```
<div class="codepen" data-height="300" data-default-tab="js,result"
data-slug-hash="YPWyrdW" data-pen-title="Ghost HEro" data-
editable="true" data-user="danilonovaisv"  data-
prefill='{"title":"Ghost HEro","tags":["cpc-
shadow","codepenchallenge","threejs","webgl","ghost"],"scripts":
[],"stylesheets":[]}'>
  <pre data-lang="html">&lt;!-- Preloader -->
&lt;div id="preloader" class="preloader">
  &lt;div class="preloader-content">
    &lt;div class="ghost-loader">
      &lt;svg class="ghost-svg" height="80" viewBox="0 0 512 512"
width="80" xmlns="http://www.w3.org/2000/svg">
        &lt;!-- Ghost body - white -->
        &lt;path class="ghost-body" d="m508.374
432.802s-46.6-39.038-79.495-275.781c-8.833-87.68-82.856-156.139-172.
879-156.139-90.015 0-164.046 68.458-172.879 156.138-32.895
236.743-79.495 275.782-79.495 275.782-15.107 25.181 20.733 28.178
38.699 27.94 35.254-.478 35.254 40.294 70.516 40.294 35.254 0
35.254-35.261 70.508-35.261s37.396 45.343 72.65 45.343 37.389-45.343
72.651-45.343c35.254 0 35.254 35.261 70.508 35.261s35.27-40.772
70.524-40.294c17.959.238 53.798-2.76 38.692-27.94z" fill="white" />
        &lt;!-- Left eye - black with pulsing animation -->
        &lt;circle class="ghost-eye left-eye" cx="208" cy="225"
r="22" fill="black" />
        &lt;!-- Right eye - black with pulsing animation -->
        &lt;circle class="ghost-eye right-eye" cx="297" cy="225"
r="22" fill="black" />
      &lt;/svg>
    &lt;/div>
    &lt;div class="loading-text">Summoning spirits&lt;/div>
    &lt;div class="loading-progress">
      &lt;div class="progress-bar">&lt;/div>
    &lt;/div>
  &lt;/div>
&lt;/div>

&lt;!-- Main Content (initially hidden) -->
&lt;div class="content" id="main-content">
  &lt;div class="quote-container">
    [BRAND AWARENESS]&lt;br />
    &lt;h1 class="quote">
      &lt;strong>Você não vê&lt;br />
        o design.&lt;/strong>&lt;br />
      &lt;h2 class="quote"> Mas ele vê você.&lt;br />
      &lt;/h2>
  &lt;/div>
&lt;/div></pre>
  <pre data-lang="css">@import url("https://fonts.googleapis.com/
css2?family=Roboto:ital,wght@0,100..900;1,100..900&display=swap");

@font-face {
  font-family: "Roboto", sans-serif;
  src: url("https://fonts.googleapis.com/css2?
family=Roboto:ital,wght@0,100..900;1,100..900&display=swap")
```

```
    format("woff2");
  font-optical-sizing: auto;
  font-weight: &lt;weight>;
  font-style: normal;
  font-variation-settings: "wdth" 300;
}

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

html,
body {
  width: 100%;
  height: 100%;
  overflow: hidden;
  background-color: #111;
  letter-spacing: -0.03em;
}

/* Preloader Styles */
.preloader {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: linear-gradient(135deg, #0a0a0a 0%, #1a1a1a 50%,
#0a0a0a 100%);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 10000;
  opacity: 1;
  transition: opacity 1s ease-out;
}

.preloader.fade-out {
  opacity: 0;
  pointer-events: none;
}

.preloader-content {
  text-align: center;
  color: #e0e0e0;
}

.ghost-loader {
  position: relative;
  width: 64px;
  height: 64px;
  margin: 0 auto 30px;
  display: flex;
  justify-content: center;
  align-items: center;
}

.ghost-svg {
  filter: drop-shadow(0 0 20px rgba(255, 255, 255, 0.3));
  animation: ghostFloat 3s ease-in-out infinite;
}

.ghost-body {
  fill: white;
  opacity: 0.9;
}

.ghost-eye {
  fill: black;
  animation: eyePulse 2s ease-in-out infinite;
  transform-origin: center;
}

.left-eye {
  animation-delay: 0s;
}

.right-eye {
  animation-delay: 0.1s;
}

@keyframes ghostFloat {
  0%,
  100% {
    transform: translateY(0px);
  }
  50% {
    transform: translateY(-8px);
  }
}

@keyframes eyePulse {
  0%,
  100% {
    transform: scale(1);
  }
  50% {
    transform: scale(1.3);
  }
}

/* Remove the old ghost-orb and ghost-trail styles */
.ghost-orb,
.ghost-trail {
  display: none;
}
```

```
.loading-text {
  font-family: "PPSupplyMono", monospace;
  font-size: 12px;
  text-transform: uppercase;
  opacity: 1;
  margin-bottom: 12px;
  animation: textPulse 2s ease-in-out infinite;
}

@keyframes textPulse {
  0%,
  100% {
    opacity: 1;
  }
  50% {
    opacity: 0.1;
  }
}

.loading-progress {
  width: 96px;
  height: 1px;
  margin: 0 auto;
  border-radius: 1px;
  overflow: hidden;
}

.progress-bar {
  height: 100%;
  background: linear-gradient(90deg, #00ff80, #00cc66);
  opacity: 0.1;
  width: 0%;
  transition: width 0.8s ease;
}

/* Main Content Styles */
.content {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding: 20px;
  text-align: center;
  color: #e0e0e0;
  opacity: 0;
  transition: opacity 1.5s ease-in;
}
```

```
.content.fade-in {
  opacity: 1;
}

.quote-container {
  max-width: 90%;
  overflow: hidden;
}

.quote {
  font-family: "Boldonse", system-ui;
  font-size: 6vw;
  line-height: 1.3;
  font-weight: 400;
  letter-spacing: -0.02em;
  margin-bottom: 5vh;
  text-transform: uppercase;
}

.author {
  font-family: "PPSupplyMono", monospace;
  font-size: 12px;
  text-transform: uppercase;
  opacity: 0.7;
  margin-top: 2vh;
}

/* Canvas initially hidden */
canvas {
  opacity: 0 !important;
  transition: opacity 2s ease-in;
}

canvas.fade-in {
  opacity: 1 !important;
}
</pre>
  <pre data-lang="typescript">import * as THREE from "https://
esm.sh/three";
import { Pane } from "https://cdn.skypack.dev/tweakpane@4.0.4";
import { EffectComposer } from "https://esm.sh/three/examples/jsm/
postprocessing/EffectComposer.js";
import { RenderPass } from "https://esm.sh/three/examples/jsm/
postprocessing/RenderPass.js";
import { UnrealBloomPass } from "https://esm.sh/three/examples/jsm/
postprocessing/UnrealBloomPass.js";
import { OutputPass } from "https://esm.sh/three/examples/jsm/
postprocessing/OutputPass.js";
import { ShaderPass } from "https://esm.sh/three/examples/jsm/
postprocessing/ShaderPass.js";

// Preloader management
class PreloaderManager {
  constructor() {
    this.preloader = document.getElementById("preloader");
    this.mainContent = document.getElementById("main-content");
    this.progressBar = document.querySelector(".progress-bar");
    this.loadingSteps = 0;
    this.totalSteps = 5; // Adjust based on loading steps
    this.isComplete = false;
  }

  updateProgress(step) {
    this.loadingSteps = Math.min(step, this.totalSteps);
    const percentage = (this.loadingSteps / this.totalSteps) * 100;
    this.progressBar.style.width = `${percentage}%`;
  }

  complete(canvas) {
    if (this.isComplete) return;
    this.isComplete = true;

    // Ensure we're at 100%
    this.updateProgress(this.totalSteps);

    // Wait a moment then start the reveal
    setTimeout(() => {
      // Fade out preloader
      this.preloader.classList.add("fade-out");

      // Fade in content and canvas simultaneously
      this.mainContent.classList.add("fade-in");
      canvas.classList.add("fade-in");

      // Remove preloader from DOM after animation
      setTimeout(() => {
        this.preloader.style.display = "none";
      }, 1000);
    }, 1500);
  }
}

// Initialize preloader
const preloader = new PreloaderManager();

// Force browser to use GPU acceleration
document.body.style.transform = "translateZ(0)";
document.body.style.backfaceVisibility = "hidden";
document.body.style.perspective = "1000px";

preloader.updateProgress(1);

// Create scene
const scene = new THREE.Scene();
scene.background = null;

const camera = new THREE.PerspectiveCamera(
  75,
  window.innerWidth / window.innerHeight,
  0.1,
  1000
);
camera.position.z = 20;

preloader.updateProgress(2);

// Enhanced renderer with transparency
const renderer = new THREE.WebGLRenderer({
  antialias: true,
  powerPreference: "high-performance",
  alpha: true,
  premultipliedAlpha: false,
  stencil: false,
  depth: true,
  preserveDrawingBuffer: false
});
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.toneMapping = THREE.ACESFilmicToneMapping;
renderer.toneMappingExposure = 0.9;
renderer.setClearColor(0x000000, 0);
document.body.appendChild(renderer.domElement);

// Canvas styling - initially hidden
renderer.domElement.style.position = "absolute";
renderer.domElement.style.top = "0";
renderer.domElement.style.left = "0";
renderer.domElement.style.zIndex = "2";
renderer.domElement.style.pointerEvents = "auto";
renderer.domElement.style.background = "transparent";

// Store original bloom values
const originalBloomSettings = {
  strength: 0.3,
  radius: 1.25,
  threshold: 0.0
};

// Setup post-processing for bloom effects
const composer = new EffectComposer(renderer);
const renderPass = new RenderPass(scene, camera);
composer.addPass(renderPass);

// Fixed bloom settings to avoid transparency issues
const bloomPass = new UnrealBloomPass(
  new THREE.Vector2(window.innerWidth, window.innerHeight),
  originalBloomSettings.strength,
  originalBloomSettings.radius,
  originalBloomSettings.threshold
);
composer.addPass(bloomPass);

preloader.updateProgress(3);
```

```
// Analog Decay Shader with simple black/white mode
const analogDecayShader = {
  uniforms: {
    tDiffuse: { value: null },
    uTime: { value: 0.0 },
    uResolution: {
      value: new THREE.Vector2(window.innerWidth,
window.innerHeight)
    },
    uAnalogGrain: { value: 0.4 },
    uAnalogBleeding: { value: 1.0 },
    uAnalogVSync: { value: 1.0 },
    uAnalogScanlines: { value: 1.0 },
    uAnalogVignette: { value: 1.0 },
    uAnalogJitter: { value: 0.4 },
    uAnalogIntensity: { value: 0.6 },
    uLimboMode: { value: 0.0 }
  },

  vertexShader: `
    varying vec2 vUv;
    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix *
vec4(position, 1.0);
    }
  `,

  fragmentShader: `
    uniform sampler2D tDiffuse;
    uniform float uTime;
    uniform vec2 uResolution;
    uniform float uAnalogGrain;
    uniform float uAnalogBleeding;
    uniform float uAnalogVSync;
    uniform float uAnalogScanlines;
    uniform float uAnalogVignette;
    uniform float uAnalogJitter;
    uniform float uAnalogIntensity;
    uniform float uLimboMode;

    varying vec2 vUv;

    float random(vec2 st) {
      return fract(sin(dot(st.xy, vec2(12.9898, 78.233))) *
43758.5453123);
    }

    float random(float x) {
      return fract(sin(x) * 43758.5453123);
    }

    // Advanced procedural grain based on film grain simulation
    float gaussian(float z, float u, float o) {
      return (1.0 / (o * sqrt(2.0 * 3.1415))) * exp(-(((z - u) * (z
- u)) / (2.0 * (o * o))));
    }

    vec3 grain(vec2 uv, float time, float intensity) {
      float seed = dot(uv, vec2(12.9898, 78.233));
      float noise = fract(sin(seed) * 43758.5453 + time * 2.0);
      noise = gaussian(noise, 0.0, 0.5 * 0.5);

      return vec3(noise) * intensity;
    }

    void main() {
      vec2 uv = vUv;
      float time = uTime * 1.8;

      // Analog Jitter - temporal instability
      vec2 jitteredUV = uv;
      if (uAnalogJitter > 0.01) {
        float jitterAmount = (random(vec2(floor(time * 60.0))) -
0.5) * 0.003 * uAnalogJitter * uAnalogIntensity;
        jitteredUV.x += jitterAmount;
        jitteredUV.y += (random(vec2(floor(time * 30.0) + 1.0)) -
0.5) * 0.001 * uAnalogJitter * uAnalogIntensity;
      }

      // VHS-style vertical sync roll
      if (uAnalogVSync > 0.01) {
        float vsyncRoll = sin(time * 2.0 + uv.y * 100.0) * 0.02 *
uAnalogVSync * uAnalogIntensity;
        float vsyncChance = step(0.95, random(vec2(floor(time *
4.0))));
        jitteredUV.y += vsyncRoll * vsyncChance;
      }

      vec4 color = texture2D(tDiffuse, jitteredUV);

      // Color bleeding/channel separation
      if (uAnalogBleeding > 0.01) {
        float bleedAmount = 0.012 * uAnalogBleeding *
uAnalogIntensity;
        float offsetPhase = time * 1.5 + uv.y * 20.0;

        vec2 redOffset = vec2(sin(offsetPhase) * bleedAmount, 0.0);
        vec2 blueOffset = vec2(-sin(offsetPhase * 1.1) * bleedAmount
* 0.8, 0.0);

        float r = texture2D(tDiffuse, jitteredUV + redOffset).r;
        float g = texture2D(tDiffuse, jitteredUV).g;
        float b = texture2D(tDiffuse, jitteredUV + blueOffset).b;

        color = vec4(r, g, b, color.a);
      }
      // Improved procedural film grain
      if (uAnalogGrain > 0.01) {
        vec3 grainEffect = grain(uv, time, 0.075 * uAnalogGrain *
uAnalogIntensity);
        grainEffect *= (1.0 - color.rgb);
        color.rgb += grainEffect;
      }

      // Scanlines
      if (uAnalogScanlines > 0.01) {
        float scanlineFreq = 600.0 + uAnalogScanlines * 400.0;
        float scanlinePattern = sin(uv.y * scanlineFreq) * 0.5 +
0.5;
        float scanlineIntensity = 0.1 * uAnalogScanlines *
uAnalogIntensity;
        color.rgb *= (1.0 - scanlinePattern * scanlineIntensity);

        float horizontalLines = sin(uv.y * scanlineFreq * 0.1) *
0.02 * uAnalogScanlines * uAnalogIntensity;
        color.rgb *= (1.0 - horizontalLines);
      }

      // Vignetting
      if (uAnalogVignette > 0.01) {
        vec2 vignetteUV = (uv - 0.5) * 2.0;
        float vignette = 1.0 - dot(vignetteUV, vignetteUV) * 0.3 *
uAnalogVignette * uAnalogIntensity;
        color.rgb *= vignette;
      }

      // Simple Limbo Mode (Black and White)
      if (uLimboMode > 0.5) {
        float gray = dot(color.rgb, vec3(0.299, 0.587, 0.114));
        color.rgb = vec3(gray);
      }

      gl_FragColor = color;
    }
  `
};

// Add analog decay pass
const analogDecayPass = new ShaderPass(analogDecayShader);
composer.addPass(analogDecayPass);

const outputPass = new OutputPass();
composer.addPass(outputPass);

// Production parameters with user's specified defaults
const params = {
  // Ghost appearance
  bodyColor: 0x0f2027,
  glowColor: "blue",
  eyeGlowColor: "violet",
  ghostOpacity: 0.88,
  ghostScale: 2.4,

  // Glow effects - updated to match screenshot
  emissiveIntensity: 5.8,
  pulseSpeed: 1.6,
  pulseIntensity: 0.6,

  // Eyes - updated to match screenshot
  eyeGlowIntensity: 4.5,
  eyeGlowDecay: 0.95,
  eyeGlowResponse: 0.31,

  // Enhanced lighting
  rimLightIntensity: 1.8,

  // Behavior - updated to match screenshot
  followSpeed: 0.05,
  wobbleAmount: 0.35,
  floatSpeed: 1.6,
  movementThreshold: 0.07,

  // Particles
  particleCount: 250,
  particleDecayRate: 0.005,
  particleColor: "violet",
  createParticlesOnlyWhenMoving: true,
  particleCreationRate: 5,

  // Background reveal - updated to match screenshot
  revealRadius: 37,
  fadeStrength: 1.7,
  baseOpacity: 0.9,
  revealOpacity: 0.05,

  // Fireflies
  fireflyGlowIntensity: 4.3,
  fireflySpeed: 0.09,

  // Analog Decay settings - updated to match screenshot
  analogIntensity: 0.9,
  analogGrain: 0.4,
  analogBleeding: 0.9,
  analogVSync: 1.7,
  analogScanlines: 1.0,
  analogVignette: 2.4,
  analogJitter: 0.5,
  limboMode: false
};

// Fluorescent color palette
const fluorescentColors = {
  cyan: 0x00ffff,
  lime: 0x00ff00,
  magenta: 0xff00ff,
  yellow: 0xffff00,
  orange: 0xff4500,
  pink: 0xff1493,
  purple: 0x9400d3,
  blue: 0x0080ff,
  green: 0x00ff80,
  red: 0xff0040,
  teal: 0x00ffaa,
  violet: 0x8a2be2
};

// Create bloom-resistant atmosphere
const atmosphereGeometry = new THREE.PlaneGeometry(300, 300);
const atmosphereMaterial = new THREE.ShaderMaterial({
  uniforms: {
    ghostPosition: { value: new THREE.Vector3(0, 0, 0) },
    revealRadius: { value: params.revealRadius },
    fadeStrength: { value: params.fadeStrength },
    baseOpacity: { value: params.baseOpacity },
    revealOpacity: { value: params.revealOpacity },
    time: { value: 0 }
  },
  vertexShader: `
    varying vec2 vUv;
    varying vec3 vWorldPosition;
    void main() {
      vUv = uv;
      vec4 worldPos = modelMatrix * vec4(position, 1.0);
      vWorldPosition = worldPos.xyz;
      gl_Position = projectionMatrix * modelViewMatrix *
vec4(position, 1.0);
    }
  `,
  fragmentShader: `
    uniform vec3 ghostPosition;
    uniform float revealRadius;
    uniform float fadeStrength;
    uniform float baseOpacity;
    uniform float revealOpacity;
    uniform float time;
    varying vec2 vUv;
    varying vec3 vWorldPosition;

    void main() {
      float dist = distance(vWorldPosition.xy, ghostPosition.xy);

      // Pulsing reveal radius
      float dynamicRadius = revealRadius + sin(time * 2.0) * 5.0;

      // Create smooth reveal gradient
      float reveal = smoothstep(dynamicRadius * 0.2, dynamicRadius,
dist);
```

```
      reveal = pow(reveal, fadeStrength);

      // Mix between revealed and base opacity
      float opacity = mix(revealOpacity, baseOpacity, reveal);

      // EXTREMELY low RGB values to avoid bloom
      gl_FragColor = vec4(0.001, 0.001, 0.002, opacity);
    }
  `,
  transparent: true,
  depthWrite: false
});

const atmosphere = new THREE.Mesh(atmosphereGeometry,
atmosphereMaterial);
atmosphere.position.z = -50;
atmosphere.renderOrder = -100;
scene.add(atmosphere);

// Minimal ambient light
const ambientLight = new THREE.AmbientLight(0x0a0a2e, 0.08);
scene.add(ambientLight);

// Create ghost group
const ghostGroup = new THREE.Group();
scene.add(ghostGroup);

// Enhanced ghost geometry
const ghostGeometry = new THREE.SphereGeometry(2, 40, 40);

// Create organic wavy bottom
const positionAttribute = ghostGeometry.getAttribute("position");
const positions = positionAttribute.array;
for (let i = 0; i &lt; positions.length; i += 3) {
  if (positions[i + 1] &lt; -0.2) {
    const x = positions[i];
    const z = positions[i + 2];
    const noise1 = Math.sin(x * 5) * 0.35;
    const noise2 = Math.cos(z * 4) * 0.25;
    const noise3 = Math.sin((x + z) * 3) * 0.15;
    const combinedNoise = noise1 + noise2 + noise3;
    positions[i + 1] = -2.0 + combinedNoise;
  }
}
ghostGeometry.computeVertexNormals();

// Ghost material
const ghostMaterial = new THREE.MeshStandardMaterial({
  color: params.bodyColor,
  transparent: true,
  opacity: params.ghostOpacity,
  emissive: fluorescentColors[params.glowColor],
  emissiveIntensity: params.emissiveIntensity,
  roughness: 0.02,
  metalness: 0.0,
  side: THREE.DoubleSide,
  alphaTest: 0.1
});

const ghostBody = new THREE.Mesh(ghostGeometry, ghostMaterial);
ghostGroup.add(ghostBody);

// Rim lights
const rimLight1 = new THREE.DirectionalLight(
  0x4a90e2,
  params.rimLightIntensity
);
rimLight1.position.set(-8, 6, -4);
scene.add(rimLight1);

const rimLight2 = new THREE.DirectionalLight(
  0x50e3c2,
  params.rimLightIntensity * 0.7
);
rimLight2.position.set(8, -4, -6);
scene.add(rimLight2);

preloader.updateProgress(4);

// Improved eyes function - 50% bigger eyes
function createEyes() {
  const eyeGroup = new THREE.Group();
  ghostGroup.add(eyeGroup);

  // Create deeper, more realistic eye sockets
  const socketGeometry = new THREE.SphereGeometry(0.45, 16, 16);
  const socketMaterial = new THREE.MeshBasicMaterial({
    color: 0x000000,
    transparent: false
  });

  // Left eye socket - positioned better
  const leftSocket = new THREE.Mesh(socketGeometry, socketMaterial);
  leftSocket.position.set(-0.7, 0.6, 1.9);
  leftSocket.scale.set(1.1, 1.0, 0.6);
  eyeGroup.add(leftSocket);

  // Right eye socket
  const rightSocket = new THREE.Mesh(socketGeometry,
socketMaterial);
  rightSocket.position.set(0.7, 0.6, 1.9);
  rightSocket.scale.set(1.1, 1.0, 0.6);
  eyeGroup.add(rightSocket);

  // Create bigger glowing eyes (50% bigger: 0.2 * 1.5 = 0.3)
  const eyeGeometry = new THREE.SphereGeometry(0.3, 12, 12);

  // Left eye glow - starts invisible
  const leftEyeMaterial = new THREE.MeshBasicMaterial({
    color: fluorescentColors[params.eyeGlowColor],
    transparent: true,
    opacity: 0
  });
  const leftEye = new THREE.Mesh(eyeGeometry, leftEyeMaterial);
  leftEye.position.set(-0.7, 0.6, 2.0);
  eyeGroup.add(leftEye);

  // Right eye glow
  const rightEyeMaterial = new THREE.MeshBasicMaterial({
    color: fluorescentColors[params.eyeGlowColor],
    transparent: true,
    opacity: 0
  });
  const rightEye = new THREE.Mesh(eyeGeometry, rightEyeMaterial);
  rightEye.position.set(0.7, 0.6, 2.0);
  eyeGroup.add(rightEye);

  // Add subtle outer glow for each eye (also 50% bigger: 0.35 * 1.5
= 0.525)
  const outerGlowGeometry = new THREE.SphereGeometry(0.525, 12, 12);

  const leftOuterGlowMaterial = new THREE.MeshBasicMaterial({
    color: fluorescentColors[params.eyeGlowColor],
    transparent: true,
    opacity: 0,
    side: THREE.BackSide
  });
  const leftOuterGlow = new THREE.Mesh(
    outerGlowGeometry,
    leftOuterGlowMaterial
  );
  leftOuterGlow.position.set(-0.7, 0.6, 1.95);
  eyeGroup.add(leftOuterGlow);

  const rightOuterGlowMaterial = new THREE.MeshBasicMaterial({
    color: fluorescentColors[params.eyeGlowColor],
    transparent: true,
    opacity: 0,
    side: THREE.BackSide
  });
  const rightOuterGlow = new THREE.Mesh(
    outerGlowGeometry,
    rightOuterGlowMaterial
  );
  rightOuterGlow.position.set(0.7, 0.6, 1.95);
  eyeGroup.add(rightOuterGlow);

  return {
    leftEye,
    rightEye,
    leftEyeMaterial,
    rightEyeMaterial,
    leftOuterGlow,
    rightOuterGlow,
    leftOuterGlowMaterial,
    rightOuterGlowMaterial
  };
}

const eyes = createEyes();

// Create fireflies with enhanced visibility
const fireflies = [];
const fireflyGroup = new THREE.Group();
scene.add(fireflyGroup);

function createFireflies() {
  for (let i = 0; i &lt; 20; i++) {
    // Create bright yellow firefly core
    const fireflyGeometry = new THREE.SphereGeometry(0.02, 2, 2);
    const fireflyMaterial = new THREE.MeshBasicMaterial({
      color: 0xffff44,
      transparent: true,
      opacity: 0.9
    });

    const firefly = new THREE.Mesh(fireflyGeometry,
fireflyMaterial);

    // Random starting position
    firefly.position.set(
      (Math.random() - 0.5) * 40,
      (Math.random() - 0.5) * 30,
      (Math.random() - 0.5) * 20
    );

    // Create visible glow around firefly
    const glowGeometry = new THREE.SphereGeometry(0.08, 8, 8);
    const glowMaterial = new THREE.MeshBasicMaterial({
      color: 0xffff88,
      transparent: true,
      opacity: 0.4,
      side: THREE.BackSide
    });

    const glow = new THREE.Mesh(glowGeometry, glowMaterial);
    firefly.add(glow);

    const fireflyLight = new THREE.PointLight(0xffff44, 0.8, 3, 2);
    firefly.add(fireflyLight);

    // Store movement data
    firefly.userData = {
      velocity: new THREE.Vector3(
        (Math.random() - 0.5) * params.fireflySpeed,
        (Math.random() - 0.5) * params.fireflySpeed,
        (Math.random() - 0.5) * params.fireflySpeed
      ),
      basePosition: firefly.position.clone(),
      phase: Math.random() * Math.PI * 2,
      pulseSpeed: 2 + Math.random() * 3,
      glow: glow,
      glowMaterial: glowMaterial,
      fireflyMaterial: fireflyMaterial,
      light: fireflyLight
    };

    fireflyGroup.add(firefly);
    fireflies.push(firefly);
  }
}

createFireflies();

// Particle system
const particles = [];
const particleGroup = new THREE.Group();
scene.add(particleGroup);

const particlePool = [];
const particleGeometries = [
  new THREE.SphereGeometry(0.05, 6, 6),
  new THREE.TetrahedronGeometry(0.04, 0),
  new THREE.OctahedronGeometry(0.045, 0)
];

const particleBaseMaterial = new THREE.MeshBasicMaterial({
  color: fluorescentColors[params.particleColor],
  transparent: true,
  opacity: 0,
  alphaTest: 0.1
});

function initParticlePool(count) {
  for (let i = 0; i &lt; count; i++) {
    const geomIndex = Math.floor(Math.random() *
particleGeometries.length);
    const geometry = particleGeometries[geomIndex];
    const material = particleBaseMaterial.clone();
    const particle = new THREE.Mesh(geometry, material);
    particle.visible = false;
    particleGroup.add(particle);
    particlePool.push(particle);
  }
}

initParticlePool(100);

function createParticle() {
  let particle;
  if (particlePool.length > 0) {
    particle = particlePool.pop();
    particle.visible = true;
  } else if (particles.length &lt; params.particleCount) {
    const geomIndex = Math.floor(Math.random() *
particleGeometries.length);
    const geometry = particleGeometries[geomIndex];
    const material = particleBaseMaterial.clone();
    particle = new THREE.Mesh(geometry, material);
    particleGroup.add(particle);
  } else {
    return null;
  }

  const particleColor = new THREE.Color(
    fluorescentColors[params.particleColor]
  );
  const hue = Math.random() * 0.1 - 0.05;
  particleColor.offsetHSL(hue, 0, 0);
  particle.material.color = particleColor;

  particle.position.copy(ghostGroup.position);
  particle.position.z -= 0.8 + Math.random() * 0.6;

  const scatterRange = 3.5;
  particle.position.x += (Math.random() - 0.5) * scatterRange;
  particle.position.y += (Math.random() - 0.5) * scatterRange - 0.8;

  const sizeVariation = 0.6 + Math.random() * 0.7;
  particle.scale.set(sizeVariation, sizeVariation, sizeVariation);

  particle.rotation.set(
    Math.random() * Math.PI * 2,
    Math.random() * Math.PI * 2,
    Math.random() * Math.PI * 2
  );

  particle.userData.life = 1.0;
  particle.userData.decay = Math.random() * 0.003 +
params.particleDecayRate;
  particle.userData.rotationSpeed = {
    x: (Math.random() - 0.5) * 0.015,
    y: (Math.random() - 0.5) * 0.015,
    z: (Math.random() - 0.5) * 0.015
  };
  particle.userData.velocity = {
    x: (Math.random() - 0.5) * 0.012,
    y: (Math.random() - 0.5) * 0.012 - 0.002,
    z: (Math.random() - 0.5) * 0.012 - 0.006
  };

  particle.material.opacity = Math.random() * 0.9;
  particles.push(particle);
  return particle;
```

```
}

// Enhanced GUI
const pane = new Pane({
  title: "Spectral Ghost",
  expanded: false
});

const paneElement = pane.element;
paneElement.style.position = "fixed";
paneElement.style.top = "20px";
paneElement.style.right = "20px";
paneElement.style.zIndex = "10000";
paneElement.style.backgroundColor = "rgba(0, 0, 0, 0.9)";
paneElement.style.borderRadius = "12px";
paneElement.style.padding = "15px";
paneElement.style.backdropFilter = "blur(10px)";
paneElement.style.border = "1px solid rgba(0, 212, 255, 0.3)";
paneElement.style.pointerEvents = "auto";

// Glow effects folder
const glowFolder = pane.addFolder({
  title: "Glow Effects",
  expanded: true
});

const glowColorBinding = glowFolder
  .addBinding(params, "glowColor", {
    label: "Glow Color",
    options: {
      Cyan: "cyan",
      Lime: "lime",
      Magenta: "magenta",
      Yellow: "yellow",
      Orange: "orange",
      Pink: "pink",
      Purple: "purple",
      Blue: "blue",
      Green: "green",
      Red: "red",
      Teal: "teal",
      Violet: "violet"
    }
  })
  .on("change", (ev) => {
    const color = fluorescentColors[ev.value];
    ghostMaterial.emissive.set(color);
  });

glowFolder
  .addBinding(params, "emissiveIntensity", {
    label: "Ghost Glow",
    min: 1.0,
    max: 10.0,
    step: 0.1
  })
  .on("change", (ev) => {
    ghostMaterial.emissiveIntensity = ev.value;
  });

// Eye controls folder
const eyeFolder = pane.addFolder({
  title: "Eye Controls",
  expanded: true
});

// Fixed eye glow color picker
eyeFolder
  .addBinding(params, "eyeGlowColor", {
    label: "Eye Glow Color",
    options: {
      Cyan: "cyan",
      Lime: "lime",
      Magenta: "magenta",
      Yellow: "yellow",
      Orange: "orange",
      Pink: "pink",
      Purple: "purple",
      Blue: "blue",
      Green: "green",
      Red: "red",
      Teal: "teal",
      Violet: "violet"
    }
  })
  .on("change", (ev) => {
    const color = fluorescentColors[ev.value];
    eyes.leftEyeMaterial.color.set(color);
    eyes.rightEyeMaterial.color.set(color);
    eyes.leftOuterGlowMaterial.color.set(color);
    eyes.rightOuterGlowMaterial.color.set(color);
  });

eyeFolder.addBinding(params, "eyeGlowDecay", {
  label: "Glow Fade Speed",
  min: 0.9,
  max: 0.99,
  step: 0.01
});

eyeFolder.addBinding(params, "eyeGlowResponse", {
  label: "Glow Response",
  min: 0.05,
  max: 0.5,
  step: 0.01
});

eyeFolder.addBinding(params, "movementThreshold", {
  label: "Movement Threshold",
  min: 0.01,
  max: 0.1,
  step: 0.01
});

// Background Reveal folder
const revealFolder = pane.addFolder({
  title: "Background Reveal",
  expanded: true
});

revealFolder
  .addBinding(params, "revealRadius", {
    label: "Reveal Radius",
    min: 5,
    max: 100,
    step: 2
  })
  .on("change", (ev) => {
    atmosphereMaterial.uniforms.revealRadius.value = ev.value;
  });

revealFolder
  .addBinding(params, "fadeStrength", {
    label: "Fade Strength",
    min: 0.1,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    atmosphereMaterial.uniforms.fadeStrength.value = ev.value;
  });

revealFolder
  .addBinding(params, "baseOpacity", {
    label: "Base Darkness",
    min: 0,
    max: 1,
    step: 0.05
  })
  .on("change", (ev) => {
    atmosphereMaterial.uniforms.baseOpacity.value = ev.value;
  });

revealFolder
  .addBinding(params, "revealOpacity", {
    label: "Revealed Opacity",
    min: 0,
    max: 0.5,
    step: 0.01
  })
  .on("change", (ev) => {
    atmosphereMaterial.uniforms.revealOpacity.value = ev.value;
  });

// Fireflies folder
const firefliesFolder = pane.addFolder({
  title: "Fireflies",
  expanded: false
});

firefliesFolder
  .addBinding(params, "fireflyGlowIntensity", {
    label: "Firefly Glow",
    min: 0,
    max: 5,
    step: 0.1
  })
  .on("change", (ev) => {
    fireflies.forEach((firefly) => {
      firefly.userData.glowMaterial.opacity = ev.value * 0.4;
      firefly.userData.fireflyMaterial.opacity = ev.value * 0.9;
      firefly.userData.light.intensity = ev.value * 0.8;
    });
  });

firefliesFolder.addBinding(params, "fireflySpeed", {
  label: "Firefly Speed",
  min: 0.005,
  max: 0.1,
  step: 0.005
});

// Analog Decay folder
const analogFolder = pane.addFolder({
  title: "Analog Decay",
  expanded: true
});

analogFolder
  .addBinding(params, "limboMode", {
    label: "Limbo"
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uLimboMode.value = ev.value ? 1.0 :
0.0;
  });

analogFolder
  .addBinding(params, "analogIntensity", {
    label: "Overall Intensity",
    min: 0,
    max: 2,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogIntensity.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogGrain", {
    label: "Film Grain",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogGrain.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogBleeding", {
    label: "Color Bleeding",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogBleeding.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogVSync", {
    label: "VSync Roll",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogVSync.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogScanlines", {
    label: "Scanlines",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogScanlines.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogVignette", {
    label: "Vignetting",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogVignette.value = ev.value;
  });

analogFolder
  .addBinding(params, "analogJitter", {
    label: "Temporal Jitter",
    min: 0,
    max: 3,
    step: 0.1
  })
  .on("change", (ev) => {
    analogDecayPass.uniforms.uAnalogJitter.value = ev.value;
  });

// Behavior folder
const behaviorFolder = pane.addFolder({
  title: "Behavior",
  expanded: false
});

behaviorFolder.addBinding(params, "followSpeed", {
  label: "Follow Speed",
  min: 0.01,
  max: 0.2,
  step: 0.005
});

behaviorFolder.addBinding(params, "wobbleAmount", {
  label: "Wobble",
  min: 0,
  max: 1,
  step: 0.05
});

// Particles folder
const particlesFolder = pane.addFolder({
  title: "Particles",
  expanded: false
});

particlesFolder
  .addBinding(params, "particleColor", {
    label: "Particle Color",
    options: {
      Cyan: "cyan",
      Lime: "lime",
      Magenta: "magenta",
      Yellow: "yellow",
      Orange: "orange",
      Pink: "pink",
      Purple: "purple",
      Blue: "blue",
      Green: "green",
      Red: "red",
      Teal: "teal",
      Violet: "violet"
    }
  })
  .on("change", (ev) => {
    const color = fluorescentColors[ev.value];
    particleBaseMaterial.color.set(color);
  });

particlesFolder.addBinding(params, "createParticlesOnlyWhenMoving",
{
  label: "Only When Moving"
});

particlesFolder.addBinding(params, "particleCount", {
  label: "Particle Count",
  min: 50,
  max: 400,
  step: 10
});

// Window resize handler
let resizeTimeout;
window.addEventListener("resize", () => {
  if (resizeTimeout) clearTimeout(resizeTimeout);
  resizeTimeout = setTimeout(() => {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
    composer.setSize(window.innerWidth, window.innerHeight);

    bloomPass.setSize(window.innerWidth, window.innerHeight);
    analogDecayPass.uniforms.uResolution.value.set(
      window.innerWidth,
      window.innerHeight
    );
  }, 250);
});

// Mouse tracking
const mouse = new THREE.Vector2();
const prevMouse = new THREE.Vector2();
const mouseSpeed = new THREE.Vector2();
let lastMouseUpdate = 0;
let isMouseMoving = false;
let mouseMovementTimer = null;

window.addEventListener("mousemove", (e) => {
  const now = performance.now();
  if (now - lastMouseUpdate > 16) {
    prevMouse.x = mouse.x;
    prevMouse.y = mouse.y;
    mouse.x = (e.clientX / window.innerWidth) * 2 - 1;
    mouse.y = -(e.clientY / window.innerHeight) * 2 + 1;
    mouseSpeed.x = mouse.x - prevMouse.x;
    mouseSpeed.y = mouse.y - prevMouse.y;
    isMouseMoving = true;

    if (mouseMovementTimer) {
      clearTimeout(mouseMovementTimer);
    }
    mouseMovementTimer = setTimeout(() => {
      isMouseMoving = false;
    }, 80);

    lastMouseUpdate = now;
  }
});

// Animation loop
let lastParticleTime = 0;
let time = 0;
let currentMovement = 0;
let lastFrameTime = 0;
let isInitialized = false;
let frameCount = 0;

function forceInitialRender() {
  for (let i = 0; i &lt; 3; i++) {
    composer.render();
  }

  for (let i = 0; i &lt; 10; i++) {
    createParticle();
  }
  composer.render();
  isInitialized = true;

  // Complete the preloader once everything is ready
  preloader.complete(renderer.domElement);
}

// Complete loading step 5 and initialize
preloader.updateProgress(5);
setTimeout(forceInitialRender, 100);

function animate(timestamp) {
  requestAnimationFrame(animate);
  if (!isInitialized) return;

  const deltaTime = timestamp - lastFrameTime;
  lastFrameTime = timestamp;
  if (deltaTime > 100) return;

  const timeIncrement = (deltaTime / 16.67) * 0.01;
  time += timeIncrement;
  frameCount++;
  // Update shader times
  atmosphereMaterial.uniforms.time.value = time;
  analogDecayPass.uniforms.uTime.value = time;
  analogDecayPass.uniforms.uLimboMode.value = params.limboMode ?
1.0 : 0.0;

  // Ghost movement
  const targetX = mouse.x * 11;
  const targetY = mouse.y * 7;
  const prevGhostPosition = ghostGroup.position.clone();

  ghostGroup.position.x +=
    (targetX - ghostGroup.position.x) * params.followSpeed;
  ghostGroup.position.y +=
    (targetY - ghostGroup.position.y) * params.followSpeed;

  // Update atmosphere reveal position

atmosphereMaterial.uniforms.ghostPosition.value.copy(ghostGroup.posi
tion);

  const movementAmount =
prevGhostPosition.distanceTo(ghostGroup.position);
  currentMovement =
    currentMovement * params.eyeGlowDecay +
    movementAmount * (1 - params.eyeGlowDecay);

  // Floating animation
  const float1 = Math.sin(time * params.floatSpeed * 1.5) * 0.03;
  const float2 = Math.cos(time * params.floatSpeed * 0.7) * 0.018;
  const float3 = Math.sin(time * params.floatSpeed * 2.3) * 0.008;
  ghostGroup.position.y += float1 + float2 + float3;

  // Pulsing effects
  const pulse1 = Math.sin(time * params.pulseSpeed) *
params.pulseIntensity;
  const pulse2 =
    Math.cos(time * params.pulseSpeed * 1.4) * params.pulseIntensity
* 0.6;
  const breathe = Math.sin(time * 0.6) * 0.12;

  ghostMaterial.emissiveIntensity = params.emissiveIntensity +
pulse1 + breathe;

  // Update fireflies with enhanced visibility
  fireflies.forEach((firefly, index) => {
    const userData = firefly.userData;

    // Pulsing glow effect
    const pulsePhase = time + userData.phase;
    const pulse = Math.sin(pulsePhase * userData.pulseSpeed) * 0.4 +
0.6;

    userData.glowMaterial.opacity = params.fireflyGlowIntensity *
0.4 * pulse;
    userData.fireflyMaterial.opacity =
      params.fireflyGlowIntensity * 0.9 * pulse;
    userData.light.intensity = params.fireflyGlowIntensity * 0.8 *
pulse;

    // Random movement
    userData.velocity.x += (Math.random() - 0.5) * 0.001;
    userData.velocity.y += (Math.random() - 0.5) * 0.001;
    userData.velocity.z += (Math.random() - 0.5) * 0.001;

    // Limit velocity
    userData.velocity.clampLength(0, params.fireflySpeed);

    // Update position
    firefly.position.add(userData.velocity);

    // Keep fireflies in bounds
    if (Math.abs(firefly.position.x) > 30) userData.velocity.x *=
-0.5;
    if (Math.abs(firefly.position.y) > 20) userData.velocity.y *=
-0.5;
    if (Math.abs(firefly.position.z) > 15) userData.velocity.z *=
-0.5;
  });

  // Body animations
  const mouseDirection = new THREE.Vector2(
    targetX - ghostGroup.position.x,
    targetY - ghostGroup.position.y
  ).normalize();

  const tiltStrength = 0.1 * params.wobbleAmount;
  const tiltDecay = 0.95;
  ghostBody.rotation.z =
    ghostBody.rotation.z * tiltDecay +
    -mouseDirection.x * tiltStrength * (1 - tiltDecay);
  ghostBody.rotation.x =
    ghostBody.rotation.x * tiltDecay +
    mouseDirection.y * tiltStrength * (1 - tiltDecay);
  ghostBody.rotation.y = Math.sin(time * 1.4) * 0.05 *
params.wobbleAmount;

  // Scale variations
  const scaleVariation =
    1 + Math.sin(time * 2.1) * 0.025 * params.wobbleAmount + pulse1
* 0.015;
  const scaleBreath = 1 + Math.sin(time * 0.8) * 0.012;
  const finalScale = scaleVariation * scaleBreath;
  ghostBody.scale.set(finalScale, finalScale, finalScale);

  // Improved eye glow animation
  const normalizedMouseSpeed =
    Math.sqrt(mouseSpeed.x * mouseSpeed.x + mouseSpeed.y *
mouseSpeed.y) * 8;
  const isMoving = currentMovement > params.movementThreshold;
  const targetGlow = isMoving ? 1.0 : 0.0;

  // Gradually change eye glow
  const glowChangeSpeed = isMoving
    ? params.eyeGlowResponse * 2
    : params.eyeGlowResponse;

  // Update both inner eye and outer glow
  const newOpacity =
    eyes.leftEyeMaterial.opacity +
    (targetGlow - eyes.leftEyeMaterial.opacity) * glowChangeSpeed;

  eyes.leftEyeMaterial.opacity = newOpacity;
  eyes.rightEyeMaterial.opacity = newOpacity;
  eyes.leftOuterGlowMaterial.opacity = newOpacity * 0.3;
  eyes.rightOuterGlowMaterial.opacity = newOpacity * 0.3;

  // Particle creation
  const shouldCreateParticles = params.createParticlesOnlyWhenMoving
    ? currentMovement > 0.005 && isMouseMoving
    : currentMovement > 0.005;

  if (shouldCreateParticles && timestamp - lastParticleTime > 100) {
    const speedRate = Math.floor(normalizedMouseSpeed * 3);
    const particleRate = Math.min(
      params.particleCreationRate,
      Math.max(1, speedRate)
    );
    for (let i = 0; i &lt; particleRate; i++) {
      createParticle();
    }
    lastParticleTime = timestamp;
  }

  // Particle updates
  const particlesToUpdate = Math.min(particles.length, 60);
  for (let i = 0; i &lt; particlesToUpdate; i++) {
    const index = (frameCount + i) % particles.length;
    if (index &lt; particles.length) {
      const particle = particles[index];
      particle.userData.life -= particle.userData.decay;
      particle.material.opacity = particle.userData.life * 0.85;

      if (particle.userData.velocity) {
        particle.position.x += particle.userData.velocity.x;
        particle.position.y += particle.userData.velocity.y;
        particle.position.z += particle.userData.velocity.z;

        const swirl = Math.cos(time * 1.8 + particle.position.y) *
0.0008;
        particle.position.x += swirl;
      }
      if (particle.userData.rotationSpeed) {
        particle.rotation.x += particle.userData.rotationSpeed.x;
        particle.rotation.y += particle.userData.rotationSpeed.y;
        particle.rotation.z += particle.userData.rotationSpeed.z;
      }

      if (particle.userData.life &lt;= 0) {
        particle.visible = false;
        particle.material.opacity = 0;
        particlePool.push(particle);
        particles.splice(index, 1);
        i--;
      }
    }
  }

  // Render with analog decay effect
  composer.render();
}

// Initialize
const fakeEvent = new MouseEvent("mousemove", {
  clientX: window.innerWidth / 2,
  clientY: window.innerHeight / 2
});
window.dispatchEvent(fakeEvent);

animate(0);
</pre></div>
<script async src="https://public.codepenassets.com/embed/
index.js"></script>
```