```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
  <title>FIZX DIAGNOSTIC</title>
  <style>
    body { margin: 0; overflow: hidden; background-color: #000; font-family: monospace; }

    /* Full Screen Fix */
    canvas {
      display: block;
      width: 100vw !important;
      height: 100vh !important;
      touch-action: none; /* Prevent scrolling */
    }

    #log {
      position: absolute; top: 10px; left: 10px; width: 90%;
      color: #00ff00; pointer-events: none; white-space: pre-wrap;
      background: rgba(0,0,0,0.5); padding: 5px; font-size: 12px;
      transition: opacity 0.5s ease-out; /* Fade animation */
      z-index: 100;
    }
    #error { color: #ff0000; font-weight: bold; }
  </style>
</head>
<body>
  <div id="log">Initializing FIZX System...<br> Tap screen to hide log.<br></div>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/three@0.128.0/examples/js/controls/OrbitControls.js"></script>

  <script>
    const log = document.getElementById('log');
    function print(msg, isError = false) {
      const span = document.createElement('span');
      span.textContent = msg + "\n";
      if(isError) span.style.color = 'red';
      log.appendChild(span);
    }

    window.onerror = function(message, source, lineno, colno, error) {
            print("CRITICAL ERROR: " + message, true);
        };

        // --- FIX 1: TOUCH TO HIDE ---
        function hideLog() {
            if(log) {
                log.style.opacity = '0';
                setTimeout(() => { log.style.display = 'none'; }, 500);
            }
        }
        document.addEventListener('touchstart', hideLog, { once: true });
        document.addEventListener('click', hideLog, { once: true });

        try {
            print("Creating Scene...");
            const scene = new THREE.Scene();
            // Dark Blue Background
            scene.background = new THREE.Color(0x000033);

            print("Setting up Camera...");
            const camera = new THREE.PerspectiveCamera(50, window.innerWidth /
window.innerHeight, 0.1, 100);
            camera.position.set(0, 10, 20);
            camera.lookAt(0,0,0);

            print("Initializing Renderer...");
            const renderer = new THREE.WebGLRenderer({
                antialias: false,
                precision: 'mediump',
                powerPreference: 'default',
                alpha: true // Enable transparency for the canvas itself if needed
            });
            // Ensure internal size matches window
            renderer.setSize(window.innerWidth, window.innerHeight);
            renderer.setPixelRatio(1.0); // Keep 1.0 for safety, CSS will stretch it
            document.body.appendChild(renderer.domElement);
            print("Renderer Created. Canvas size: " + renderer.domElement.width + "x" +
renderer.domElement.height);

            const controls = new THREE.OrbitControls(camera, renderer.domElement);
            controls.autoRotate = true;
            controls.autoRotateSpeed = 1.0;

            // --- TEST OBJECT 1: Simple Cube (Visible Reference) ---
            print("Adding Reference Cube...");
            const boxGeo = new THREE.BoxGeometry(2,2,2);
            const boxMat = new THREE.MeshBasicMaterial({ color: 0xffffff, wireframe: true,
transparent: true, opacity: 0.1 });
            const box = new THREE.Mesh(boxGeo, boxMat);
            scene.add(box);

            // --- TEST OBJECT 2: Horn Torus Particles ---
            print("Generating Fluid System...");
            const R = 6.0;
            const particleCount = 4000; // Increased count for better look
            const geo = new THREE.BufferGeometry();
            const pos = new Float32Array(particleCount * 3);
            const col = new Float32Array(particleCount * 3);

            // Store particle data for animation
            const particles = [];

            for(let i=0; i<particleCount; i++) {
                const theta = Math.random() * Math.PI * 2;
                const phi = Math.random() * Math.PI * 2;
                const r = 6.0;

                // Physics state
                particles.push({
                  theta: theta,
                  phi: phi,
                  speed: 0.005 + Math.random() * 0.02,
                  offset: Math.sqrt(Math.random()) * r // Uniform volume distribution
                });

                // Initial Position
                const tx = R + (particles[i].offset * Math.cos(theta));
                const ty = particles[i].offset * Math.sin(theta);

                pos[i*3] = tx * Math.cos(phi);
                pos[i*3+1] = ty;
                pos[i*3+2] = tx * Math.sin(phi);

                // Initial Color (Will be overwritten in animate)
                col[i*3] = 1; col[i*3+1] = 1; col[i*3+2] = 1;
            }
            geo.setAttribute('position', new THREE.BufferAttribute(pos, 3));
            geo.setAttribute('color', new THREE.BufferAttribute(col, 3));
```

```
const mat = new THREE.PointsMaterial({
   size: 0.25,
   vertexColors: true,
   sizeAttenuation: true
});
const system = new THREE.Points(geo, mat);
scene.add(system);
print("System Added.");

// --- NEW OBJECT: Glass Horn Torus Shell ---
print("Adding Glass Shell...");
// Add lights for the glass material so it shines
const ambientLight = new THREE.AmbientLight(0x404040); // soft white light
scene.add(ambientLight);
const pointLight1 = new THREE.PointLight(0xffffff, 1, 100);
pointLight1.position.set(15, 20, 15);
scene.add(pointLight1);
const pointLight2 = new THREE.PointLight(0xaaddff, 0.8, 100); // bluish light from below
pointLight2.position.set(-15, -20, -15);
scene.add(pointLight2);

// Horn Torus Geometry (R=r=6.0) to match particle bounds
// Using higher segments for a smooth glass look
const shellR = 6.0;
const shellr = 6.0;
const shellGeo = new THREE.TorusGeometry(shellR, shellr, 64, 128);

// Glass Material (Translucent, shiny, colored clear)
const shellMat = new THREE.MeshPhongMaterial({
   color: 0xaaddff, // Light cyan/blue tint for "colored clear glass"
   transparent: true,
   opacity: 0.15,   // "nearly translucent"
   shininess: 100,  // High shininess for sharp reflections
   specular: 0xffffff, // White highlights
   side: THREE.DoubleSide, // Render inside and outside surfaces
   depthWrite: false // Important: prevents the glass from occluding particles incorrectly
});

const shell = new THREE.Mesh(shellGeo, shellMat);
// Rotate to lay flat, matching the particle system's orientation
shell.rotation.x = -Math.PI / 2;
scene.add(shell);
print("Glass Shell Added.");
```

```
function animate() {
    requestAnimationFrame(animate);

    box.rotation.y += 0.02;

    const positions = system.geometry.attributes.position.array;
    const colors = system.geometry.attributes.color.array;

    for(let i=0; i<particleCount; i++) {
        const p = particles[i];

        // Move particles (Poloidal Flow)
        p.theta += p.speed;
        p.phi += p.speed * 0.2;

        // Calculate 3D Position
        const tx = R + (p.offset * Math.cos(p.theta));
        const ty = p.offset * Math.sin(p.theta);

        const x = tx * Math.cos(p.phi);
        const y = ty;
        const z = tx * Math.sin(p.phi);

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

        // --- FIX 3: HEMISPHERE COLORS ---
        if (y >= 0) {
            // NORTHERN HEMISPHERE -> NEON YELLOW (1, 1, 0)
            colors[i*3] = 1.0;     // R
            colors[i*3+1] = 1.0;   // G
            colors[i*3+2] = 0.0;   // B
        } else {
            // SOUTHERN HEMISPHERE -> NEON ORANGE (1, 0.5, 0)
            colors[i*3] = 1.0;     // R
            colors[i*3+1] = 0.5;   // G
            colors[i*3+2] = 0.0;   // B
        }
    }

    system.geometry.attributes.position.needsUpdate = true;
    system.geometry.attributes.color.needsUpdate = true;
```

```
                renderer.render(scene, camera);
            }
            animate();
            print("Animation Loop Started.");

        } catch (e) {
            print("SETUP FAILED: " + e.message, true);
        }

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