```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width,initial-scale=1"/>
<title>Universe — Final Interactive Experience</title>
<link href="https://fonts.googleapis.com/css2?
family=Inter:wght@300;400;600;800&display=swap" rel="stylesheet">
<style>
 :root{
  --bg:#030517; --card:#07102633; --muted:#9fb0bd; --accent:#7ee7c8; --
glass:rgba(255,255,255,0.03);
  --radius:14px;
 }
 html,body{height:100%;margin:0;font-family:Inter,system-ui,Arial;background: radial-
gradient(1200px 600px at 10% 10%, #081226 0%, #030517 25%, #000 100%); color:#e9f2f7; -
webkit-font-smoothing:antialiased;}
 #app{display:grid; grid-template-columns: 1fr 380px; gap:18px; height:100vh; padding:18px;
box-sizing:border-box;}
 @media (max-width:1000px){ #app{grid-template-columns:1fr; grid-auto-rows:min-content;}
#panel{height:44vh; overflow:auto;} #canvas-wrap{height:56vh;} }
 /* canvas area */
 #canvas-wrap{ position:relative; border-radius:var(--radius); overflow:hidden;
background:linear-gradient(180deg, rgba(20,25,40,0.2), transparent); border:1px solid
rgba(255,255,255,0.03); box-shadow:0 12px 60px rgba(0,0,0,0.6);}
 canvas{ display:block; width:100%; height:100%; background:transparent; }
 /* HUD */
 .hud{ position:absolute; left:16px; top:16px; z-index:30; display:flex; gap:10px; }
 .btn{ background:linear-gradient(180deg, rgba(255,255,255,0.02), rgba(255,255,255,0.01));
border:1px solid rgba(255,255,255,0.04); color:var(--accent); padding:8px 12px; border-
radius:10px; cursor:pointer; font-weight:700; font-size:13px; }
 .btn.secondary{ color:var(--muted); background:transparent; border:1px dashed
rgba(255,255,255,0.03); font-weight:600; }
 .control-row{ display:flex; gap:8px; align-items:center; background:linear-gradient(90deg,
rgba(255,255,255,0.012), rgba(255,255,255,0.006)); padding:8px; border-radius:10px; border:1px
solid rgba(255,255,255,0.02); }
 input[type="range"]{ accent-color:var(--accent);}
 /* labels */
 #labels{ position:absolute; left:0; top:0; pointer-events:none; width:100%; height:100%; z-
index:25; }
 .label{ position:absolute; transform:translate(-50%,-120%); background:linear-gradient(90deg,
rgba(2,8,18,0.7), rgba(255,255,255,0.02)); padding:6px 10px; border-radius:999px; font-
```

```
weight:700; display:flex; gap:8px; align-items:center; border:1px solid rgba(255,255,255,0.04);
box-shadow:0 10px 30px rgba(1,6,12,0.6); font-size:12px;}
 .label .sub{ font-weight:500; color:var(--muted); font-size:11px; margin-left:6px;}
 /* canvas small UI */
 .canvas-ui{ position:absolute; right:14px; bottom:14px; z-index:30; display:flex; flex-
direction:column; gap:8px; align-items:flex-end; }
 .chip{ padding:8px 12px; border-radius:999px; background:var(--glass); border:1px solid
rgba(255,255,255,0.02); color:var(--muted); font-size:13px; }
 /* side panel */
 #panel{ border-radius:var(--radius); background:linear-gradient(180deg,
rgba(255,255,255,0.015), rgba(255,255,255,0.01)); padding:18px; box-shadow:0 12px 40px
rgba(0,0,0,0.6); border:1px solid rgba(255,255,255,0.03); overflow:auto;}
 .hero{ display:flex; gap:12px; align-items:center; margin-bottom:12px;}
 .logo{ width:52px;height:52px;border-radius:12px;background:linear-
gradient(135deg,#7ee7c8,#7dd3fc); display:flex; align-items:center; justify-content:center; font-
weight:900; color:#022; font-size:20px;}
 h1{ margin:0;font-size:18px;line-height:1; }
 .muted{ color:var(--muted); font-size:13px; margin-top:6px;}
 .search{ display:flex; gap:8px; margin-top:12px; }
 input[type="search"]{ flex:1; padding:10px 12px; background:transparent; border:1px solid
rgba(255,255,255,0.03); border-radius:10px; color:var(--muted); outline:none;}
 .list{ margin-top:12px; display:grid; gap:8px; max-height:40vh; overflow:auto; padding-
right:6px; }
 .card{ background:rgba(255,255,255,0.02); padding:10px; border-radius:10px; display:flex;
gap:10px; align-items:center; cursor:pointer; border:1px solid rgba(255,255,255,0.015);}
 .dot{ width:46px;height:46px;border-radius:10px; display:flex; align-items:center; justify-
content:center; font-weight:800; color:#021; }
 #info{ margin-top:12px; padding:12px; border-radius:10px; background:linear-
gradient(180deg, rgba(255,255,255,0.01), transparent); border:1px solid rgba(255,255,255,0.02);
min-height:120px; }
 .small{ font-size:13px;color:var(--muted); }
 footer{ margin-top:14px;color:var(--muted); font-size:12px; text-align:center;}
 /* quiz */
 .quiz .quiz-row{ display:flex; gap:8px; margin-top:8px;}
 .timeline{ margin-top:12px; border-radius:8px; padding:10px;
background:rgba(255,255,255,0.015); border:1px solid rgba(255,255,255,0.02); }
</style>
</head>
<body>
 <div id="app" role="application" aria-label="Universe interactive explorer">
  <div id="canvas-wrap" aria-hidden="false">
   <div class="hud">
    <div class="control-row" role="group" aria-label="controls">
     <button id="playBtn" class="btn" title="Play/Pause (Space)">Pause</button>
```

```
      <button id="tourBtn" class="btn secondary" title="Guided tour">Start Tour</button>
      <button id="storyBtn" class="btn secondary" title="Timeline story">Story</button>
      <label style="display:flex;align-items:center;color:var(--muted);font-size:13px;">
       Speed <input id="speed" type="range" min="0" max="6" step="0.05" value="1"
style="margin-left:8px;width:140px"/>
      </label>
     </div>
    </div>

    <div id="labels"></div>

    <div class="canvas-ui">
     <div class="chip" id="timeChip">Time: 1×</div>
     <div class="chip" id="selectedChip">Selected: —</div>
    </div>
   </div>

   <aside id="panel" aria-label="Information panel">
    <div class="hero">
     <div class="logo">U</div>
     <div>
      <h1>Universe — Final Build</h1>
      <div class="muted">Interactive 3D solar system with textures, GLTF models, bloom, rings,
timeline and quiz.</div>
     </div>
    </div>

    <div class="search" role="search">
     <input id="search" type="search" placeholder="Search a body (Earth, Jupiter...)" aria-
label="Search celestial body" />
     <button id="screenshot" class="btn secondary" title="Download screenshot">📷</button>
    </div>

    <div class="list" id="list" role="list">
     <!-- filled dynamically -->
    </div>

    <div id="info" aria-live="polite">
     <h2>Welcome</h2>
     <div class="small">Click a planet, moon, spacecraft, or comet. Use the guided tour or
timeline for a narrated story.</div>
    </div>

    <div style="margin-top:12px;">
```

```
      <h3 style="margin:6px 0 6px 0">Quick Quiz</h3>
      <div class="quiz" id="quiz">
       <div class="card">Which planet is known as the Red Planet?
        <div class="quiz-row" style="margin-top:8px;">
         <button class="btn secondary quiz-btn" data-q="1" data-a="Mars">Mars</button>
         <button class="btn secondary quiz-btn" data-q="1" data-a="Venus">Venus</button>
         <button class="btn secondary quiz-btn" data-q="1" data-a="Jupiter">Jupiter</button>
        </div>
        <div class="small" id="r1"></div>
       </div>
       <div class="card" style="margin-top:8px">Which is the largest planet?
        <div class="quiz-row" style="margin-top:8px;">
         <button class="btn secondary quiz-btn" data-q="2" data-a="Saturn">Saturn</button>
         <button class="btn secondary quiz-btn" data-q="2" data-a="Jupiter">Jupiter</button>
         <button class="btn secondary quiz-btn" data-q="2" data-a="Neptune">Neptune</button>
        </div>
        <div class="small" id="r2"></div>
       </div>
      </div>
     </div>

     <div class="timeline" id="timeline" aria-label="solar system timeline">
      <h4 style="margin:0 0 8px 0">Timeline Story Mode</h4>
      <div class="small" id="storyText">Formation of the solar system ~4.6 billion years ago — press "Story" to begin.</div>
      <div style="display:flex;gap:8px;margin-top:8px;">
       <button id="prevStory" class="btn secondary">Prev</button>
       <button id="nextStory" class="btn">Next</button>
       <button id="autoStory" class="btn secondary">Auto Play</button>
      </div>
     </div>

     <footer>Keyboard: Space play/pause, 1 overview, 2 Earth, 3 Jupiter.</footer>
    </aside>
   </div>

   <!-- Libraries -->
   <script src="https://cdn.jsdelivr.net/npm/three@0.156.0/build/three.min.js"></script>
   <script src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/controls/OrbitControls.js"></script>
   <script src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/pmrem/PMREMGenerator.js"></script>
```

```
 <script src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/loaders/RGBELoader.js">
</script>
 <script src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/loaders/GLTFLoader.js">
</script>
 <script
src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/postprocessing/EffectComposer.js"
></script>
 <script
src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/postprocessing/RenderPass.js">
</script>
 <script
src="https://cdn.jsdelivr.net/npm/three@0.156.0/examples/js/postprocessing/UnrealBloomPass.js
"></script>

 <script>
 // Why: single-file final app. Minimal essential comments only.

 // DOM refs
 const canvasWrap = document.getElementById('canvas-wrap');
 const labelsDiv = document.getElementById('labels');
 const listElem = document.getElementById('list');
 const infoPane = document.getElementById('info');
 const searchInput = document.getElementById('search');
 const playBtn = document.getElementById('playBtn');
 const speedSlider = document.getElementById('speed');
 const timeChip = document.getElementById('timeChip');
 const selectedChip = document.getElementById('selectedChip');
 const tourBtn = document.getElementById('tourBtn');
 const storyBtn = document.getElementById('storyBtn');
 const screenshotBtn = document.getElementById('screenshot');

 // renderer + scene + camera
 const renderer = new THREE.WebGLRenderer({ antialias:true, alpha:true });
 renderer.setPixelRatio(Math.min(window.devicePixelRatio, 1.6));
 renderer.outputEncoding = THREE.sRGBEncoding;
 canvasWrap.appendChild(renderer.domElement);

 const scene = new THREE.Scene();
 const camera = new THREE.PerspectiveCamera(45, 1, 0.1, 20000);
 camera.position.set(0, 220, 780);
 const controls = new THREE.OrbitControls(camera, renderer.domElement);
 controls.enableDamping = true;
 controls.dampingFactor = 0.12;
 controls.minDistance = 20;
```

```
  controls.maxDistance = 8000;

  // lighting
  const ambient = new THREE.AmbientLight(0x666a78, 0.35);
  scene.add(ambient);
  const sunLight = new THREE.PointLight(0xfff2d1, 2.4, 10000, 2);
  scene.add(sunLight);

  // composer + bloom
  const composer = new THREE.EffectComposer(renderer);
  composer.setSize(window.innerWidth, window.innerHeight);
  composer.addPass(new THREE.RenderPass(scene, camera));
  const bloom = new THREE.UnrealBloomPass(new THREE.Vector2(window.innerWidth,
window.innerHeight), 0.9, 0.4, 0.85);
  bloom.threshold = 0.1;
  bloom.strength = 1.2;
  bloom.radius = 0.8;
  composer.addPass(bloom);

  // PMREM + environment (attempt HDR then fallback to cube)
  const pmrem = new THREE.PMREMGenerator(renderer);
  pmrem.compileEquirectangularShader();

  // small library of textures (public examples). Some URLs may 404 in rare cases — app handles
missing gracefully.
  const TEX = {
    sun: 'https://threejsfundamentals.org/threejs/resources/images/sun.jpg',
    earth: 'https://threejsfundamentals.org/threejs/resources/images/earth-day.jpg',
    earthClouds: 'https://threejsfundamentals.org/threejs/resources/images/earth-clouds.png',
    moon: 'https://threejsfundamentals.org/threejs/resources/images/moon.jpg',
    mars: 'https://threejsfundamentals.org/threejs/resources/images/mars.jpg',
    mercury: 'https://threejsfundamentals.org/threejs/resources/images/mercury.jpg',
    venus: 'https://threejsfundamentals.org/threejs/resources/images/venus.jpg',
    jupiter: 'https://threejsfundamentals.org/threejs/resources/images/jupiter.jpg',
    saturn: 'https://threejsfundamentals.org/threejs/resources/images/saturn.jpg',
    uranus: 'https://threejsfundamentals.org/threejs/resources/images/uranus.jpg',
    neptune: 'https://threejsfundamentals.org/threejs/resources/images/neptune.jpg',
    pluto: 'https://threejsfundamentals.org/threejs/resources/images/pluto.jpg',
    ceres: 'https://threejsfundamentals.org/threejs/resources/images/ceres.jpg',
    ringSaturn: 'https://threejsfundamentals.org/threejs/resources/images/saturn-ring.png',
    starsBox: [
      "https://threejs.org/examples/textures/cube/space/px.jpg",
      "https://threejs.org/examples/textures/cube/space/nx.jpg",
      "https://threejs.org/examples/textures/cube/space/py.jpg",
```

```
    "https://threejs.org/examples/textures/cube/space/ny.jpg",
    "https://threejs.org/examples/textures/cube/space/pz.jpg",
    "https://threejs.org/examples/textures/cube/space/nz.jpg"
  ]
};

// data model — approximate orbit km values just for info; on-screen we scale them.
const rawData = [
  {id:'sun', name:'Sun', type:'star', radius_km:696340, scaleRadius:36, orbit_km:0,
texture:TEX.sun, descr:'The star at the center of the solar system.'},
  {id:'mercury', name:'Mercury', type:'planet', radius_km:2440, scaleRadius:3.0, orbit_km:57.9,
texture:TEX.mercury, descr:'Small, rocky and closest to the Sun.'},
  {id:'venus', name:'Venus', type:'planet', radius_km:6052, scaleRadius:5.9, orbit_km:108.2,
texture:TEX.venus, descr:'Dense atmosphere and extreme greenhouse effect.'},
  {id:'earth', name:'Earth', type:'planet', radius_km:6371, scaleRadius:6.1, orbit_km:149.6,
texture:TEX.earth, cloudTexture:TEX.earthClouds, descr:'Our home planet, the only known life-
hosting world.'},
  {id:'moon', name:'Moon', type:'moon', parent:'earth', radius_km:1737, scaleRadius:1.6,
orbit_km:0.384, texture:TEX.moon, descr:'Earth\'s natural satellite.'},
  {id:'mars', name:'Mars', type:'planet', radius_km:3390, scaleRadius:3.3, orbit_km:227.9,
texture:TEX.mars, descr:'The Red Planet — dusty with giant volcanoes.'},
  {id:'ceres', name:'Ceres', type:'dwarf', radius_km:473, scaleRadius:1.2, orbit_km:414,
texture:TEX.ceres, descr:'Largest object in the asteroid belt.'},
  {id:'jupiter', name:'Jupiter', type:'planet', radius_km:69911, scaleRadius:17.0, orbit_km:778.6,
texture:TEX.jupiter, descr:'Gas giant with Great Red Spot.'},
  {id:'saturn', name:'Saturn', type:'planet', radius_km:58232, scaleRadius:14.0, orbit_km:1427,
texture:TEX.saturn, ringTexture:TEX.ringSaturn, descr:'Iconic rings made of ice and rock.'},
  {id:'uranus', name:'Uranus', type:'planet', radius_km:25362, scaleRadius:9.5, orbit_km:2871,
texture:TEX.uranus, descr:'Ice giant with tilted axis.'},
  {id:'neptune', name:'Neptune', type:'planet', radius_km:24622, scaleRadius:9.3, orbit_km:4495,
texture:TEX.neptune, descr:'Farthest classical planet; vivid blue.'},
  {id:'pluto', name:'Pluto', type:'dwarf', radius_km:1188, scaleRadius:1.8, orbit_km:5906,
texture:TEX.pluto, descr:'Dwarf planet in the Kuiper Belt.'}
];

// scales for visualization (distances are not to scale)
const distScale = 0.45; // scale AU-ish numbers to screen units
const radiusScale = 1.0; // use scaleRadius in data

// storage
const bodies = {}; // id -> {pivot,mesh,data,labelEl}
const texLoader = new THREE.TextureLoader();
const gltfLoader = new THREE.GLTFLoader();
```

```
  // create orbit pivots and meshes
  function createLabelEl(name, sub){
    const el = document.createElement('div');
    el.className = 'label';
    el.innerHTML = `<strong>${name}</strong><span class="sub">${sub||''}</span>`;
    labelsDiv.appendChild(el);
    return el;
  }

  rawData.forEach(d=>{
    // pivot rotates around sun
    const pivot = new THREE.Object3D();
    scene.add(pivot);

    // material
    const geom = new THREE.SphereGeometry(Math.max(d.scaleRadius * radiusScale, 0.2), 40,
40);
    const matOpts = { roughness:1, metalness:0 };
    try { matOpts.map = texLoader.load(d.texture); } catch(e){}
    const material = new THREE.MeshStandardMaterial(matOpts);
    const mesh = new THREE.Mesh(geom, material);
    mesh.userData = { id:d.id, name:d.name };
    // position mesh at orbit distance along +x
    const orbitPos = (d.orbit_km || 0) * distScale;
    mesh.position.set(orbitPos, 0, 0);
    pivot.add(mesh);

    // if planet with clouds (Earth) add cloud layer
    if(d.cloudTexture){
      const cloudGeo = new THREE.SphereGeometry(Math.max(d.scaleRadius * radiusScale * 1.01,
0.25), 40, 40);
      const cloudMat = new THREE.MeshPhongMaterial({ map: texLoader.load(d.cloudTexture),
transparent:true, opacity:0.45, depthWrite:false });
      const cloudMesh = new THREE.Mesh(cloudGeo, cloudMat);
      mesh.add(cloudMesh);
      mesh.cloudMesh = cloudMesh;
    }

    // orbital ring (visual aid)
    if(d.orbit_km && d.orbit_km > 0.5){
      const radius = orbitPos;
      const ringGeo = new THREE.RingGeometry(radius - 0.6, radius + 0.6, 256);
      const ringMat = new THREE.MeshBasicMaterial({ color:0xffffff, transparent:true,
opacity:0.02, side:THREE.DoubleSide });
```

```
    const ring = new THREE.Mesh(ringGeo, ringMat);
    ring.rotation.x = Math.PI / 2;
    scene.add(ring);
   }

   // Saturn rings special
   if(d.ringTexture){
    const inner = (radiusScale * d.scaleRadius * 1.15);
    const outer = inner + 18;
    const ringGeo = new THREE.RingGeometry(inner, outer, 256);
    const ringMap = texLoader.load(d.ringTexture);
    const ringMat = new THREE.MeshBasicMaterial({ map: ringMap, side:THREE.DoubleSide,
transparent:true, opacity:0.95 });
    const ringMesh = new THREE.Mesh(ringGeo, ringMat);
    ringMesh.rotation.x = Math.PI / 2;
    mesh.add(ringMesh);
   }

   // additional: create particles for ice giants to simulate faint rings
   if(d.id === 'uranus' || d.id === 'neptune'){
    const count = 500;
    const positions = new Float32Array(count * 3);
    for(let i=0;i<count;i++){
     const r = (radiusScale * d.scaleRadius) + 10 + Math.random() * 30;
     const theta = Math.random() * Math.PI * 2;
     positions[i*3] = Math.cos(theta) * r;
     positions[i*3+1] = (Math.random()-0.5) * 2;
     positions[i*3+2] = Math.sin(theta) * r;
    }
    const pGeo = new THREE.BufferGeometry();
    pGeo.setAttribute('position', new THREE.BufferAttribute(positions, 3));
    const pMat = new THREE.PointsMaterial({ color:0x99ddee, size:0.4, transparent:true,
opacity:0.7 });
    const points = new THREE.Points(pGeo, pMat);
    mesh.add(points);
   }

   // shiny halo for sun
   if(d.id === 'sun'){
    const spriteMap = texLoader.load('https://threejs.org/examples/textures/sprites/glow.png');
    const spriteMat = new THREE.SpriteMaterial({ map: spriteMap, color: 0xffddaa,
transparent:true, opacity:0.7, blending:THREE.AdditiveBlending });
    const sprite = new THREE.Sprite(spriteMat);
    sprite.scale.set(260,260,1);
```

```
    mesh.add(sprite);
    mesh.material.emissive = new THREE.Color(0xffcc66);
    mesh.material.emissiveIntensity = 0.8;
   }

   // store
   bodies[d.id] = { pivot, mesh, data:d, labelEl: createLabelEl(d.name, d.type) };
  });

  // attach moon pivot to earth (local orbit)
  (function(){
   const moon = bodies['moon'], earth = bodies['earth'];
   if(moon && earth){
    scene.remove(moon.pivot);
    // position moon pivot relative to earth
    moon.pivot = new THREE.Object3D();
    moon.pivot.position.copy(earth.mesh.position);
    scene.add(moon.pivot);
    moon.mesh.position.set((moon.data.orbit_km || 0.384) * distScale +
(earth.data.scaleRadius*radiusScale*1.8), 0, 0);
    moon.pivot.add(moon.mesh);
   }
  })();

  // add starfield points for depth
  (function createStarfield(){
   const geo = new THREE.BufferGeometry();
   const count = 3500;
   const positions = new Float32Array(count * 3);
   const colors = new Float32Array(count * 3);
   for(let i=0;i<count;i++){
    const r = 800 + Math.random() * 2200;
    const theta = Math.random() * Math.PI * 2;
    const phi = Math.acos((Math.random()*2)-1);
    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;
    const c = 0.6 + Math.random() * 0.4;
    colors[i*3] = c; colors[i*3+1] = c; colors[i*3+2] = c;
   }
   geo.setAttribute('position', new THREE.BufferAttribute(positions,3));
   geo.setAttribute('color', new THREE.BufferAttribute(colors,3));
```

```
  const mat = new THREE.PointsMaterial({ size:1.2, vertexColors:true, transparent:true,
opacity:0.95, depthWrite:false });
  const pts = new THREE.Points(geo, mat);
  scene.add(pts);
 })();

 // comet: small object + trail
 const cometObj = (function(){
  const geom = new THREE.SphereGeometry(1.1, 12, 12);
  const mat = new THREE.MeshStandardMaterial({ color:0xfff1d6, roughness:0.6, metalness:0.1
});
  const mesh = new THREE.Mesh(geom, mat);
  mesh.position.set(-700, 200, -300);
  scene.add(mesh);
  // tail using Line
  const tailMat = new THREE.LineBasicMaterial({ color:0xfff1d6, transparent:true, opacity:0.45
});
  const tailGeo = new THREE.BufferGeometry().setFromPoints([ new
THREE.Vector3(-705,195,-305), new THREE.Vector3(-695,205,-295), new
THREE.Vector3(-680,220,-280) ]);
  const tail = new THREE.Line(tailGeo, tailMat);
  scene.add(tail);
  return { mesh, tail };
 })();

 // environment/background (try HDR-ish equirect then fallback to cube)
 (function loadEnv(){
  const rgbe = new THREE.RGBELoader();
  rgbe.setDataType(THREE.UnsignedByteType);
  // attempt to load a small HDR-like panorama (if blocked, fallback)
  rgbe.load('https://threejs.org/examples/textures/equirectangular/venice_sunset_1k.hdr',
   function(tex){
    const env = pmrem.fromEquirectangular(tex).texture;
    scene.environment = env;
    tex.dispose();
   },
   undefined,
   function(err){
    // fallback: cube textures
    const loader = new THREE.CubeTextureLoader();
    scene.background = loader.load(TEX.starsBox);
   }
  );
 })();
```

```
  // load GLTF models (ISS, Voyager) — use sample repo
  (function loadGLTFs(){
    const models = [
      {url:'https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-
Models/master/2.0/ISS/glTF/ISS.gltf', pos:[140,40,20], scale:0.12, name:'ISS'},
      {url:'https://raw.githubusercontent.com/KhronosGroup/glTF-Sample-
Models/master/2.0/Voyager/glTF/Voyager.gltf', pos:[420,120,-80], scale:3, name:'Voyager'}
    ];
    models.forEach(m=>{
      gltfLoader.load(m.url, (g)=>{
        g.scene.traverse(n=>{ if(n.isMesh){ n.castShadow=false; n.receiveShadow=false; }});
        g.scene.position.set(...m.pos);
        g.scene.scale.setScalar(m.scale);
        g.scene.name = m.name;
        scene.add(g.scene);
      }, undefined, ()=>{/*fail silently*/});
    });
  })();

  // raycasting & interaction
  const raycaster = new THREE.Raycaster();
  const mouse = new THREE.Vector2();
  let hovered = null;
  let selectedId = null;

  function onPointerMove(e){
    const rect = renderer.domElement.getBoundingClientRect();
    mouse.x = ((e.clientX - rect.left) / rect.width) * 2 - 1;
    mouse.y = -((e.clientY - rect.top) / rect.height) * 2 + 1;
  }
  window.addEventListener('pointermove', onPointerMove);

  renderer.domElement.addEventListener('click', (e)=>{
    onPointerMove(e);
    raycaster.setFromCamera(mouse, camera);
    const allMeshes = Object.values(bodies).map(b=>b.mesh);
    const intersects = raycaster.intersectObjects(allMeshes, true);
    if(intersects.length){
      const mesh = intersects[0].object;
      const b = Object.values(bodies).find(x => x.mesh === mesh || x.mesh === mesh.parent ||
mesh.parent === x.mesh);
      if(b) selectBody(b.data.id);
    } else {
```

```
    // maybe clicked a model
    const modelIntersects = raycaster.intersectObjects(scene.children, true).filter(i=>i.object &&
i.object.parent && (i.object.parent.name === 'ISS' || i.object.parent.name === 'Voyager'));
    if(modelIntersects.length){ alert('Spacecraft selected — info pane can be extended.'); }
    else selectBody(null);
   }
  });

  function selectBody(id){
   if(selectedId && bodies[selectedId]) bodies[selectedId].mesh.material.emissiveIntensity =
bodies[selectedId].data.id==='sun' ? 0.8 : 0.0;
   selectedId = id;
   if(!id){
    selectedChip.textContent = 'Selected: —';
    infoPane.innerHTML = `<h2>Welcome</h2><div class="small">Click a body to view details.
</div>`;
    return;
   }
   const b = bodies[id];
   selectedChip.textContent = `Selected: ${b.data.name}`;
   // highlight
   b.mesh.material.emissive = new THREE.Color(0xffffff);
   b.mesh.material.emissiveIntensity = 0.18;
   // show info
   infoPane.innerHTML = `
    <h2>${b.data.name} <small style="font-size:12px;color:${'#9fb0bd'}">(${b.data.type})
</small></h2>
    <div class="small"><strong>Radius:</strong> ${b.data.radius_km.toLocaleString()} km<br/>
    <strong>Orbit:</strong> ${b.data.orbit_km || '—'} AU (visualized, scaled)<br/>
<br/>${b.data.descr}</div>
    <div style="margin-top:8px;"><button class="btn" id="focusBtn">Focus</button> <button
class="btn secondary" id="infoMore">More</button></div>
   `;
   document.getElementById('focusBtn').addEventListener('click', ()=>{
focusCameraOn(b.mesh.getWorldPosition(new THREE.Vector3()),
Math.max(b.data.scaleRadius*6, 60)); });
   document.getElementById('infoMore').addEventListener('click', ()=>{ alert(`${b.data.name}
— more info can be loaded (links, images).`); });
  }

  // camera focus tween (simple)
  let camTween = null;
  function focusCameraOn(targetPos, distance=120, duration=900){
   const fromPos = camera.position.clone();
```

```
  const fromTarget = controls.target.clone();
  const dir = new THREE.Vector3().subVectors(camera.position, controls.target).normalize();
  const toPos = targetPos.clone().add(dir.multiplyScalar(distance));
  const t0 = performance.now();
  camTween = (now)=>{
   const t = Math.min(1, (now - t0)/duration);
   camera.position.lerpVectors(fromPos, toPos, easeOutCubic(t));
   controls.target.lerpVectors(fromTarget, targetPos, easeOutCubic(t));
   if(t >= 1) camTween = null;
  };
 }
 function easeOutCubic(t){ return (--t)*t*t+1; }

 // build sidebar list
 function buildList(filter){
  listElem.innerHTML = '';
  rawData.filter(d => !filter ||
d.name.toLowerCase().includes(filter.toLowerCase())).forEach(d=>{
   const card = document.createElement('div');
   card.className = 'card';
   const color = '#444';
   card.innerHTML = `<div class="dot" style="background:${color};">${d.name[0]}</div>
    <div style="flex:1"><strong>${d.name}</strong><div class="small">${d.type} —
${d.descr.slice(0,70)}${d.descr.length>70?'...':''}</div></div>`;
   card.addEventListener('click', ()=> selectBody(d.id));
   listElem.appendChild(card);
  });
 }
 buildList();

 // labels update each frame
 function updateLabels(){
  const rect = renderer.domElement.getBoundingClientRect();
  Object.values(bodies).forEach(b=>{
   const pos = b.mesh.getWorldPosition(new THREE.Vector3());
   pos.project(camera);
   const x = (pos.x * 0.5 + 0.5) * rect.width;
   const y = (-pos.y * 0.5 + 0.5) * rect.height;
   const visible = pos.z < 1 && pos.z > -1;
   b.labelEl.style.display = visible ? 'flex' : 'none';
   b.labelEl.style.left = `${x}px`;
   b.labelEl.style.top = `${y}px`;
   // update subtitle with scaled orbit info
   const sub = b.data.orbit_km ? `${b.data.orbit_km} AU` : b.data.type;
```

```
    b.labelEl.querySelector('.sub').textContent = sub;
  });
 }

 // animation loop variables
 let running = true;
 let timeScale = 1;
 let lastTime = performance.now();

 function animate(now){
  const dt = (now - lastTime) * 0.001 * timeScale;
  lastTime = now;

  // orbit rotations and self-rotations
  Object.values(bodies).forEach(b=>{
   // orbit speed tuned for visibility
   if(b.data.orbit_km && b.data.orbit_km > 0) b.pivot.rotation.y += (0.05 /
Math.max(b.data.orbitalPeriod || 100, 1)) * dt * 60;
   b.mesh.rotation.y += (0.2 / Math.max(Math.abs(b.data.rotation || 1), 0.1)) * dt * 60;
   if(b.mesh.cloudMesh) b.mesh.cloudMesh.rotation.y += 0.02 * dt * 60;
  });

  // comet movement (simple path)
  const t = now * 0.00006 * timeScale;
  cometObj.mesh.position.set(Math.cos(t) * 740, Math.sin(t*0.6) * 180 + 140, Math.sin(t) * 420);
  cometObj.tail.position.copy(cometObj.mesh.position);

  // hover highlighting
  raycaster.setFromCamera(mouse, camera);
  const intersects = raycaster.intersectObjects(Object.values(bodies).map(b=>b.mesh));
  if(intersects.length){
   const id = intersects[0].object.userData.id;
   if(hovered !== id){
    if(hovered && bodies[hovered]) bodies[hovered].mesh.material.emissiveIntensity = 0;
    hovered = id;
    if(hovered && bodies[hovered]) { bodies[hovered].mesh.material.emissive = new
THREE.Color(0xffffff); bodies[hovered].mesh.material.emissiveIntensity = 0.08; }
   }
  } else if(hovered){
   if(bodies[hovered]) bodies[hovered].mesh.material.emissiveIntensity = 0;
   hovered = null;
  }

  controls.update();
```

```
  if(camTween) camTween(now);
  updateLabels();
  composer.render();
  if(running) requestAnimationFrame(animate);
 }

 requestAnimationFrame((t)=>{ lastTime = t; animate(t); });

 // resize handler
 function onResize(){
  const w = canvasWrap.clientWidth, h = canvasWrap.clientHeight;
  renderer.setSize(w, h);
  composer.setSize(w, h);
  camera.aspect = w / h;
  camera.updateProjectionMatrix();
 }
 window.addEventListener('resize', onResize);
 onResize();

 // controls wiring
 playBtn.addEventListener('click', ()=>{
  running = !running;
  if(running){ playBtn.textContent = 'Pause'; lastTime = performance.now();
requestAnimationFrame(animate); }
  else { playBtn.textContent = 'Play'; }
 });
 window.addEventListener('keydown', (e)=>{
  if(e.code === 'Space'){ e.preventDefault(); playBtn.click(); }
  if(e.key==='1') cameraPresetOverview();
  if(e.key==='2') cameraPresetClose('earth');
  if(e.key==='3') cameraPresetClose('jupiter');
 });

 speedSlider.addEventListener('input', ()=>{
  timeScale = parseFloat(speedSlider.value);
  timeChip.textContent = `Time: ${timeScale.toFixed(2)}×`;
 });

 // camera presets
 function cameraPresetOverview(){ focusCameraOn(new THREE.Vector3(0,0,0), 1200, 1000); }
 function cameraPresetClose(id){ if(bodies[id])
focusCameraOn(bodies[id].mesh.getWorldPosition(new THREE.Vector3()),
Math.max(bodies[id].data.scaleRadius*6, 60), 900); }
```

```
  // screenshot
  screenshotBtn.addEventListener('click', ()=>{
    renderer.domElement.toBlob((blob)=>{
      const a = document.createElement('a');
      a.href = URL.createObjectURL(blob);
      a.download = `universe-snapshot.png`;
      document.body.appendChild(a); a.click(); a.remove();
    });
  });

  // search
  searchInput.addEventListener('input', (e)=> buildList(e.target.value.trim()));

  // tour (sequence of waypoints)
  const tourWaypoints = [
    {pos:new THREE.Vector3(0,120,1200), target:new THREE.Vector3(0,0,0), text:'Overview: The Sun and its family.'},
    {pos:new THREE.Vector3(160,40,220), target:null, pick:'earth', text:'Earth: our home. Clouds and blue oceans.'},
    {pos:new THREE.Vector3(380,60,120), target:null, pick:'jupiter', text:'Jupiter: gas giant and the Great Red Spot.'},
    {pos:new THREE.Vector3(450,40,50), target:null, pick:'saturn', text:'Saturn: rings and many moons.'},
    {pos:new THREE.Vector3(600,120,-80), target:new THREE.Vector3(420,120,-80), text:'Voyager probe (model) in deep space.'},
  ];
  let tourActive = false, tourIndex = 0, tourTimeout;
  tourBtn.addEventListener('click', ()=>{ if(tourActive){ stopTour(); } else { startTour(); }});
  function startTour(){
    tourActive = true; tourBtn.textContent = 'Stop Tour'; tourIndex = 0; runTourStep();
  }
  function stopTour(){ tourActive = false; tourBtn.textContent = 'Start Tour'; if(tourTimeout) clearTimeout(tourTimeout); }
  function runTourStep(){
    if(!tourActive) return;
    const step = tourWaypoints[tourIndex++];
    if(!step){ stopTour(); return; }
    const target = step.pick ? bodies[step.pick].mesh.getWorldPosition(new THREE.Vector3()) : (step.target || new THREE.Vector3(0,0,0));
    focusCameraOn(target, 150 + (tourIndex*80), 1500);
    infoPane.innerHTML = `<h2>Tour</h2><div class="small">${step.text}</div>`;
    tourTimeout = setTimeout(()=>{ if(tourActive) runTourStep(); }, 3500);
  }
```

```
// story mode (timeline)
const storySteps = [
  {title:'4.6 Billion years ago', text:'Solar nebula collapses, Sun forms at center; planetesimals form.'},
  {title:'Accretion', text:'Planets grow by collisions — rocky worlds form near the Sun, gas giants farther out.'},
  {title:'Heavy bombardment', text:'Early Earth and other bodies were bombarded by asteroids and comets.'},
  {title:'Life emerges', text:'On Earth, life appears about 3.5 billion years ago.'},
  {title:'Human exploration', text:'In 20th-21st centuries, human-made probes explore the solar system.'}
];
let storyIndex = 0, storyAuto = false, storyInterval;
const storyText = document.getElementById('storyText');
document.getElementById('nextStory').addEventListener('click', ()=>{ nextStory(); });
document.getElementById('prevStory').addEventListener('click', ()=>{ prevStory(); });
document.getElementById('autoStory').addEventListener('click', ()=>{ storyAuto = !storyAuto; if(storyAuto){ document.getElementById('autoStory').textContent='Stop'; storyInterval = setInterval(nextStory, 4200);} else { document.getElementById('autoStory').textContent='Auto Play'; clearInterval(storyInterval);} });

function showStory(i){
  const s = storySteps[i];
  if(!s) return;
  storyText.innerHTML = `<strong>${s.title}</strong><div style="margin-top:6px" class="small">${s.text}</div>`;
  // focal camera suggestion: jump to interesting object for step
  if(i===1) cameraPresetClose('jupiter');
  if(i===3) cameraPresetClose('earth');
}
function nextStory(){ storyIndex = Math.min(storySteps.length-1, storyIndex+1); showStory(storyIndex); }
function prevStory(){ storyIndex = Math.max(0, storyIndex-1); showStory(storyIndex); }
showStory(0);

// quiz buttons
document.querySelectorAll('.quiz-btn').forEach(btn=>{
  btn.addEventListener('click', (e)=>{
    const q = e.currentTarget.dataset.q;
    const a = e.currentTarget.dataset.a;
    let ok=false;
    if(q==='1' && a==='Mars') ok=true;
    if(q==='2' && a==='Jupiter') ok=true;
    document.getElementById('r'+q).textContent = ok ? '✅ Correct' : '❌ Incorrect';
```

```
  });
});

// light tracking (sun position)
function updateSunLight(){
  const sun = bodies['sun'];
  if(sun) sunLight.position.copy(sun.mesh.getWorldPosition(new THREE.Vector3()));
}

// hover loop integrated in animate

// expose simple API in console for exploration
window.UniverseAPI = { bodies, focusCameraOn, cameraPresetOverview, cameraPresetClose };

// initial camera framing
cameraPresetOverview();

// accessibility focus style
Array.from(document.querySelectorAll('button,input')).forEach(el=>{
  el.addEventListener('focus', ()=> el.style.boxShadow='0 8px 30px rgba(124, 226, 185, 0.08)');
  el.addEventListener('blur', ()=> el.style.boxShadow='');
});

// End of app script
</script>
</body>
</html>
```