;
    letter-spacing: 0.02em;
}
.header .badge {
    display: inline-block;
    background: #e74c3c;
    color: #fff;
    font-size: 0.7rem;
    padding: 3px 10px;
    border-radius: 12px;
    margin-left: 6px;
    font-weight: 600;
    letter-spacing: 0.04em;
    vertical-align: middle;
    animation: pulse-badge 2.2s ease-in-out infinite;
}
@keyframes pulse-badge {
    0%,
    100% {
        opacity: 1;
    }
    50% {
        opacity: 0.7;
    }
```

```
}

/* 画布区域 */
.canvas-wrapper {
    position: relative;
    background: #fdfdfb;
    padding: 8px 10px;
    display: flex;
    justify-content: center;
    align-items: center;
    border-bottom: 1px solid var(--border);
    flex-shrink: 0;
}
canvas {
    display: block;
    max-width: 100%;
    height: auto;
    border-radius: 10px;
    background: #fefefd;
    box-shadow: inset 0 0 0 1px rgba(0, 0, 0, 0.04);
}
.canvas-label {
    position: absolute;
    top: 18px;
    left: 50%;
    transform: translateX(-50%);
    font-size: 0.8rem;
    color: #999;
    pointer-events: none;
    letter-spacing: 0.06em;
    background: rgba(255, 255, 255, 0.7);
    padding: 2px 10px;
    border-radius: 8px;
}

/* 控制面板 */
.control-panel {
    padding: 16px 20px 20px;
    display: flex;
    flex-wrap: wrap;
    gap: 16px;
    background: #fcfbf8;
    border-top: 1px solid #f0ece4;
}
```

```
.control-group {
        flex: 1 1 200px;
        min-width: 180px;
        display: flex;
        flex-direction: column;
        gap: 5px;
}
.control-group label {
        font-size: 0.82rem;
        font-weight: 600;
        color: #4a4238;
        letter-spacing: 0.02em;
        display: flex;
        align-items: center;
        gap: 6px;
}
.control-group label .icon {
        font-size: 1rem;
}
.control-row {
        display: flex;
        align-items: center;
        gap: 8px;
}
.control-row input[type="range"] {
        flex: 1;
        -webkit-appearance: none;
        height: 7px;
        border-radius: 10px;
        background: #e0d9ce;
        outline: none;
        cursor: pointer;
}
.control-row input[type="range"]::-webkit-slider-thumb {
        -webkit-appearance: none;
        width: 26px;
        height: 26px;
        border-radius: 50%;
        background: #c0392b;
        cursor: pointer;
        border: 3px solid #fff;
        box-shadow: 0 2px 8px rgba(0, 0, 0, 0.18);
        transition: all 0.15s;
}
```

```
.control-row input[type="range"]::-webkit-slider-thumb:hover {
        background: #e74c3c;
        transform: scale(1.08);
        box-shadow: 0 3px 12px rgba(0, 0, 0, 0.25);
}
.control-row input[type="range"]::-webkit-slider-thumb:active {
        background: #a93226;
        transform: scale(1.04);
}
.control-row input[type="number"] {
        width: 62px;
        padding: 7px 8px;
        border: 1.5px solid #d5cfc6;
        border-radius: 8px;
        font-size: 0.85rem;
        text-align: center;
        background: var(--input-bg);
        color: #2c3e50;
        font-weight: 500;
        transition: border-color 0.2s, box-shadow 0.2s;
        font-family: inherit;
}
.control-row input[type="number"]:focus {
        outline: none;
        border-color: #c0392b;
        box-shadow: 0 0 0 3px rgba(192, 57, 43, 0.08);
}
.unit {
        font-size: 0.7rem;
        color: #999;
        font-weight: 400;
}
.value-display {
        font-size: 0.75rem;
        color: #c0392b;
        font-weight: 700;
        min-width: 30px;
        text-align: center;
        letter-spacing: 0.02em;
}

/* 复选框样式 */
.checkbox-group {
        flex: 0 0 auto;
```

```
        display: flex;
        align-items: flex-end;
        gap: 16px;
        padding-bottom: 2px;
}
.checkbox-group label {
        display: flex;
        align-items: center;
        gap: 5px;
        font-size: 0.8rem;
        color: #4a4238;
        cursor: pointer;
        font-weight: 500;
        user-select: none;
}
.checkbox-group input[type="checkbox"] {
        accent-color: #c0392b;
        width: 16px;
        height: 16px;
        cursor: pointer;
}

/* 图例提示 */
.legend-mini {
        display: flex;
        align-items: center;
        gap: 6px;
        font-size: 0.7rem;
        color: #888;
        flex-wrap: wrap;
        margin-top: 2px;
}
.legend-mini .dot {
        width: 10px;
        height: 10px;
        border-radius: 50%;
        display: inline-block;
}
.legend-mini .dot.low {
        background: #5b9bd5;
}
.legend-mini .dot.mid {
        background: #f7dc6f;
}
```

```
            .legend-mini .dot.high {
                background: #e74c3c;
            }

            /* 响应式 */
            @media (max-width: 700px) {
                .control-panel {
                    gap: 10px;
                    padding: 12px 14px 16px;
                }
                .control-group {
                    flex: 1 1 140px;
                    min-width: 140px;
                }
                .control-row input[type="number"] {
                    width: 48px;
                    padding: 5px 4px;
                    font-size: 0.78rem;
                }
                .control-row input[type="range"]::-webkit-slider-thumb {
                    width: 22px;
                    height: 22px;
                }
                .header h1 {
                    font-size: 1.2rem;
                }
            }
        </style>
    </head>
    <body>

        <div class="main-container">
            <!-- 标题 -->
            <div class="header">
                <h1>应力集中现象 • 交互演示</h1>
                <span class="subtitle">带圆孔平板在均匀拉伸载荷下的应力流线分布</span>
                <span class="badge">K<sub>t</sub> ≈ 3.0</span>
            </div>

            <!-- 画布 -->
            <div class="canvas-wrapper">
                <canvas id="mainCanvas" width="880" height="470" aria-label="应力流线动画演示区域"></canvas>
            </div>
```

```
        <!-- 控制面板 -->
        <div class="control-panel" id="controlPanel">
            <!-- 圆孔半径 -->
            <div class="control-group">
                <label><span class="icon">⭕</span> 圆 孔 半 径 <span class="unit">(a)</span></label>
                <div class="control-row">
                    <input type="range" id="radiusSlider" min="12" max="110" value="48" step="1">
                    <span class="value-display" id="radiusValue">48</span>
                    <span class="unit">px</span>
                    <input type="number" id="radiusInput" min="12" max="110" value="48" step="1">
                </div>
            </div>

            <!-- 圆孔水平位置 -->
            <div class="control-group">
                <label><span class="icon">↔</span> 水 平 位 置 <span class="unit">(x0)</span></label>
                <div class="control-row">
                    <input type="range" id="xSlider" min="-240" max="240" value="0" step="2">
                    <span class="value-display" id="xValue">0</span>
                    <span class="unit">px</span>
                    <input type="number" id="xInput" min="-240" max="240" value="0" step="2">
                </div>
            </div>

            <!-- 圆孔垂直位置 -->
            <div class="control-group">
                <label><span class="icon">↕</span> 垂 直 位 置 <span class="unit">(y0)</span></label>
                <div class="control-row">
                    <input type="range" id="ySlider" min="-130" max="130" value="0" step="2">
                    <span class="value-display" id="yValue">0</span>
                    <span class="unit">px</span>
                    <input type="number" id="yInput" min="-130" max="130" value="0" step="2">
                </div>
            </div>
```

```
            <!-- 流线数量 -->
            <div class="control-group">
                <label><span class="icon">〰</span> 流线数量</label>
                <div class="control-row">
                    <input type="range" id="streamSlider" min="6" max="32" value="17" step="1">
                    <span class="value-display" id="streamValue">17</span>
                    <span class="unit">条</span>
                    <input type="number" id="streamInput" min="6" max="32" value="17" step="1">
                </div>
            </div>

            <!-- 复选框 -->
            <div class="checkbox-group">
                <label>
                    <input type="checkbox" id="heatmapCheckbox" checked> 显示应力热力图
                </label>
                <label>
                    <input type="checkbox" id="particleCheckbox" checked> 显示流动粒子
                </label>
            </div>

            <!-- 图例 -->
            <div class="legend-mini">
                应力水平：
                <span class="dot low"></span> 低
                <span class="dot mid"></span> 中
                <span class="dot high"></span> 高（应力集中）
                 |  流线越<span style="color:#e74c3c;font-weight:600;">密</span>→应力越<span style="color:#e74c3c;font-weight:600;">大</span>
            </div>
        </div>
    </div>

    <script>
        (function() {
            // ============ DOM 元素 ============
            const canvas = document.getElementById('mainCanvas');
            const ctx = canvas.getContext('2d');
```

```
// 滑块
const radiusSlider = document.getElementById('radiusSlider');
const xSlider = document.getElementById('xSlider');
const ySlider = document.getElementById('ySlider');
const streamSlider = document.getElementById('streamSlider');

// 数字输入
const radiusInput = document.getElementById('radiusInput');
const xInput = document.getElementById('xInput');
const yInput = document.getElementById('yInput');
const streamInput = document.getElementById('streamInput');

// 显示值
const radiusValue = document.getElementById('radiusValue');
const xValue = document.getElementById('xValue');
const yValue = document.getElementById('yValue');
const streamValue = document.getElementById('streamValue');

// 复选框
const heatmapCheckbox = document.getElementById('heatmapCheckbox');
const particleCheckbox = document.getElementById('particleCheckbox');

// ============ 参数 ============
const canvasW = canvas.width;
const canvasH = canvas.height;

// 工件在 Canvas 中的矩形区域
const workpieceLeft = 85;
const workpieceRight = canvasW - 85;
const workpieceTop = 55;
const workpieceBottom = canvasH - 55;
const workpieceCenterX = (workpieceLeft + workpieceRight) / 2;
const workpieceCenterY = (workpieceTop + workpieceBottom) / 2;
const workpieceHalfW = (workpieceRight - workpieceLeft) / 2;
const workpieceHalfH = (workpieceBottom - workpieceTop) / 2;

// 物理坐标原点在工件中心，y 向上
function toCanvasX(px) { return workpieceCenterX + px; }

function toCanvasY(py) { return workpieceCenterY - py; }

function toPhysX(cx) { return cx - workpieceCenterX; }

function toPhysY(cy) { return workpieceCenterY - cy; }
```

```
// 状态
let holeRadius = 48; // 圆孔半径 (物理像素)
let holeX = 0; // 圆孔中心物理 x
let holeY = 0; // 圆孔中心物理 y
let numStreamlines = 17; // 流线数量
let showHeatmap = true;
let showParticles = true;

// 粒子系统
let particles = [];
const numParticles = 90;
const farFieldSpeed = 0.35; // 远场粒子速度（每帧像素）

// 流线缓存
let streamlineCache = [];
let heatmapCache = null; // 存储热力图 ImageData

// ============ 坐标转换工具 ============
function physToCanvas(px, py) {
    return { cx: toCanvasX(px), cy: toCanvasY(py) };
}

// ============ 三次方程求解（牛顿法）============
// 求解: Y³ - Y_far*Y² + Y*(X² - a²) - Y_far*X² = 0
// 其中 Y = y - holeY, X = x - holeX, Y_far = y_far - holeY
function solveCubicForY(Y_far, X, a, initialGuess) {
    const X2 = X * X;
    const a2 = a * a;

    // 如果|X|非常大，Y ≈ Y_far
    if (Math.abs(X) > 20 * a && Math.abs(initialGuess - Y_far) < a * 0.5) {
        return Y_far;
    }

    let Y = initialGuess;
    const maxIter = 30;
    const tol = 1e-8;

    for (let iter = 0; iter < maxIter; iter++) {
        const Y2 = Y * Y;
        const Y3 = Y2 * Y;
        const f = Y3 - Y_far * Y2 + Y * (X2 - a2) - Y_far * X2;
        const df = 3 * Y2 - 2 * Y_far * Y + (X2 - a2);
```

```
            if (Math.abs(df) < 1e-12) {
                // 导数为零，微调
                Y += (Math.random() - 0.5) * 0.01;
                continue;
            }

            const delta = f / df;
            Y -= delta;

            if (Math.abs(delta) < tol) break;

            // 防止发散
            if (Math.abs(Y) > Math.abs(Y_far) * 3 + a * 5) {
                Y = initialGuess;
                break;
            }
        }

        // 确保 Y 在合理范围（对于 Y_far>0，Y 应大于圆孔上方边界）
        if (Y_far > 0 && Math.abs(X) < a) {
            const boundY = Math.sqrt(Math.max(0, a2 - X2));
            if (Y < boundY + 0.01) Y = boundY + 0.05;
        }
        if (Y_far < 0 && Math.abs(X) < a) {
            const boundY = -Math.sqrt(Math.max(0, a2 - X2));
            if (Y > boundY - 0.01) Y = boundY - 0.05;
        }

        return Y;
    }

    // ============ 计算单条流线 ============
    function computeStreamline(yFar, a, hx, hy) {
        const Y_far = yFar - hy;
        const points = [];
        const xMin = -workpieceHalfW - 60;
        const xMax = workpieceHalfW + 60;
        const step = 1.5; // 步长（物理像素）

        let prevY = Y_far; // 初始猜测
        let x = xMin;
        let firstPoint = true;
```

```
        while (x <= xMax) {
            const X = x - hx;
            let guess;
            if (firstPoint) {
                guess = Y_far;
                firstPoint = false;
            } else {
                guess = prevY;
            }

            const Y = solveCubicForY(Y_far, X, a, guess);
            const py = Y + hy;
            const px = x;

            // 检查是否在工件内部（用于裁剪）
            const inWorkpiece =
                px >= -workpieceHalfW && px <= workpieceHalfW &&
                py >= -workpieceHalfH && py <= workpieceHalfH;

            points.push({
                px: px,
                py: py,
                inWorkpiece: inWorkpiece,
                Y: Y,
                X: X,
            });

            prevY = Y;
            x += step;
        }

        return points;
    }

    // ============ 计算所有流线 ============
    function computeAllStreamlines() {
        streamlineCache = [];
        const a = holeRadius;
        const hx = holeX;
        const hy = holeY;

        // 生成均匀分布的远场 y 坐标
        const margin = 8; // 距离工件边界的边距
        const availableHeight = 2 * workpieceHalfH - 2 * margin;
```

```
            const spacing = availableHeight / (numStreamlines - 1);
            const yStart = -workpieceHalfH + margin;

            for (let i = 0; i < numStreamlines; i++) {
                const yFar = yStart + i * spacing;
                // 跳过恰好穿过圆孔中心的流线（会导致数值问题）
                const adjustedYFar = Math.abs(yFar - hy) < 0.15 ? hy + 0.15 *
Math.sign(yFar - hy + 0.001) : yFar;
                const slPoints = computeStreamline(adjustedYFar, a, hx, hy);
                streamlineCache.push({
                    yFar: adjustedYFar,
                    points: slPoints,
                });
            }
        }

        // ============ 热力图计算 ============
        function computeHeatmapData(gridW, gridH) {
            const a = holeRadius;
            const hx = holeX;
            const hy = holeY;
            const data = new Float32Array(gridW * gridH);

            const physXMin = -workpieceHalfW;
            const physXMax = workpieceHalfW;
            const physYMin = -workpieceHalfH;
            const physYMax = workpieceHalfH;
            const dx = (physXMax - physXMin) / gridW;
            const dy = (physYMax - physYMin) / gridH;

            const a2 = a * a;
            const sigma0 = 1.0;

            for (let iy = 0; iy < gridH; iy++) {
                const py = physYMin + (iy + 0.5) * dy;
                for (let ix = 0; ix < gridW; ix++) {
                    const px = physXMin + (ix + 0.5) * dx;
                    const rx = px - hx;
                    const ry = py - hy;
                    const r2 = rx * rx + ry * ry;
                    const r = Math.sqrt(r2);

                    let sigmaMax;
                    if (r < a * 1.001 && r > 0) {
```

```
                    // 在圆孔边界或内部
                    sigmaMax = 0;
                } else if (r < 0.01) {
                    sigmaMax = sigma0;
                } else {
                    const rho = r / a;
                    const rho2 = rho * rho;
                    const rho4 = rho2 * rho2;
                    const cosTheta = ry / r;
                    const sinTheta = rx / r;
                    const cos2t = cosTheta * cosTheta - sinTheta * sinTheta;
                    const sin2t = 2 * sinTheta * cosTheta;

                    // 极坐标应力分量（无量纲，除以 sigma0）
                    const sigmaR = 0.5 * (1 - 1 / rho2) + 0.5 * (1 - 4 / rho2 + 3 /
rho4) * cos2t;
                    const sigmaTheta = 0.5 * (1 + 1 / rho2) - 0.5 * (1 + 3 / rho4) *
cos2t;
                    const tauRTheta = -0.5 * (1 + 2 / rho2 - 3 / rho4) * sin2t;

                    // 转换到直角坐标
                    const cs = cosTheta;
                    const sn = sinTheta;
                    const cs2 = cs * cs;
                    const sn2 = sn * sn;
                    const sigmaX = sigmaR * cs2 + sigmaTheta * sn2 - 2 *
tauRTheta * sn * cs;
                    const sigmaY = sigmaR * sn2 + sigmaTheta * cs2 + 2 *
tauRTheta * sn * cs;
                    const tauXY = (sigmaR - sigmaTheta) * sn * cs + tauRTheta *
(cs2 - sn2);

                    // 最大主应力
                    const avg = (sigmaX + sigmaY) / 2;
                    const diff = (sigmaX - sigmaY) / 2;
                    sigmaMax = avg + Math.sqrt(diff * diff + tauXY * tauXY);
                }

                data[iy * gridW + ix] = sigmaMax;
            }
        }
        return { data, gridW, gridH, physXMin, physXMax, physYMin, physYMax };
    }
```

```
function getHeatmapColor(value) {
    // value = sigmaMax/sigma0, 范围大约 0 到 3
    const v = Math.max(0, Math.min(3.2, value));
    const t = v / 3.2; // 归一化到 0-1

    // 颜色映射：深蓝(低应力) -> 浅蓝 -> 白 -> 黄 -> 橙 -> 红(高应力>3)
    let r, g, b;
    if (t < 0.25) {
        // 深蓝到浅蓝
        const s = t / 0.25;
        r = 30 + s * 130;
        g = 60 + s * 150;
        b = 180 - s * 50;
    } else if (t < 0.5) {
        // 浅蓝到白色
        const s = (t - 0.25) / 0.25;
        r = 160 + s * 95;
        g = 210 + s * 45;
        b = 130 + s * 110;
    } else if (t < 0.7) {
        // 白色到黄色
        const s = (t - 0.5) / 0.2;
        r = 255;
        g = 255 - s * 100;
        b = 240 - s * 200;
    } else if (t < 0.9) {
        // 黄色到橙红
        const s = (t - 0.7) / 0.2;
        r = 255;
        g = 155 - s * 80;
        b = 40 - s * 30;
    } else {
        // 橙红到深红
        const s = (t - 0.9) / 0.1;
        r = 255 - s * 30;
        g = 75 - s * 40;
        b = 10 - s * 5;
    }
    return {
        r: Math.round(Math.max(0, Math.min(255, r))),
        g: Math.round(Math.max(0, Math.min(255, g))),
        b: Math.round(Math.max(0, Math.min(255, b))),
    };
}
```

```
function renderHeatmapToOffscreen(gridData) {
    const { data, gridW, gridH, physXMin, physXMax, physYMin, physYMax } =
gridData;
    const offCanvas = document.createElement('canvas');
    offCanvas.width = gridW;
    offCanvas.height = gridH;
    const offCtx = offCanvas.getContext('2d');
    const imgData = offCtx.createImageData(gridW, gridH);

    for (let iy = 0; iy < gridH; iy++) {
        for (let ix = 0; ix < gridW; ix++) {
            const val = data[iy * gridW + ix];
            const color = getHeatmapColor(val);
            const idx = (iy * gridW + ix) * 4;
            imgData.data[idx] = color.r;
            imgData.data[idx + 1] = color.g;
            imgData.data[idx + 2] = color.b;
            imgData.data[idx + 3] = 200; // 半透明
        }
    }
    offCtx.putImageData(imgData, 0, 0);
    return { offCanvas, physXMin, physXMax, physYMin, physYMax };
}

let cachedHeatmapOffscreen = null;

function updateHeatmapCache() {
    const gridW = 180;
    const gridH = 100;
    const gridData = computeHeatmapData(gridW, gridH);
    cachedHeatmapOffscreen = renderHeatmapToOffscreen(gridData);
}

// ============ 粒子系统 ============
function initParticles() {
    particles = [];
    for (let i = 0; i < numParticles; i++) {
        particles.push({
            px: -workpieceHalfW + Math.random() * 2 * workpieceHalfW,
            py: -workpieceHalfH + Math.random() * 2 * workpieceHalfH,
            active: true,
        });
    }
```

```
}

function getVelocityAt(px, py) {
    const a = holeRadius;
    const hx = holeX;
    const hy = holeY;
    const rx = px - hx;
    const ry = py - hy;
    const r2 = rx * rx + ry * ry;
    const r = Math.sqrt(r2);
    const a2 = a * a;
    const r4 = r2 * r2;

    if (r < a * 1.002) {
        return { vx: 0, vy: 0, speed: 0 };
    }

    // 势流速度场（远场速度 U=1）
    const factor1 = 1 - a2 / r2;
    const factor2 = 2 * a2 * ry * ry / r4;
    const vx = factor1 + factor2;
    const vy = -2 * a2 * rx * ry / r4;
    const speed = Math.sqrt(vx * vx + vy * vy);

    return { vx, vy, speed };
}

function updateParticles() {
    const dt = 1.0;
    const a = holeRadius;
    const hx = holeX;
    const hy = holeY;

    for (const p of particles) {
        if (!p.active) {
            // 重置到左边界
            p.px = -workpieceHalfW - 5 + Math.random() * 15;
            p.py = -workpieceHalfH + Math.random() * 2 * workpieceHalfH;
            p.active = true;
            continue;
        }

        const vel = getVelocityAt(p.px, p.py);
        const speed = vel.speed;
```

```
                    if (speed < 0.001) {
                        p.active = false;
                        continue;
                    }

                    // 调整步长使动画平滑
                    const stepSize = farFieldSpeed / Math.max(0.3, speed);
                    const adjustedDt = dt * stepSize;

                    const newPx = p.px + vel.vx * adjustedDt;
                    const newPy = p.py + vel.vy * adjustedDt;

                    // 检查是否碰到圆孔
                    const rx = newPx - hx;
                    const ry = newPy - hy;
                    const distToHole = Math.sqrt(rx * rx + ry * ry);

                    if (distToHole < a * 1.01) {
                        // 粒子碰到圆孔，沿切向弹开
                        const angle = Math.atan2(ry, rx);
                        const bounceAngle = angle + (Math.random() > 0.5 ? 1 : -1) * (0.3 +
Math.random() * 0.6);

                        p.px = hx + (a + 2) * Math.cos(bounceAngle);
                        p.py = hy + (a + 2) * Math.sin(bounceAngle);
                    } else {
                        p.px = newPx;
                        p.py = newPy;
                    }

                    // 检查是否超出工件范围
                    if (p.px > workpieceHalfW + 10 || p.px < -workpieceHalfW - 20 ||
                        p.py > workpieceHalfH + 10 || p.py < -workpieceHalfH - 10) {
                        p.active = false;
                    }
                }
            }

            // ============ 绘制函数 ============
            function drawWorkpieceOutline() {
                const cxLeft = toCanvasX(-workpieceHalfW);
                const cxRight = toCanvasX(workpieceHalfW);
                const cyTop = toCanvasY(workpieceHalfH);
                const cyBottom = toCanvasY(-workpieceHalfH);
```

```
        // 工件填充
        ctx.fillStyle = '#fdfdfc';
        ctx.fillRect(cxLeft, cyTop, cxRight - cxLeft, cyBottom - cyTop);

        // 工件边框
        ctx.strokeStyle = '#3a3530';
        ctx.lineWidth = 3;
        ctx.strokeRect(cxLeft, cyTop, cxRight - cxLeft, cyBottom - cyTop);

        // 阴影效果（底部和右侧）
        ctx.strokeStyle = 'rgba(0,0,0,0.12)';
        ctx.lineWidth = 5;
        ctx.beginPath();
        ctx.moveTo(cxRight, cyTop);
        ctx.lineTo(cxRight, cyBottom + 2);
        ctx.lineTo(cxLeft - 2, cyBottom + 2);
        ctx.stroke();
    }

    function drawHole() {
        const cx = toCanvasX(holeX);
        const cy = toCanvasY(holeY);
        const r = holeRadius;

        // 圆孔阴影
        ctx.fillStyle = 'rgba(0,0,0,0.08)';
        ctx.beginPath();
        ctx.arc(cx + 2, cy + 2, r, 0, Math.PI * 2);
        ctx.fill();

        // 圆孔主体
        const holeGrad = ctx.createRadialGradient(cx - r * 0.2, cy - r * 0.25, r * 0.1, cx,
cy, r);
        holeGrad.addColorStop(0, '#f5f0e8');
        holeGrad.addColorStop(0.7, '#e8e0d5');
        holeGrad.addColorStop(0.9, '#d5ccc0');
        holeGrad.addColorStop(1, '#b8aFA0');
        ctx.fillStyle = holeGrad;
        ctx.beginPath();
        ctx.arc(cx, cy, r, 0, Math.PI * 2);
        ctx.fill();

        // 圆孔边框
```

```
            ctx.strokeStyle = '#5a4e3e';
            ctx.lineWidth = 2.5;
            ctx.beginPath();
            ctx.arc(cx, cy, r, 0, Math.PI * 2);
            ctx.stroke();

            // 内阴影
            ctx.strokeStyle = 'rgba(0,0,0,0.2)';
            ctx.lineWidth = 4;
            ctx.beginPath();
            ctx.arc(cx, cy, r - 2, -Math.PI * 0.6, -Math.PI * 0.15);
            ctx.stroke();
        }

        function drawHeatmapBackground() {
            if (!showHeatmap || !cachedHeatmapOffscreen) return;

            const { offCanvas, physXMin, physXMax, physYMin, physYMax } =
cachedHeatmapOffscreen;

            const cxLeft = toCanvasX(physXMin);
            const cxRight = toCanvasX(physXMax);
            const cyTop = toCanvasY(physYMax);
            const cyBottom = toCanvasY(physYMin);

            ctx.save();
            // 裁剪到工件内部
            ctx.beginPath();
            const wl = toCanvasX(-workpieceHalfW);
            const wr = toCanvasX(workpieceHalfW);
            const wt = toCanvasY(workpieceHalfH);
            const wb = toCanvasY(-workpieceHalfH);
            ctx.rect(wl, wt, wr - wl, wb - wt);
            ctx.clip();

            // 绘制热力图
            ctx.drawImage(offCanvas, cxLeft, cyTop, cxRight - cxLeft, cyBottom - cyTop);

            ctx.restore();
        }

        function drawStreamlines() {
            const a = holeRadius;
            const hx = holeX;
```

```
const hy = holeY;

ctx.save();
// 裁剪到工件内部
ctx.beginPath();
const wl = toCanvasX(-workpieceHalfW);
const wr = toCanvasX(workpieceHalfW);
const wt = toCanvasY(workpieceHalfH);
const wb = toCanvasY(-workpieceHalfH);
ctx.rect(wl, wt, wr - wl, wb - wt);
ctx.clip();

for (const sl of streamlineCache) {
    const pts = sl.points;
    if (pts.length < 2) continue;

    ctx.beginPath();
    let started = false;
    let prevInWorkpiece = false;

    for (let i = 0; i < pts.length; i++) {
        const pt = pts[i];
        const cx = toCanvasX(pt.px);
        const cy = toCanvasY(pt.py);
        const inWp = pt.inWorkpiece;

        if (inWp && !started) {
            ctx.moveTo(cx, cy);
            started = true;
        } else if (inWp && prevInWorkpiece) {
            ctx.lineTo(cx, cy);
        } else if (inWp && !prevInWorkpiece) {
            // 重新进入工件
            ctx.moveTo(cx, cy);
            started = true;
        } else if (!inWp && prevInWorkpiece) {
            // 离开工件，绘制到边界
            ctx.lineTo(cx, cy);
            started = false;
        }
        prevInWorkpiece = inWp;
    }

    ctx.strokeStyle = 'rgba(30,45,70,0.75)';
```

```
            ctx.lineWidth = 1.3;
            ctx.stroke();
        }

        ctx.restore();
    }

    function drawParticles() {
        if (!showParticles) return;

        const a = holeRadius;
        const hx = holeX;
        const hy = holeY;

        ctx.save();
        ctx.beginPath();
        const wl = toCanvasX(-workpieceHalfW);
        const wr = toCanvasX(workpieceHalfW);
        const wt = toCanvasY(workpieceHalfH);
        const wb = toCanvasY(-workpieceHalfH);
        ctx.rect(wl, wt, wr - wl, wb - wt);
        ctx.clip();

        for (const p of particles) {
            if (!p.active) continue;
            const cx = toCanvasX(p.px);
            const cy = toCanvasY(p.py);

            // 检查是否在工件内
            if (p.px < -workpieceHalfW || p.px > workpieceHalfW ||
                p.py < -workpieceHalfH || p.py > workpieceHalfH) continue;

            // 检查是否在圆孔内
            const rx = p.px - hx;
            const ry = p.py - hy;
            if (Math.sqrt(rx * rx + ry * ry) < a * 0.98) continue;

            // 发光粒子
            const vel = getVelocityAt(p.px, p.py);
            const stressLevel = Math.min(3, vel.speed);
            const intensity = 0.5 + stressLevel / 3 * 0.5;

            // 光晕
            const glowGrad = ctx.createRadialGradient(cx, cy, 0, cx, cy, 5);
```

```
                        const alpha = Math.round(160 * intensity);
                        if (stressLevel > 1.8) {
                            glowGrad.addColorStop(0, `rgba(220,50,30,${alpha/255})`);
                            glowGrad.addColorStop(0.5, `rgba(240,120,40,${alpha/255*0.6})`);
                            glowGrad.addColorStop(1, 'rgba(240,120,40,0)');
                        } else if (stressLevel > 1.2) {
                            glowGrad.addColorStop(0, `rgba(230,150,40,${alpha/255})`);
                            glowGrad.addColorStop(0.5, `rgba(240,180,60,${alpha/255*0.5})`);
                            glowGrad.addColorStop(1, 'rgba(240,180,60,0)');
                        } else {
                            glowGrad.addColorStop(0, `rgba(60,130,200,${alpha/255})`);
                            glowGrad.addColorStop(0.5,
`rgba(100,160,220,${alpha/255*0.4})`);
                            glowGrad.addColorStop(1, 'rgba(100,160,220,0)');
                        }
                        ctx.fillStyle = glowGrad;
                        ctx.beginPath();
                        ctx.arc(cx, cy, 5, 0, Math.PI * 2);
                        ctx.fill();

                        // 粒子核心
                        if (stressLevel > 1.8) {
                            ctx.fillStyle = 'rgba(200,35,20,0.9)';
                        } else if (stressLevel > 1.2) {
                            ctx.fillStyle = 'rgba(220,130,30,0.85)';
                        } else {
                            ctx.fillStyle = 'rgba(40,100,180,0.8)';
                        }
                        ctx.beginPath();
                        ctx.arc(cx, cy, 2.2, 0, Math.PI * 2);
                        ctx.fill();
                    }

                    ctx.restore();
                }

                function drawArrows() {
                    // 左侧拉伸箭头
                    const arrowY = toCanvasY(0);
                    const arrowLeftX = toCanvasX(-workpieceHalfW - 25);
                    const arrowRightX = toCanvasX(workpieceHalfW + 25);

                    const drawArrow = (x, y, direction) => {
                        const sign = direction === 'right' ? 1 : -1;
```

```
            const tipX = x + sign * 28;
            const baseX = x - sign * 22;

            ctx.fillStyle = 'rgba(180,60,40,0.85)';
            ctx.strokeStyle = 'rgba(140,40,25,0.9)';
            ctx.lineWidth = 2;

            // 箭杆
            ctx.beginPath();
            ctx.moveTo(baseX, y);
            ctx.lineTo(tipX - sign * 8, y);
            ctx.stroke();

            // 箭头
            ctx.beginPath();
            ctx.moveTo(tipX, y);
            ctx.lineTo(tipX - sign * 10, y - 7);
            ctx.lineTo(tipX - sign * 10, y + 7);
            ctx.closePath();
            ctx.fill();
            ctx.stroke();
        };

        drawArrow(arrowLeftX, arrowY, 'right');
        drawArrow(arrowRightX, arrowY, 'left');

        // 标签
        ctx.fillStyle = '#8b3a30';
        ctx.font = 'bold 13px "PingFang SC","Microsoft YaHei","Segoe UI",sans-serif';
        ctx.textAlign = 'center';
        ctx.fillText('σ', toCanvasX(-workpieceHalfW - 38), arrowY + 5);
        ctx.fillText('σ', toCanvasX(workpieceHalfW + 38), arrowY + 5);
        ctx.fillText('拉伸方向', toCanvasX(-workpieceHalfW - 38), arrowY - 22);
        ctx.fillText('拉伸方向', toCanvasX(workpieceHalfW + 38), arrowY - 22);
        ctx.textAlign = 'start';
    }

    function drawAnnotations() {
        const cx = toCanvasX(holeX);
        const cy = toCanvasY(holeY);
        const a = holeRadius;

        // 应力集中区域标记（圆孔顶部和底部）
        const topY = cy - a;
```

```
            const bottomY = cy + a;

            // 顶部应力集中标记
            ctx.fillStyle = 'rgba(200,40,30,0.7)';
            ctx.font = 'bold 11px "PingFang SC","Microsoft YaHei","Segoe UI",sans-serif';
            ctx.textAlign = 'center';
            ctx.fillText('应力集中', cx, topY - 18);
            ctx.fillText('Kt≈3.0', cx, topY - 35);

            // 小括号标记
            ctx.strokeStyle = 'rgba(200,40,30,0.5)';
            ctx.lineWidth = 1.5;
            ctx.setLineDash([3, 3]);
            ctx.beginPath();
            ctx.arc(cx, topY - 8, a + 16, -Math.PI * 0.55, -Math.PI * 0.45, false);
            ctx.stroke();
            ctx.beginPath();
            ctx.arc(cx, bottomY + 8, a + 16, Math.PI * 0.45, Math.PI * 0.55, false);
            ctx.stroke();
            ctx.setLineDash([]);

            // 底部应力集中标记
            ctx.fillText('应力集中', cx, bottomY + 28);
            ctx.fillText('Kt≈3.0', cx, bottomY + 45);

            // 流线疏密说明
            ctx.fillStyle = 'rgba(60,50,40,0.7)';
            ctx.font = 'italic 11px "PingFang SC","Microsoft YaHei","Segoe UI",sans-serif';
            ctx.textAlign = 'right';
            const labelX = toCanvasX(workpieceHalfW - 15);
            ctx.fillText('流线密集区 → 高应力', labelX, toCanvasY(workpieceHalfH *
0.55));
            ctx.fillText('流线稀疏区 → 低应力', labelX, toCanvasY(workpieceHalfH *
0.05));
            ctx.textAlign = 'start';
        }

        // ============ 主渲染 ============
        function render() {
            ctx.clearRect(0, 0, canvasW, canvasH);

            // 背景
            const bgGrad = ctx.createRadialGradient(canvasW / 2, canvasH / 2, canvasW *
0.15, canvasW / 2, canvasH /
```

```
            2, canvasW * 0.8);
        bgGrad.addColorStop(0, '#fefefd');
        bgGrad.addColorStop(1, '#f5f2ec');
        ctx.fillStyle = bgGrad;
        ctx.fillRect(0, 0, canvasW, canvasH);

        // 绘制顺序
        drawHeatmapBackground();
        drawWorkpieceOutline();
        drawHole();
        drawStreamlines();
        drawParticles();
        drawArrows();
        drawAnnotations();

        // 标题水印
        ctx.fillStyle = 'rgba(0,0,0,0.04)';
        ctx.font = 'bold 28px "PingFang SC","Microsoft YaHei","Segoe UI",sans-serif';
        ctx.textAlign = 'center';
        ctx.fillText('应力集中 • 圆孔应力流线', canvasW / 2, canvasH - 18);
        ctx.textAlign = 'start';
}

// ============ 动画循环 ============
function animate() {
        updateParticles();
        render();
        requestAnimationFrame(animate);
}

// ============ 参数更新 ============
function updateAllParams() {
        holeRadius = parseFloat(radiusInput.value) || 48;
        holeX = parseFloat(xInput.value) || 0;
        holeY = parseFloat(yInput.value) || 0;
        numStreamlines = parseInt(streamInput.value) || 17;
        showHeatmap = heatmapCheckbox.checked;
        showParticles = particleCheckbox.checked;

        // 限制范围
        holeRadius = Math.max(12, Math.min(110, holeRadius));
        holeX = Math.max(-240, Math.min(240, holeX));
        holeY = Math.max(-130, Math.min(130, holeY));
        numStreamlines = Math.max(6, Math.min(32, numStreamlines));
```

```
        // 确保圆孔完全在工件内
        const maxR = Math.min(
            workpieceHalfW - Math.abs(holeX) - 8,
            workpieceHalfH - Math.abs(holeY) - 8
        );
        if (holeRadius > maxR) {
            holeRadius = Math.max(12, maxR);
        }

        // 同步 UI
        radiusSlider.value = holeRadius;
        radiusInput.value = holeRadius;
        radiusValue.textContent = Math.round(holeRadius);

        xSlider.value = holeX;
        xInput.value = holeX;
        xValue.textContent = Math.round(holeX);

        ySlider.value = holeY;
        yInput.value = holeY;
        yValue.textContent = Math.round(holeY);

        streamSlider.value = numStreamlines;
        streamInput.value = numStreamlines;
        streamValue.textContent = numStreamlines;

        // 重新计算
        computeAllStreamlines();
        updateHeatmapCache();
    }

    // ============ 事件监听 ============
    function syncSliderToInput(slider, input, valueDisplay, formatFn) {
        const val = parseFloat(slider.value);
        input.value = val;
        if (valueDisplay && formatFn) {
            valueDisplay.textContent = formatFn(val);
        } else if (valueDisplay) {
            valueDisplay.textContent = Math.round(val);
        }
        updateAllParams();
    }
```

```
function syncInputToSlider(input, slider, valueDisplay, formatFn) {
    let val = parseFloat(input.value);
    if (isNaN(val)) return;
    val = Math.max(parseFloat(input.min), Math.min(parseFloat(input.max), val));
    input.value = val;
    slider.value = val;
    if (valueDisplay && formatFn) {
        valueDisplay.textContent = formatFn(val);
    } else if (valueDisplay) {
        valueDisplay.textContent = Math.round(val);
    }
    updateAllParams();
}

radiusSlider.addEventListener('input', () => {
    syncSliderToInput(radiusSlider, radiusInput, radiusValue, v => Math.round(v));
});
radiusInput.addEventListener('input', () => {
    syncInputToSlider(radiusInput, radiusSlider, radiusValue, v => Math.round(v));
});
radiusInput.addEventListener('change', () => {
    syncInputToSlider(radiusInput, radiusSlider, radiusValue, v => Math.round(v));
});

xSlider.addEventListener('input', () => {
    syncSliderToInput(xSlider, xInput, xValue, v => Math.round(v));
});
xInput.addEventListener('input', () => {
    syncInputToSlider(xInput, xSlider, xValue, v => Math.round(v));
});
xInput.addEventListener('change', () => {
    syncInputToSlider(xInput, xSlider, xValue, v => Math.round(v));
});

ySlider.addEventListener('input', () => {
    syncSliderToInput(ySlider, yInput, yValue, v => Math.round(v));
});
yInput.addEventListener('input', () => {
    syncInputToSlider(yInput, ySlider, yValue, v => Math.round(v));
});
yInput.addEventListener('change', () => {
    syncInputToSlider(yInput, ySlider, yValue, v => Math.round(v));
});
```

```
        streamSlider.addEventListener('input', () => {
            syncSliderToInput(streamSlider, streamInput, streamValue, v => Math.round(v));
        });
        streamInput.addEventListener('input', () => {
            syncInputToSlider(streamInput, streamSlider, streamValue, v => Math.round(v));
        });
        streamInput.addEventListener('change', () => {
            syncInputToSlider(streamInput, streamSlider, streamValue, v => Math.round(v));
        });

        heatmapCheckbox.addEventListener('change', updateAllParams);
        particleCheckbox.addEventListener('change', updateAllParams);

        // ============ 键盘快捷键 ============
        document.addEventListener('keydown', (e) => {
            switch (e.key.toLowerCase()) {
                case 'r':
                    // 重置
                    holeRadius = 48;
                    holeX = 0;
                    holeY = 0;
                    numStreamlines = 17;
                    showHeatmap = true;
                    showParticles = true;
                    heatmapCheckbox.checked = true;
                    particleCheckbox.checked = true;
                    updateAllParams();
                    // 同步所有 UI
                    radiusSlider.value = 48;
                    radiusInput.value = 48;
                    radiusValue.textContent = '48';
                    xSlider.value = 0;
                    xInput.value = 0;
                    xValue.textContent = '0';
                    ySlider.value = 0;
                    yInput.value = 0;
                    yValue.textContent = '0';
                    streamSlider.value = 17;
                    streamInput.value = 17;
                    streamValue.textContent = '17';
                    break;
```

```
case 'h':
    // 切换热力图
    showHeatmap = !showHeatmap;
    heatmapCheckbox.checked = showHeatmap;
    updateAllParams();
    break;
case 'p':
    // 切换粒子
    showParticles = !showParticles;
    particleCheckbox.checked = showParticles;
    updateAllParams();
    break;
case 'arrowleft':
    holeX = Math.max(-240, holeX - 5);
    xSlider.value = holeX;
    xInput.value = holeX;
    xValue.textContent = Math.round(holeX);
    updateAllParams();
    e.preventDefault();
    break;
case 'arrowright':
    holeX = Math.min(240, holeX + 5);
    xSlider.value = holeX;
    xInput.value = holeX;
    xValue.textContent = Math.round(holeX);
    updateAllParams();
    e.preventDefault();
    break;
case 'arrowup':
    holeY = Math.min(130, holeY + 5);
    ySlider.value = holeY;
    yInput.value = holeY;
    yValue.textContent = Math.round(holeY);
    updateAllParams();
    e.preventDefault();
    break;
case 'arrowdown':
    holeY = Math.max(-130, holeY - 5);
    ySlider.value = holeY;
    yInput.value = holeY;
    yValue.textContent = Math.round(holeY);
    updateAllParams();
    e.preventDefault();
    break;
```

```
        }
    });

    // ============ 初始化 ============
    function init() {
        updateAllParams();
        initParticles();
        computeAllStreamlines();
        updateHeatmapCache();
        render();
    }

    init();
    animate();

    // ============ 响应式 Canvas 缩放 ============
    function handleResize() {
        const wrapper = document.querySelector('.canvas-wrapper');
        if (wrapper && window.innerWidth < 920) {
            const scale = Math.min(1, (window.innerWidth - 30) / canvasW);
            canvas.style.width = (canvasW * scale) + 'px';
            canvas.style.height = (canvasH * scale) + 'px';
        } else if (canvas.style.width) {
            canvas.style.width = '';
            canvas.style.height = '';
        }
    }
    window.addEventListener('resize', handleResize);
    handleResize();

    console.log('✅ 应力集中现象演示已就绪');
    console.log('   🖱️ 拖动下方滑块或直接输入数值来调整参数');
    console.log('   ⌨️ 键盘快捷键: R=重置 H=切换热力图 P=切换粒子 方向键=移动圆孔');
    console.log('   📐 圆孔两侧流线密集处即为应力集中区域(Kt≈3.0)');
    console.log('   🔴 粒子在应力集中区域移动更快、颜色更暖');
})();
</script>
</body>
</html>
```