```
import { setupScene, setupResizeHandler } from './render-utils.js';

export function initBCCRender(containerId) {
  const { scene, camera, renderer } = setupScene(containerId);

  // Add OrbitControls for rotation
  const controls = new THREE.OrbitControls(camera, renderer.domElement);
  controls.target.set(1, 1, 1);

  // Add lights
  const pointLight1 = new THREE.PointLight(0xefffef, 1, 50);
  pointLight1.position.set(-10, -10, 10);
  scene.add(pointLight1);

  const pointLight2 = new THREE.PointLight(0xffefef, 1, 50);
  pointLight2.position.set(-10, 10, 10);
  scene.add(pointLight2);

  const pointLight3 = new THREE.PointLight(0xefefff, 1, 50);
  pointLight3.position.set(10, -10, 10);
  scene.add(pointLight3);

  const ambientLight = new THREE.AmbientLight(0xffffff, 0.4);
  scene.add(ambientLight);

  // BCC-specific code for creating spheres, disks, and edges...
    // Function to create a whole sphere with realistic material
    function createWholeSphere(radius, color) {
      const geometry = new THREE.SphereGeometry(radius, 32, 32);
      const material = new THREE.MeshStandardMaterial({
        color: color,
        roughness: 0.6,
        metalness: 0.1,
        side: THREE.DoubleSide
      });
      return new THREE.Mesh(geometry, material);
    }

    // Function to create a corner sphere (1/8 sphere) with realistic material
    function createCornerSphere(radius, color) {
      const geometry = new THREE.SphereGeometry(radius, 32, 16, Math.PI / 2, Math.PI / 2,
       0, Math.PI / 2);
      const material = new THREE.MeshStandardMaterial({
        color: color,
        roughness: 0.6,
        metalness: 0.1,
        side: THREE.DoubleSide
      });
      return new THREE.Mesh(geometry, material);
    }

    // Function to create a corner disk to cap corner sphere cut surfaces with realistic
    material
    function createCornerDisk(radius, color, rotation) {
      const geometry = new THREE.CircleGeometry(radius, 32, 0, Math.PI / 2);
      const material = new THREE.MeshStandardMaterial({
        color: color,
        roughness: 0.6,
        metalness: 0.1,
        side: THREE.DoubleSide
      });
      const disk = new THREE.Mesh(geometry, material);
      const quaternion = new THREE.Quaternion().setFromEuler(rotation);
      disk.applyQuaternion(quaternion);
```

```
    return disk;
  }

  // Function to create the corner sphere group
  function createCornerSphereGroup(radius, color) {
    const group = new THREE.Group();
    const sphere = createCornerSphere(radius, color);
    group.add(sphere);

    const cDisk1 = createCornerDisk(radius, color, new THREE.Euler(0, 0, 0));
    const cDisk2 = createCornerDisk(radius, color, new THREE.Euler(0, -Math.PI / 2, 0
    ));
    const cDisk3 = createCornerDisk(radius, color, new THREE.Euler(Math.PI / 2, 0, 0));

    group.add(cDisk1);
    group.add(cDisk2);
    group.add(cDisk3);

    return group;
  }

  // Function to create the edges for a cube with vertices from (0,0,0) to (1,1,1)
  function createCubeEdges() {
    const vertices = [
      [0, 0, 0], [1, 0, 0],
      [1, 0, 0], [1, 1, 0],
      [1, 1, 0], [0, 1, 0],
      [0, 1, 0], [0, 0, 0],
      [0, 0, 1], [1, 0, 1],
      [1, 0, 1], [1, 1, 1],
      [1, 1, 1], [0, 1, 1],
      [0, 1, 1], [0, 0, 1],
      [0, 0, 0], [0, 0, 1],
      [1, 0, 0], [1, 0, 1],
      [1, 1, 0], [1, 1, 1],
      [0, 1, 0], [0, 1, 1],
    ];

    const geometry = new THREE.BufferGeometry();
    const positionAttribute = new THREE.Float32BufferAttribute(vertices.flat(), 3);
    geometry.setAttribute('position', positionAttribute);

    const lineMaterial = new THREE.LineBasicMaterial({ color: 'gray', linewidth: 1 });
    return new THREE.LineSegments(geometry, lineMaterial);
  }

  // Create the unit cell group
  const unitCellGroup = new THREE.Group();

  // Define the radius of the spheres
  const radius = Math.sqrt(3) / 4;

  // Define positions and rotations for the eight corners of the unit cell
  const positions = [
    [0, 0, 0],
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1],
    [1, 1, 0],
    [1, 0, 1],
    [0, 1, 1],
    [1, 1, 1]
  ];

```

```
    const rotations = [
      new THREE.Quaternion().setFromEuler(new THREE.Euler(0, 0, 0)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(0, -Math.PI / 2, 0)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, 0, 0)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(0, Math.PI / 2, 0)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, -Math.PI / 2, 0)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(-Math.PI / 2, -Math.PI / 2, 0
      )),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, Math.PI / 2, Math.
      PI)),
      new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, 0, Math.PI))
    ];

    // Create and position the corner spheres in the unit cell
    positions.forEach((pos, index) => {
      const color = (index === 7) ? 0xff0000 : 0xFFCC00; // Red color for the atom at
      (1,1,1)
      const cornerSphereGroup = createCornerSphereGroup(radius, color);
      cornerSphereGroup.quaternion.copy(rotations[index]);
      cornerSphereGroup.position.set(pos[0], pos[1], pos[2]);
      unitCellGroup.add(cornerSphereGroup);
    });

    // Add cube edges to the unit cell group
    const cubeEdges = createCubeEdges();
    unitCellGroup.add(cubeEdges);

    // Add a whole sphere in the center of the unit cell
    const centerSphere = createWholeSphere(radius, 0xFFCC00); // Yellow color for the
    center sphere
    centerSphere.position.set(0.5, 0.5, 0.5);
    unitCellGroup.add(centerSphere);

    // Add the unit cell group to the scene
    scene.add(unitCellGroup);

    // Function to update the distances between unit cell groups
    function updateDistances(distance) {
      const translations = [
        [2 * distance, 0, 0],
        [0, 0, 2 * distance],
        [2 * distance, 0, 2 * distance],
        [0, 2 * distance, 0]
      ];
      translations.forEach((trans, index) => {
        const clonedGroup = clonedGroups[index];
        clonedGroup.position.set(trans[0], trans[1], trans[2]);
      });
    }

    // Clone and translate copies of the unit cell group
    const clonedGroups = [];
    const distanceSlider = document.getElementById('distanceSlider');
    const initialDistance = distanceSlider.value;

    const initialTranslations = [
      [2 * initialDistance, 0, 0],
      [0, 0, 2 * initialDistance],
      [2 * initialDistance, 0, 2 * initialDistance],
      [0, 2 * initialDistance, 0]
    ];

    const rotationAngles = [
      new THREE.Euler(0, -Math.PI / 2, 0),
```

```
        new THREE.Euler(0, Math.PI / 2, 0),
        new THREE.Euler(0, Math.PI, 0),
        new THREE.Euler(0, 0, -Math.PI / 2)
      ];

      initialTranslations.forEach((trans, index) => {
        const clonedGroup = unitCellGroup.clone();
        clonedGroup.position.set(trans[0], trans[1], trans[2]);
        const quaternion = new THREE.Quaternion().setFromEuler(rotationAngles[index]);
        clonedGroup.applyQuaternion(quaternion);
        scene.add(clonedGroup);
        clonedGroups.push(clonedGroup);
      });

      // Event listener for the distance slider
      distanceSlider.addEventListener('input', (event) => {
        const distance = event.target.value;
        updateDistances(distance);
      });

      // Function to update the visibility of unit cell groups based on checkbox state
      function updateVisibility() {
        clonedGroups.forEach((group, index) => {
          const checkbox = document.getElementById(`cell${index + 1}`);
          group.visible = checkbox.checked;
        });
      }

      // Add event listeners for the checkboxes
      const checkboxes = document.querySelectorAll('.controls-container
      input[type="checkbox"]');
      checkboxes.forEach(checkbox => {
        checkbox.addEventListener('change', updateVisibility);
      });

      // Set the camera position
      const mag = 5;
      const initialCameraPosition = new THREE.Vector3(mag * 0.75, mag * 0.75, mag);
      camera.position.copy(initialCameraPosition);

      // Function to reset the camera position and show all cells
      function resetCameraPosition() {
        camera.position.copy(initialCameraPosition);
        controls.update();
        checkboxes.forEach(checkbox => {
          checkbox.checked = true;
        });
        updateVisibility();
        distanceSlider.value = initialDistance;
        updateDistances(initialDistance);
      }

      // Add event listener for the reset view button
      const resetViewButton = document.getElementById('resetViewButton');
      resetViewButton.addEventListener('click', resetCameraPosition);

      // Update distances initially
      updateDistances(initialDistance);

    // Set initial camera position
    camera.position.set(5, 5, 5);
    camera.lookAt(controls.target);

    // Animation loop
```

```
    function animate() {
      requestAnimationFrame(animate);
      controls.update();
      renderer.render(scene, camera);
    }
    animate();

    setupResizeHandler(camera, renderer, containerId);
  }
```