```
<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Ring GLB Viewer</title>

<meta name="viewport" content="width=device-width,initial-scale=1.0">

<style>

  body { margin:0; background:#000; overflow:hidden; font-family: Arial, sans-serif; color:#fff; }

  #overlay { position: absolute; left: 12px; top: 12px; z-index:5; background: rgba(0,0,0,0.35);
padding:8px 12px; border-radius:8px; }

  button { margin-left:8px; }

  #credit { position: absolute; right: 12px; bottom:12px; color:#ddd; font-size:12px; z-index:5; }

</style>

</head>

<body>

<div id="overlay">

  <strong>Silver Ring • Red Coral</strong>

  <button id="playPause">Pause</button>
```

```
  <button id="resetCam">Reset</button>

</div>

<div id="credit">Place <code>ring.glb</code> in same folder</div>

<script type="module">

  import * as THREE from 'https://cdn.jsdelivr.net/npm/three@0.150.0/build/three.module.js';

  import { OrbitControls } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/controls/OrbitControls.js';

  import { GLTFLoader } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/loaders/GLTFLoader.js';

  import { RGBELoader } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/loaders/RGBELoader.js';

  import { EffectComposer } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/postprocessing/EffectComposer.js';

  import { RenderPass } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/postprocessing/RenderPass.js';

  import { UnrealBloomPass } from 'https://cdn.jsdelivr.net/npm/three@0.150.0/examples/jsm/postprocessing/UnrealBloomPass.js';

  // Renderer

  const renderer = new THREE.WebGLRenderer({ antialias:true });

  renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2));
```

```
  renderer.setSize(window.innerWidth, window.innerHeight);

  renderer.outputEncoding = THREE.sRGBEncoding;

  renderer.physicallyCorrectLights = true;

  renderer.toneMapping = THREE.ACESFilmicToneMapping;

  document.body.appendChild(renderer.domElement);



  // Scene + Camera

  const scene = new THREE.Scene();

  scene.background = new THREE.Color(0x060606);



  const camera = new THREE.PerspectiveCamera(45, window.innerWidth/window.innerHeight,
0.1, 100);

  const INITIAL_CAMERA_POS = new THREE.Vector3(6, -6, 3.5);

  camera.position.copy(INITIAL_CAMERA_POS);

  camera.lookAt(0,0,0);



  // Controls

  const controls = new OrbitControls(camera, renderer.domElement);
```

```
controls.target.set(0,0,0.12);

controls.enableDamping = true;

controls.minDistance = 1.6;

controls.maxDistance = 20;



// Light (scene-level)

const hemi = new THREE.HemisphereLight(0xffffff, 0x222244, 0.2);

scene.add(hemi);



const dir = new THREE.DirectionalLight(0xffffff, 0.6);

dir.position.set(5, -5, 6);

scene.add(dir);



// small rim light

const rim = new THREE.DirectionalLight(0x99ddff, 0.6);

rim.position.set(-2, 5, 3);

scene.add(rim);
```

```
// Floor reflection plane (simple)

const floorGeo = new THREE.PlaneGeometry(40, 40);

const floorMat = new THREE.MeshStandardMaterial({ color:0x050505, roughness:0.75, metalness:0.0 });

const floor = new THREE.Mesh(floorGeo, floorMat);

floor.rotation.x = -Math.PI/2;

floor.position.y = -0.6;

floor.receiveShadow = true;

scene.add(floor);



// Postprocessing (bloom)

const composer = new EffectComposer(renderer);

composer.addPass(new RenderPass(scene, camera));

const bloom = new UnrealBloomPass(new THREE.Vector2(window.innerWidth, window.innerHeight), 0.28, 0.6, 0.1);

bloom.strength = 0.55;

composer.addPass(bloom);



// Load GLB
```

```
const loader = new GLTFLoader();

let mixer = null;

let modelRoot = new THREE.Group();

scene.add(modelRoot);



loader.load('ring.glb', (gltf) => {

  modelRoot.add(gltf.scene);

  // center & scale safety (if needed)

  const box = new THREE.Box3().setFromObject(gltf.scene);

  const size = new THREE.Vector3();

  box.getSize(size);

  const maxdim = Math.max(size.x, size.y, size.z);

  if (maxdim > 2.0) {

    const s = 1.6 / maxdim;

    gltf.scene.scale.setScalar(s);

  }

  // Animation (if present in glb)
```

```
  if (gltf.animations && gltf.animations.length) {
    mixer = new THREE.AnimationMixer(gltf.scene);
    for (const clip of gltf.animations) {
      const action = mixer.clipAction(clip);
      action.play();
      action.setLoop(THREE.LoopRepeat);
    }
  } else {
    // if no animation exists, you can rotate the model manually in render loop
  }
}, undefined, (err) => {
  console.error('Error loading ring.glb — make sure file is next to this HTML', err);
});

// Auto-rotate fallback if glb had no animations
let autoRotate = true;
const playBtn = document.getElementById('playPause');
playBtn.addEventListener('click', () => {
```

```
  autoRotate = !autoRotate;
  playBtn.textContent = autoRotate ? 'Pause' : 'Play';
});

document.getElementById('resetCam').addEventListener('click', () => {
  camera.position.copy(INITIAL_CAMERA_POS);
  controls.target.set(0,0,0.12);
  controls.update();
});

// Render Loop
const clock = new THREE.Clock();
function animate() {
  requestAnimationFrame(animate);
  const dt = clock.getDelta();
  if (mixer) mixer.update(dt);
  else if (autoRotate) modelRoot.rotation.z += 0.2 * dt; // fallback auto-rotate
```

```
    controls.update();

    composer.render();

  }

  animate();



  window.addEventListener('resize', () => {

    camera.aspect = window.innerWidth/window.innerHeight;

    camera.updateProjectionMatrix();

    renderer.setSize(window.innerWidth, window.innerHeight);

    composer.setSize(window.innerWidth, window.innerHeight);

  });

</script>

</body>

</html>
```