```
<script>
  const LiquidButton = class LiquidButton {
  constructor(svg) {
    const options = svg.dataset;
    this.id = this.constructor.id || (this.constructor.id = 1);
    this.constructor.id++;
    this.xmlns = 'http://www.w3.org/2000/svg';
    this.tension = options.tension * 1 || 0.4;
    this.width  = options.width  * 1 || 218;
    this.height  = options.height * 1 || 56;
    this.margin  = options.margin  || 40;
    this.hoverFactor = options.hoverFactor || -0.1;
    this.gap    = options.gap    || 5;
    this.debug  = options.debug  || false;
    this.forceFactor = options.forceFactor || 0.2;
    this.color1 = options.color1 || '#151617';
    this.color2 = options.color2 || '#151617';
    this.color3 = options.color3 || '#151617';
    this.textColor = options.textColor || '#fff';
    this.text = options.text   || 'Button';
    this.svg = svg;
    this.layers = [{
      points: [],
      viscosity: 0.5,
      mouseForce: 100,
      forceLimit: 2,
    },{
      points: [],
      viscosity: 0.8,
      mouseForce: 150,
      forceLimit: 3,
    }];
    for (let layerIndex = 0; layerIndex < this.layers.length; layerIndex++) {
      const layer = this.layers[layerIndex];
      layer.viscosity = options['layer-' + (layerIndex + 1) + 'Viscosity'] * 1 || layer.viscosity;
      layer.mouseForce = options['layer-' + (layerIndex + 1) + 'MouseForce'] * 1 || layer.mouseForce;
      layer.forceLimit = options['layer-' + (layerIndex + 1) + 'ForceLimit'] * 1 || layer.forceLimit;
      layer.path = document.createElementNS(this.xmlns, 'path');
      this.svg.appendChild(layer.path);
    }
    this.wrapperElement = options.wrapperElement || document.body;
    if (this.svg.parentElement) {
      this.wrapperElement.append(this.svg);
    }

    this.svgText = document.createElementNS(this.xmlns, 'text');
    this.svgText.setAttribute('x', '50%');
    this.svgText.setAttribute('y', '50%');
    this.svgText.setAttribute('dy', `~~(this.height / 8) + 'px');
    this.svgText.setAttribute('font-size', `~~(this.height / 3)`);
    this.svgText.style.fontFamily = 'sans-serif';
    this.svgText.setAttribute('text-anchor', 'middle');
    this.svgText.setAttribute('pointer-events', 'none');
    this.svg.appendChild(this.svgText);

    this.svgDefs = document.createElementNS(this.xmlns, 'defs')
    this.svg.appendChild(this.svgDefs);

    this.touches = [];
    this.noise = options.noise || 0;
    document.body.addEventListener('touchstart', this.touchHandler);
    document.body.addEventListener('touchmove', this.touchHandler);
    document.body.addEventListener('touchend', this.clearHandler);
    document.body.addEventListener('touchcancel', this.clearHandler);
    this.svg.addEventListener('mousemove', this.mouseHandler);
    this.svg.addEventListener('mouseout', this.clearHandler);
    this.initOrigins();
    this.animate();
  }

  get mouseHandler() {
    return (e) => {
      this.touches = [{
        x: e.offsetX,
        y: e.offsetY,
        force: 1,
      }];
    }
  }

  get touchHandler() {
    return (e) => {
      this.touches = [];
      const rect = this.svg.getBoundingClientRect();
      for (let touchIndex = 0; touchIndex < e.changedTouches.length; touchIndex++)
{
        const touch = e.changedTouches[touchIndex];
        const x = touch.pageX - rect.left;
        const y = touch.pageY - rect.top;
        if (x > 0 && y > 0 && x < this.svgWidth && y < this.svgHeight) {
          this.touches.push({x, y, force: touch.force || 1});
        }
      }
      e.preventDefault();
    };
  }

  get clearHandler() {
    return (e) => {
      this.touches = [];
    };
  }

  get raf() {
    return this.__raf || (this.__raf = (
      window.requestAnimationFrame ||
      window.webkitRequestAnimationFrame ||
      window.mozRequestAnimationFrame ||
      function(callback){ setTimeout(callback, 10)}
    ).bind(window));
  }

  distance(p1, p2) {
    return Math.sqrt(Math.pow(p1.x - p2.x, 2) + Math.pow(p1.y - p2.y, 2));
  }

  update() {
    for (let layerIndex = 0; layerIndex < this.layers.length; layerIndex++) {
      const layer = this.layers[layerIndex];
      const points = layer.points;
      for (let pointIndex = 0; pointIndex < points.length; pointIndex++) {
        const point = points[pointIndex];
        const dx = point.ox - point.x + (Math.random() - 0.5) * this.noise;
        const dy = point.oy - point.y + (Math.random() - 0.5) * this.noise;
        const d = Math.sqrt(dx * dx + dy * dy);
        const f = d * this.forceFactor;
        point.vx += f * (dx / d) || 0;
        point.vy += f * (dy / d) || 0;
        for (let touchIndex = 0; touchIndex < this.touches.length; touchIndex++) {
          const touch = this.touches[touchIndex];
          let mouseForce = layer.mouseForce;
          if (
            touch.x > this.margin &&
            touch.x < this.margin + this.width &&
            touch.y > this.margin &&
            touch.y < this.margin + this.height
          ) {
            mouseForce *= -this.hoverFactor;
          }
          const mx = point.x - touch.x;
          const my = point.y - touch.y;
          const md = Math.sqrt(mx * mx + my * my);
          const mf = Math.max(-layer.forceLimit, Math.min(layer.forceLimit,
(mouseForce * touch.force) / md));
          point.vx += mf * (mx / md) || 0;
          point.vy += mf * (my / md) || 0;
        }
        point.vx *= layer.viscosity;
        point.vy *= layer.viscosity;
        point.x += point.vx;
        point.y += point.vy;
      }
      for (let pointIndex = 0; pointIndex < points.length; pointIndex++) {
        const prev = points[(pointIndex + points.length - 1) % points.length];
        const point = points[pointIndex];
        const next = points[(pointIndex + points.length + 1) % points.length];
        const dPrev = this.distance(point, prev);
        const dNext = this.distance(point, next);

        const line = {
          x: next.x - prev.x,
          y: next.y - prev.y,
        };
        const dLine = Math.sqrt(line.x * line.x + line.y * line.y);

        point.cPrev = {
          x: point.x - (line.x / dLine) * dPrev * this.tension,
          y: point.y - (line.y / dLine) * dPrev * this.tension,
        };
        point.cNext = {
          x: point.x + (line.x / dLine) * dNext * this.tension,
          y: point.y + (line.y / dLine) * dNext * this.tension,
        };
      }
    }
  }

  animate() {
    this.raf(() => {
      this.update();
      this.draw();
      this.animate();
    });
  }

  get svgWidth() {
    return this.width + this.margin * 2;
  }

  get svgHeight() {
    return this.height + this.margin * 2;
  }

  draw() {
    for (let layerIndex = 0; layerIndex < this.layers.length; layerIndex++) {
      const layer = this.layers[layerIndex];
      if (layerIndex === 1) {
        if (this.touches.length > 0) {
          while (this.svgDefs.firstChild) {
            this.svgDefs.removeChild(this.svgDefs.firstChild);
          }
          for (let touchIndex = 0; touchIndex < this.touches.length; touchIndex++) {
            const touch = this.touches[touchIndex];
            const gradient = document.createElementNS(this.xmlns, 'radialGradient');
            gradient.id = 'liquid-gradient-' + this.id + '-' + touchIndex;
            const start = document.createElementNS(this.xmlns, 'stop');
            start.setAttribute('stop-color', this.color3);
            start.setAttribute('offset', '0%');
            const stop = document.createElementNS(this.xmlns, 'stop');
            stop.setAttribute('stop-color', this.color2);
            stop.setAttribute('offset', '100%');
            gradient.appendChild(start);
            gradient.appendChild(stop);
            this.svgDefs.appendChild(gradient);
            gradient.setAttribute('cx', touch.x / this.svgWidth);
            gradient.setAttribute('cy', touch.y / this.svgHeight);
            gradient.setAttribute('r', touch.force);
            layer.path.style.fill = `url(#` + gradient.id + `)`;
          }
        } else {
          layer.path.style.fill = this.color2;
        }
      } else {
        layer.path.style.fill = this.color1;
      }
      const points = layer.points;
      const commands = [];
      commands.push(`M${ points[0].x, points[0].y};
      for (let pointIndex = 1; pointIndex < points.length; pointIndex += 1) {
        commands.push(`C${
          points[(pointIndex + 0) % points.length].cNext.x,
          points[(pointIndex + 0) % points.length].cNext.y,
          points[(pointIndex + 1) % points.length].cPrev.x,
          points[(pointIndex + 1) % points.length].cPrev.y,
          points[(pointIndex + 1) % points.length].x,
          points[(pointIndex + 1) % points.length].y
        `);
      }
      commands.push('Z');
      layer.path.setAttribute('d', commands.join(' '));
    }
    this.svgText.textContent = this.text;
    this.svgText.style.fill = this.textColor;
  }

  createPoint(x, y) {
    return {
      x: x,
      y: y,
      ox: x,
      oy: y,
      vx: 0,
      vy: 0,
    };
  }

  initOrigins() {
    this.svg.setAttribute('width', this.svgWidth);
    this.svg.setAttribute('height', this.svgHeight);
    for (let layerIndex = 0; layerIndex < this.layers.length; layerIndex++) {
      const layer = this.layers[layerIndex];
      const points = [];
      for (let x = ~~(this.height / 2); x < this.width - ~~(this.height / 2); x += this.gap)
{
        points.push(this.createPoint(
          x + this.margin,
          this.margin
        ));
      }
      for (let alpha = ~~(this.height * 1.25); alpha >= 0; alpha -= this.gap) {
        const angle = (Math.PI / ~~(this.height * 1.25)) * alpha;
        points.push(this.createPoint(
          Math.sin(angle) * this.height / 2 + this.margin + this.width - this.height / 2,
          Math.cos(angle) * this.height / 2 + this.margin + this.height / 2
        ));
      }
      for (let x = this.width - ~~(this.height / 2) - 1; x >= ~~(this.height / 2); x -=
this.gap) {
        points.push(this.createPoint(
          x + this.margin,
          this.margin + this.height
        ));
      }
      for (let alpha = 0; alpha <= ~~(this.height * 1.25); alpha += this.gap) {
        const angle = (Math.PI / ~~(this.height * 1.25)) * alpha;
        points.push(this.createPoint(
          (this.height - Math.sin(angle) * this.height / 2) + this.margin - this.height / 2,
          Math.cos(angle) * this.height / 2 + this.margin + this.height / 2
        ));
      }
      layer.points = points;
    }
  }
}


const redraw = () => {
  button.initOrigins();
};

const buttons = document.getElementsByClassName('btn0009_liquid');
for (let buttonIndex = 0; buttonIndex < buttons.length; buttonIndex++) {
  const button = buttons[buttonIndex];
  button.liquidButton = new LiquidButton(button);
}
</script>
```