SnowDog — Awwwards-Level Upgrade Brief

Critique (what’s missing to feel “high-end”)

• **Depth + cinematic lighting**: Right now it’s a flat 2D scale + CSS filters. There’s no parallax, no “camera,” no atmospheric perspective, and no convincing light transport (bloom, haze, rim-light).
• **Interactivity as spectacle**: The scene doesn’t *respond* to you. High-end interactive sites feel alive: cursor wind, drag momentum, subtle tilt, and micro-reactive UI.
• **Data → motion direction**: MCAP / buy pressure updates are “state changes,” not *events*. You want smoothing + anticipation + secondary motion (overshoot, decay) so price action feels physical.
• **Particle fidelity**: “SnowLayer” (DOM particles) rarely feels expensive. A shader/Canvas particle field with depth layers and motion blur instantly upgrades perceived production value.
• **Sound design**: No audio = no “game feel.” A minimal, tasteful soundscape (wind bed + sparkle ticks + pump bark) makes the experience feel premium and viral.
• **Art direction cohesion**: Background, dog, snow, and glow read like separate effects. You want a unified grade (tone mapping), consistent shadows, and a signature “SnowDog” visual motif (aurora / halo / ice caustics).

3 specific high-impact concepts (pick any 1–3)

Concept 1 — Shader-based snow field (depth layers) + cursor wind (BEST ROI)

A full-screen WebGL shader draws layered snow with parallax depth. **Buy pressure** controls density + speed. **Pump/dump** controls color temperature (warm gold vs cold blue). Cursor/touch acts as wind direction.

Concept 2 — Mouse parallax “diorama” + dog tilt + eye glint

Split the scene into 3–5 layers (BG, fog, snow, dog, foreground flakes). Use Framer Motion springs so the camera subtly *leans* into cursor. Add a small specular “eye glint” that tracks cursor — it’s cheap and looks insanely premium.

Concept 3 — Web Audio reactive soundscape (wind + sparkle + bark)

Web Audio API with a low-volume wind bed, sparse sparkle one-shots (scaled by buy pressure), and a tasteful “bark / chime” on pumps. Include a **mute toggle** + respect `prefers-reduced-motion` and user gesture requirements.

Code It (Chosen: Concept 1 — Shader Snow + Cursor Wind)

This is a **drop-in** implementation for your existing structure:

• Add `components/ShaderSnow.tsx`

• Update `components/SantaDogScene.tsx` to use it

1) New file: `components/ShaderSnow.tsx`

```
'use client';

import { useEffect, useMemo, useRef } from 'react';

type ShaderSnowProps = {
  /** 0..1 density/speed control */
  intensity: number;
  /** -1..1 (cursor wind X) */
  windX?: number;
  /** -1..1 (cursor wind Y) */
  windY?: number;
  /** warm/cold tint blend, -1 (cold) .. +1 (warm) */
  mood?: number;
  className?: string;
};

const VERT = `
attribute vec2 a_pos;
varying vec2 v_uv;
void main(){
  v_uv = a_pos * 0.5 + 0.5;
  gl_Position = vec4(a_pos, 0.0, 1.0);
}
`;

// Procedural layered snow with depth + cursor wind.
// No textures, no deps, just WebGL.
const FRAG = `
precision highp float;

uniform vec2 u_res;
uniform float u_time;
uniform float u_intensity;
uniform vec2 u_wind;
uniform float u_mood;

varying vec2 v_uv;

float hash21(vec2 p){
  p = fract(p*vec2(123.34, 456.21));
  p += dot(p, p+45.32);
  return fract(p.x*p.y);
}

float smoothCircle(vec2 p, float r){
  float d = length(p);
  return 1.0 - smoothstep(r, r*1.15, d);
}

vec3 grade(vec3 c, float mood){
  // mood: -1 cold, +1 warm
  vec3 cold = vec3(0.65, 0.85, 1.15);
  vec3 warm = vec3(1.20, 1.05, 0.70);
  vec3 tint = mix(cold, warm, clamp(mood*0.5+0.5, 0.0, 1.0));
  // gentle filmic-ish curve
  c *= tint;
  c = c / (c + vec3(1.0));
  c = pow(c, vec3(1.0/1.12));
  return c;
}

void main(){
  vec2 uv = v_uv;
  vec2 px = (uv * u_res);

  float t = u_time;
  float I = clamp(u_intensity, 0.0, 1.0);

  // Base alpha accumulation
  float a = 0.0;

  // Wind is subtle and scaled by intensity
  vec2 wind = u_wind * (0.25 + 0.65*I);

  // 3 depth layers
  for(int layer=0; layer<3; layer++){
    float z = float(layer); // 0 near, 2 far
    float depth = mix(1.25, 0.45, z/2.0);

    // Grid size varies by layer
    float cell = mix(55.0, 125.0, z/2.0);

    vec2 p = px;
    p += wind * u_res * (0.03 + 0.02*z);
    p.y += t * u_res.y * (0.10 + 0.08*I) * depth;
    p.x += t * u_res.x * (0.02 + 0.02*I) * depth;

    vec2 g = floor(p / cell);
    vec2 f = fract(p / cell) - 0.5;

    // 3x3 neighborhood for stable flakes
    for(int j=-1; j<=1; j++){
      for(int i=-1; i<=1; i++){
        vec2 gg = g + vec2(float(i), float(j));
        float rnd = hash21(gg);

        // Density per layer
        float spawn = step(1.0 - (0.12 + 0.20*I) * mix(1.0, 0.65, z/2.0), rnd);

        // Flake position inside cell
        vec2 off = vec2(hash21(gg + 12.3), hash21(gg + 78.9)) - 0.5;
        vec2 fp = f - vec2(float(i), float(j)) - off;

        // Size varies with randomness + depth
        float r = mix(0.07, 0.02, z/2.0) * mix(0.65, 1.35, rnd);

        // Slight streaking by wind (cheap motion blur)
        fp.x += wind.x * 0.18;
        fp.y += wind.y * 0.08;

        float flake = smoothCircle(fp, r);

        // Soft sparkle core
        float core = smoothCircle(fp, r*0.45) * 0.6;

        a += spawn * (flake + core) * (0.55 + 0.25*I) * mix(1.0, 0.55, z/2.0);
      }
    }
  }

  // Clamp and shape alpha
  a = 1.0 - exp(-a);
  a *= mix(0.55, 0.95, I);

  // Color: icy white with grade + mood
  vec3 col = vec3(0.92, 0.97, 1.0);
  col = grade(col, u_mood);

  // Subtle vignette so snow feels “in camera”
  vec2 q = uv - 0.5;
  float vig = smoothstep(0.95, 0.25, dot(q,q));
  a *= vig;

  gl_FragColor = vec4(col, a);
}
`;

function compile(gl: WebGLRenderingContext, type: number, src: string) {
  const s = gl.createShader(type);
  if (!s) throw new Error('shader alloc failed');
  gl.shaderSource(s, src);
  gl.compileShader(s);
  if (!gl.getShaderParameter(s, gl.COMPILE_STATUS)) {
    const msg = gl.getShaderInfoLog(s) || 'shader compile failed';
    gl.deleteShader(s);
    throw new Error(msg);
  }
  return s;
}

function link(gl: WebGLRenderingContext, vs: WebGLShader, fs: WebGLShader) {
  const p = gl.createProgram();
  if (!p) throw new Error('program alloc failed');
  gl.attachShader(p, vs);
  gl.attachShader(p, fs);
  gl.linkProgram(p);
  if (!gl.getProgramParameter(p, gl.LINK_STATUS)) {
    const msg = gl.getProgramInfoLog(p) || 'program link failed';
    gl.deleteProgram(p);
    throw new Error(msg);
  }
  return p;
}
```

```
export default function ShaderSnow({
  intensity,
  windX = 0,
  windY = 0,
  mood = 0,
  className,
}: ShaderSnowProps) {
  const canvasRef = useRef<HTMLCanvasElement | null>(null);
  const rafRef = useRef<number | null>(null);

  // Soft-clamp incoming values
  const params = useMemo(() => {
    return {
      intensity: Math.max(0, Math.min(1, intensity)),
      windX: Math.max(-1, Math.min(1, windX)),
      windY: Math.max(-1, Math.min(1, windY)),
      mood: Math.max(-1, Math.min(1, mood)),
    };
  }, [intensity, windX, windY, mood]);

  useEffect(() => {
    const canvas = canvasRef.current;
    if (!canvas) return;

    const gl = canvas.getContext('webgl', {
      alpha: true,
      antialias: false,
      premultipliedAlpha: true,
      powerPreference: 'high-performance',
    });

    if (!gl) return;

    let prog: WebGLProgram | null = null;
    let buf: WebGLBuffer | null = null;

    const u_res = { loc: null as WebGLUniformLocation | null };
    const u_time = { loc: null as WebGLUniformLocation | null };
    const u_intensity = { loc: null as WebGLUniformLocation | null };
    const u_wind = { loc: null as WebGLUniformLocation | null };
    const u_mood = { loc: null as WebGLUniformLocation | null };

    const start = performance.now();

    const resize = () => {
      const dpr = Math.min(2, window.devicePixelRatio || 1);
      const w = Math.max(1, Math.floor(canvas.clientWidth * dpr));
      const h = Math.max(1, Math.floor(canvas.clientHeight * dpr));
      if (canvas.width !== w || canvas.height !== h) {
        canvas.width = w;
        canvas.height = h;
        gl.viewport(0, 0, w, h);
      }
      gl.uniform2f(u_res.loc, canvas.width, canvas.height);
    };
```

```
try {
  const vs = compile(gl, gl.VERTEX_SHADER, VERT);
  const fs = compile(gl, gl.FRAGMENT_SHADER, FRAG);
  prog = link(gl, vs, fs);
  gl.deleteShader(vs);
  gl.deleteShader(fs);

  gl.useProgram(prog);

  // Full-screen quad
  buf = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, buf);
  gl.bufferData(
    gl.ARRAY_BUFFER,
    new Float32Array([
      -1, -1,
       1, -1,
      -1,  1,
      -1,  1,
       1, -1,
       1,  1,
    ]),
    gl.STATIC_DRAW
  );

  const a_pos = gl.getAttribLocation(prog, 'a_pos');
  gl.enableVertexAttribArray(a_pos);
  gl.vertexAttribPointer(a_pos, 2, gl.FLOAT, false, 0, 0);

  u_res.loc = gl.getUniformLocation(prog, 'u_res');
  u_time.loc = gl.getUniformLocation(prog, 'u_time');
  u_intensity.loc = gl.getUniformLocation(prog, 'u_intensity');
  u_wind.loc = gl.getUniformLocation(prog, 'u_wind');
  u_mood.loc = gl.getUniformLocation(prog, 'u_mood');

  // Blend over background
  gl.enable(gl.BLEND);
  gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);

  const onResize = () => resize();
  window.addEventListener('resize', onResize);

  const frame = () => {
    const now = performance.now();
    const t = (now - start) / 1000;

    resize();

    gl.clearColor(0, 0, 0, 0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.uniform1f(u_time.loc, t);
    gl.uniform1f(u_intensity.loc, params.intensity);
    gl.uniform2f(u_wind.loc, params.windX, params.windY);
    gl.uniform1f(u_mood.loc, params.mood);
```

```
        gl.drawArrays(gl.TRIANGLES, 0, 6);

        rafRef.current = requestAnimationFrame(frame);
      };

      rafRef.current = requestAnimationFrame(frame);

      return () => {
        window.removeEventListener('resize', onResize);
        if (rafRef.current) cancelAnimationFrame(rafRef.current);
        if (buf) gl.deleteBuffer(buf);
        if (prog) gl.deleteProgram(prog);
      };
    } catch {
      // If shader fails (rare), fail silently: keep site usable.
      return;
    }
  }, [params]);

  return (
    <canvas
      ref={canvasRef}
      className={className}
      style={{ width: '100%', height: '100%' }}
    />
  );
}
```

2) Update: `components/SantaDogScene.tsx` (replace SnowLayer with shader snow + cursor wind

```
'use client';

import { useEffect, useMemo, useState } from 'react';
import { motion, useMotionValue, useSpring, useTransform } from 'framer-motion';
import Image from 'next/image';
import { useSnowDog } from '../hooks/useSnowDog';
import { CONFIG } from '../lib/config';
import ShaderSnow from './ShaderSnow';

export default function SantaDogScene() {
  const { stats, loading } = useSnowDog();
  const [isMounted, setIsMounted] = useState(false);
  useEffect(() => setIsMounted(true), []);

  const isPumping = stats.change24h > 5;
  const isDumping = stats.change24h < -5;

  // Size Logic: $0=0.6x, $1M=2.5x
  const baseScale = Math.min(Math.max(0.6, 0.6 + (stats.mcap / 1_000_000) * 1.4), 2.5);

  // Cursor → camera drift (springy)
  const mx = useMotionValue(0); // -1..1
  const my = useMotionValue(0);
  const sx = useSpring(mx, { stiffness: 120, damping: 20, mass: 0.6 });
  const sy = useSpring(my, { stiffness: 120, damping: 20, mass: 0.6 });

  useEffect(() => {
    const onMove = (e: PointerEvent) => {
      const nx = (e.clientX / window.innerWidth) * 2 - 1;
      const ny = (e.clientY / window.innerHeight) * 2 - 1;
      mx.set(nx);
      my.set(ny);
    };
    window.addEventListener('pointermove', onMove, { passive: true });
    return () => window.removeEventListener('pointermove', onMove);
  }, [mx, my]);

  // Dog tilt + drift
  const dogX = useTransform(sx, (v) => v * 18);
  const dogY = useTransform(sy, (v) => v * 10);
  const rotY = useTransform(sx, (v) => v * 8);
  const rotX = useTransform(sy, (v) => -v * 6);

  // Snow intensity feels better with a curve
  const snowIntensity = useMemo(() => {
    const bp = stats.buyPressure ?? 0.5;
    // push mid-range higher, clamp 0.15..1
    const curved = Math.pow(Math.max(0, Math.min(1, bp)), 0.55);
    return Math.max(0.15, Math.min(1, curved));
  }, [stats.buyPressure]);

  const mood = isPumping ? 0.9 : isDumping ? -0.9 : 0.0;

  const dogVariants = {
    idle: {
      opacity: 1,
      scale: 1,
      y: 0,
      filter: 'drop-shadow(0 16px 18px rgba(0,0,0,0.35))',
      transition: { duration: 3.5, ease: 'easeInOut' },
    },
    barking: {
      opacity: 1,
      scale: 1,
      y: 0,
      filter: 'drop-shadow(0 0 60px rgba(255, 215, 0, 0.55)) brightness(1.06)',
      transition: { duration: 1.6, ease: 'easeInOut', repeat: Infinity, repeatType: 'reverse'
},
    },
    sad: {
      opacity: 1,
      y: 0,
      scale: 1,
      filter: 'grayscale(0.75) hue-rotate(180deg) brightness(0.92)',
      transition: { duration: 1.8, ease: 'easeInOut' },
    },
  };

  if (!isMounted) return null;
  return (
    <div className="relative w-full h-screen overflow-hidden bg-slate-950 text-white font-
mono">
      {/* Background */}
      <div className="absolute inset-0 z-0">
        <Image src="/new-bg.webp" alt="BG" fill className="object-cover opacity-80" priority />
        <div className="absolute inset-0 bg-black/35" />
      </div>

      {/* Shader Snow (depth, cursor wind) */}
      <div className="absolute inset-0 z-10 pointer-events-none">
        <ShaderSnow
          intensity={snowIntensity}
          windX={sx.get()}
          windY={sy.get()}
          mood={mood}
          className="absolute inset-0"
        />
      </div>

      {/* Dog */}
      <div className="relative z-20 w-full h-full flex items-center justify-center pt-10
[perspective:900px]">
        {!loading && (
          <motion.div
            animate={isPumping ? 'barking' : isDumping ? 'sad' : 'idle'}
            variants={dogVariants}
            className="relative will-change-transform"
            style={{
              width: 400 * baseScale,
              height: 400 * baseScale,
              x: dogX,
              y: dogY,
              rotateX: rotX,
              rotateY: rotY,
              transformStyle: 'preserve-3d',
            }}
          >
            <Image src={CONFIG.DOG_IMAGE} alt="Snow Dog" fill className="object-contain"
priority />
          </motion.div>
        )}
      </div>
    </div>
  );
}
```

> Note: If you want the shader to be perfectly reactive to Framer springs, pass `windX`/`windY` as state updated in `requestAnimationFrame` (or track pointer position in the shader component). The above version keeps the code simple.