明白了。抛弃讲座表演的背景，我们完全聚焦于**“Web交互艺术”**本身。

你提出的这几个核心意象——**“冷暖并存”、“边界模糊”、“靠近即消散”、“疏离感”**——非常适合用**粒子系统（Particle System）结合交互式模糊算法（Interactive Blur）**来实现。

在这个方案中，屏幕上的图像并不稳固，它像是一层浮在水面上的油彩，或者透过满是雾气的玻璃看到的远处灯火。

项目名称：《彼岸之火 **(The Unreachable Light)**》

---

1. Task (任务定义)

核心任务：

构建一个基于浏览器的视觉寓言。创造一个视觉上极具诱惑力（暖光在冷调中闪烁）的场景，诱导用户去“看清”或“触碰”，但通过交互机制给予“惩罚”——用户的介入（鼠标靠近/点击）会导致画面的崩解和模糊。

体验目标：

让用户在互动中体会到一种**“徒劳感”**。唯有保持距离（鼠标远离核心区域或静止不动），画面才会凝聚成清晰、温暖的真相；一旦试图介入，一切又回归冰冷和混沌。

---

2. Implementation Plan (3天冲刺计划)

- **Day 1:** 视觉构建 **(The Atmosphere)**
  - 利用 Three.js 创建一个由数万个微小粒子组成的场景。
  - 冷暖并存：背景是深邃的渐变蓝（雾气），粒子是暖橙色/钨丝灯色（光的聚合）。
  - 这些粒子在初始状态下会聚合成一个模糊的具体形态（如一扇窗、一个人影、或者仅仅是一个完美的光球）。
- **Day 2:** 交互逻辑 **(The Alienation Algorithm)**
  - 编写 Shader（着色器）：计算鼠标与像素的距离。
  - 核心逻辑：鼠标越近 = 粒子扰动幅度越大 + 透明度越低 + 边缘越模糊（高斯模糊）。
  - 边界模糊：粒子之间没有明确界限，使用 AdditiveBlending 让光晕互相融合。
- **Day 3:** 声音与调优 **(The Sound of Distance)**
  - 加入双层音轨：
    - 底层（永远存在）：寒冷的风声、低频深海声（冷）。
    - 表层（靠近时消失）：微弱温暖的电流声、或是模糊的人声（暖）。当你试图靠近听清时，这个声音会被底层的风声吞没。

---

3. Project Core (项目本体)

A. 视觉核心：印象派粒子 (Impressionist Particles)

画面不是高清的照片，而是像莫奈的《日出·印象》或透纳的风暴画。

- 冷环境：整个屏幕充斥着流动的蓝色雾气（Perlin Noise 驱动的体积雾），象征着无法逾越的距离。
- 暖核心：屏幕中央有一团温暖的橙色光晕。它看起来像是一扇温暖的窗户，或者是一个在等待的人。它非常模糊，让你本能地想凑近看清楚。

B. 交互机制：海森堡不确定性 (The Uncertainty Mechanic)

这是对"越想靠近越看不清"的代码转译：

1. 扰动场 (Distortion Field)：鼠标周围有一个不可见的力场。当你移动鼠标去"指"那个暖光时，力场会像手拨动水面一样，把组成暖光的粒子推开、打散。
2. 失焦 (De-focus)：鼠标不仅仅是推开粒子，还会改变粒子的 uBlur 属性。鼠标所指之处，像素急剧羽化，原本清晰的边缘瞬间变成一团化开的烟雾。

C. 隐喻的传递

- 疏离感：你只能在屏幕边缘旁观那个温暖的核心。一旦你想成为参与者（移动鼠标进入中心），温暖就消散了，只剩下冰冷的蓝色背景。

---

4. Walkthrough (演示/导览)

阶段一：诱惑 (The Lure)

用户打开网页。屏幕是一片寂静的深蓝（#001133）。

屏幕中央，无数微小的金色粒子正在缓慢汇聚。它们极其缓慢地旋转、吸附，逐渐形成了一个模糊但温暖的轮廓——也许看起来像是一个家里的窗户。

- 氛围：只有当你把鼠标停在屏幕角落不动时，这个轮廓才会变得稍微清晰一点，显露出暖色的光晕。

阶段二：介入 (The Approach)

用户处于好奇，移动鼠标想要去点击或看清那个光晕。

当鼠标指针滑向屏幕中央时：

- 视觉：就像手伸进了水中倒影。金色的粒子瞬间受到惊吓，向四周四散逃逸（Repulsion）。原本凝聚的形态崩解成无数杂乱的火星。
- 色彩：随着粒子的散开，背景的深蓝色趁虚而入，吞噬了原本的暖色。画面变得浑浊、冰冷。

阶段三：失落与领悟 (The Alienation)

用户停下鼠标，试图等待它恢复。

- 恢复的过程极慢（Damping高阻尼）。粒子像是有记忆一样，在漫长的几秒钟内，战战兢兢地重新试探着聚拢。

- 用户意识到：我的存在就是干扰。
- 为了留住那个美好的暖像，用户必须把鼠标移到屏幕最边缘，或者彻底松开手。这种**“被迫的旁观”**，就是你想要的疏离感。

---

5. 极速原型代码 (Copy-Paste Solution)

这是一个基于 Three.js 的极简原型，实现了**“粒子聚像”和“鼠标靠近即打散”**的核心逻辑。你可以直接运行看效果。

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>The Unreachable Light</title>
  <style>
    body { margin: 0; overflow: hidden; background: #000510; } /* 极深的蓝黑背景 */
    canvas { display: block; }
  </style>
</head>
<body>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>

  <script>
    // --- 场景设置 ---
    const scene = new THREE.Scene();
    // 添加一点环境雾，增加深邃感
    scene.fog = new THREE.FogExp2(0x000510, 0.002);

    const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1,
1000);
    camera.position.z = 50;

    const renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
    renderer.setSize(window.innerWidth, window.innerHeight);
    renderer.setPixelRatio(window.devicePixelRatio); // 保证高清屏清晰
    document.body.appendChild(renderer.domElement);

    // --- 核心：粒子系统 ---
    const particleCount = 15000; // 粒子数量
    const geometry = new THREE.BufferGeometry();
```

```
    const positions = new Float32Array(particleCount * 3);
    const originalPositions = new Float32Array(particleCount * 3); // 记住它们的“家”
    const colors = new Float32Array(particleCount * 3);
    const sizes = new Float32Array(particleCount);

    const colorInside = new THREE.Color(0xffaa33); // 暖橙色 (核心)
    const colorOutside = new THREE.Color(0x0044ff); // 冷蓝色 (边缘)

    for (let i = 0; i < particleCount; i++) {
      // 在中心生成一个球体形状的聚合体
      const u = Math.random();
      const v = Math.random();
      const theta = 2 * Math.PI * u;
      const phi = Math.acos(2 * v - 1);
      const r = 10 + Math.random() * 5; // 半径10-15的球体

      const x = r * Math.sin(phi) * Math.cos(theta);
      const y = r * Math.sin(phi) * Math.sin(theta);
      const z = r * Math.cos(phi);

      positions[i * 3] = x;
      positions[i * 3 + 1] = y;
      positions[i * 3 + 2] = z;

      // 保存原始位置，用于恢复
      originalPositions[i * 3] = x;
      originalPositions[i * 3 + 1] = y;
      originalPositions[i * 3 + 2] = z;

      // 颜色混合：中心暖，边缘冷
      const mixedColor = colorInside.clone().lerp(colorOutside, Math.random() * 0.3);
      colors[i * 3] = mixedColor.r;
      colors[i * 3 + 1] = mixedColor.g;
      colors[i * 3 + 2] = mixedColor.b;

      sizes[i] = Math.random() * 0.8;
    }

    geometry.setAttribute('position', new THREE.BufferAttribute(positions, 3));
    geometry.setAttribute('color', new THREE.BufferAttribute(colors, 3));
    geometry.setAttribute('size', new THREE.BufferAttribute(sizes, 1));

    // 自定义 Shader 材质，实现发光和模糊感
    const material = new THREE.ShaderMaterial({
      uniforms: {
        pointTexture: { value: new
THREE.TextureLoader().load('https://threejs.org/examples/textures/sprites/spark1.png') }
      },
```

```
        vertexShader: `
            attribute float size;
            varying vec3 vColor;
            void main() {
                vColor = color;
                vec4 mvPosition = modelViewMatrix * vec4(position, 1.0);
                gl_PointSize = size * (300.0 / -mvPosition.z);
                gl_Position = projectionMatrix * mvPosition;
            }
        `,
        fragmentShader: `
            uniform sampler2D pointTexture;
            varying vec3 vColor;
            void main() {
                gl_FragColor = vec4(vColor, 1.0);
                gl_FragColor = gl_FragColor * texture2D(pointTexture, gl_PointCoord);
                if (gl_FragColor.a < 0.1) discard; // 剔除透明部分
            }
        `,
        blending: THREE.AdditiveBlending, //以此实现发光叠加
        depthTest: false,
        transparent: true
    });

    const particles = new THREE.Points(geometry, material);
    scene.add(particles);

    // --- 交互逻辑 ---
    const mouse = new THREE.Vector2(9999, 9999); // 初始鼠标在很远的地方
    const raycaster = new THREE.Raycaster();
    const plane = new THREE.Plane(new THREE.Vector3(0, 0, 1), 0); // 虚拟平面用于计算鼠标3D位置

    document.addEventListener('mousemove', (event) => {
        // 归一化鼠标坐标
        mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
        mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;
    });

    // --- 动画循环 ---
    function animate() {
        requestAnimationFrame(animate);

        // 1. 获取鼠标在3D空间中的位置投影
        raycaster.setFromCamera(mouse, camera);
        const target = new THREE.Vector3();
        raycaster.ray.intersectPlane(plane, target);

        const positions = particles.geometry.attributes.position.array;
```

```
        // 2. 遍历所有粒子
        for (let i = 0; i < particleCount; i++) {
            const px = positions[i * 3];
            const py = positions[i * 3 + 1];
            const pz = positions[i * 3 + 2];

            const ox = originalPositions[i * 3];
            const oy = originalPositions[i * 3 + 1];
            const oz = originalPositions[i * 3 + 2];

            // 计算粒子与鼠标的距离
            const dx = px - target.x;
            const dy = py - target.y;
            // z轴简单处理，假设鼠标主要在XY平面影响
            const dist = Math.sqrt(dx * dx + dy * dy);

            // --- 核心算法：疏离感 ---

            // 力场半径：鼠标周围 15 单位内的粒子会受到影响
            const repulsionRadius = 15.0;

            if (dist < repulsionRadius) {
                // 如果鼠标太近，粒子被推开 (Repulsion)
                // 力度与距离成反比
                const force = (repulsionRadius - dist) / repulsionRadius;
                const angle = Math.atan2(dy, dx);

                // 粒子向外飞散，并且带一点随机的混乱(Noise/Blur)
                positions[i * 3] += Math.cos(angle) * force * 1.5 + (Math.random()-0.5)*0.5;
                positions[i * 3 + 1] += Math.sin(angle) * force * 1.5 + (Math.random()-0.5)*0.5;

                // 甚至把它们推向深处（Z轴），让它们看起来变模糊变小
                positions[i * 3 + 2] -= force * 0.5;
            } else {
                // 如果鼠标远离，粒子慢慢回到原来的位置 (Memory/Home)
                // 0.02 是一个很小的数字，代表"慢节奏"
                positions[i * 3] += (ox - px) * 0.02;
                positions[i * 3 + 1] += (oy - py) * 0.02;
                positions[i * 3 + 2] += (oz - pz) * 0.02;
            }
        }

        particles.geometry.attributes.position.needsUpdate = true;

        // 整体缓慢旋转，增加梦幻感
        particles.rotation.y += 0.001;
```

```
        renderer.render(scene, camera);
    }

    animate();

    // 窗口调整
    window.addEventListener('resize', () => {
        camera.aspect = window.innerWidth / window.innerHeight;
        camera.updateProjectionMatrix();
        renderer.setSize(window.innerWidth, window.innerHeight);
    });
  </script>
</body>
</html>
```

关键调整说明

1. **AdditiveBlending (加法混合)**：代码中使用了这个属性，使得橙色粒子重叠时会变得非常亮（像真实的火光），而散开时会变暗。这强化了“聚则暖，散则冷”的效果。
2. **Repulsion Force (斥力)**：我设置了一个 repulsionRadius。只要你鼠标进入这个半径，粒子就会被暴力推开并加入随机噪点（震动），这就是“看不清真相”的视觉化。
3. **Return Speed (回归速度)**：(ox - px) * 0.02 里的 0.02 决定了粒子回家的速度。这个数值很小，意味着破坏很容易，但重建很慢。

你可以直接运行这段代码，它就是你想要的：一个在远处看很美、很暖，但当你试图触碰时就会破碎流失的“蓝调幻象”。