

```

1  import { setupScene, setupResizeHandler } from './render-utils.js';
2
3  export function initBCCRender(containerId) {
4      const { scene, camera, renderer } = setupScene(containerId);
5
6      // Add OrbitControls for rotation
7      const controls = new THREE.OrbitControls(camera, renderer.domElement);
8      controls.target.set(1, 1, 1);
9
10     // Add lights
11     const pointLight1 = new THREE.PointLight(0xefffef, 1, 50);
12     pointLight1.position.set(-10, -10, 10);
13     scene.add(pointLight1);
14
15     const pointLight2 = new THREE.PointLight(0xffefef, 1, 50);
16     pointLight2.position.set(-10, 10, 10);
17     scene.add(pointLight2);
18
19     const pointLight3 = new THREE.PointLight(0xefefff, 1, 50);
20     pointLight3.position.set(10, -10, 10);
21     scene.add(pointLight3);
22
23     const ambientLight = new THREE.AmbientLight(0xffffff, 0.4);
24     scene.add(ambientLight);
25
26     // BCC-specific code for creating spheres, disks, and edges...
27     // Function to create a whole sphere with realistic material
28     function createWholeSphere(radius, color) {
29         const geometry = new THREE.SphereGeometry(radius, 32, 32);
30         const material = new THREE.MeshStandardMaterial({
31             color: color,
32             roughness: 0.6,
33             metalness: 0.1,
34             side: THREE.DoubleSide
35         });
36         return new THREE.Mesh(geometry, material);
37     }
38
39     // Function to create a corner sphere (1/8 sphere) with realistic material
40     function createCornerSphere(radius, color) {
41         const geometry = new THREE.SphereGeometry(radius, 32, 16, Math.PI / 2, Math.PI / 2,
42             0, Math.PI / 2);
43         const material = new THREE.MeshStandardMaterial({
44             color: color,
45             roughness: 0.6,
46             metalness: 0.1,
47             side: THREE.DoubleSide
48         });
49         return new THREE.Mesh(geometry, material);
50     }
51
52     // Function to create a corner disk to cap corner sphere cut surfaces with realistic
53     // material
54     function createCornerDisk(radius, color, rotation) {
55         const geometry = new THREE.CircleGeometry(radius, 32, 0, Math.PI / 2);
56         const material = new THREE.MeshStandardMaterial({
57             color: color,
58             roughness: 0.6,
59             metalness: 0.1,
60             side: THREE.DoubleSide
61         });
62         const disk = new THREE.Mesh(geometry, material);
63         const quaternion = new THREE.Quaternion().setFromEuler(rotation);
64         disk.applyQuaternion(quaternion);

```

```

63     return disk;
64 }
65
66 // Function to create the corner sphere group
67 function createCornerSphereGroup(radius, color) {
68     const group = new THREE.Group();
69     const sphere = createCornerSphere(radius, color);
70     group.add(sphere);
71
72     const cDisk1 = createCornerDisk(radius, color, new THREE.Euler(0, 0, 0));
73     const cDisk2 = createCornerDisk(radius, color, new THREE.Euler(0, -Math.PI / 2, 0));
74     const cDisk3 = createCornerDisk(radius, color, new THREE.Euler(Math.PI / 2, 0, 0));
75
76     group.add(cDisk1);
77     group.add(cDisk2);
78     group.add(cDisk3);
79
80     return group;
81 }
82
83 // Function to create the edges for a cube with vertices from (0,0,0) to (1,1,1)
84 function createCubeEdges() {
85     const vertices = [
86         [0, 0, 0], [1, 0, 0],
87         [1, 0, 0], [1, 1, 0],
88         [1, 1, 0], [0, 1, 0],
89         [0, 1, 0], [0, 0, 0],
90         [0, 0, 1], [1, 0, 1],
91         [1, 0, 1], [1, 1, 1],
92         [1, 1, 1], [0, 1, 1],
93         [0, 1, 1], [0, 0, 1],
94         [0, 0, 0], [0, 0, 1],
95         [1, 0, 0], [1, 0, 1],
96         [1, 1, 0], [1, 1, 1],
97         [0, 1, 0], [0, 1, 1],
98     ];
99
100     const geometry = new THREE.BufferGeometry();
101     const positionAttribute = new THREE.Float32BufferAttribute(vertices.flat(), 3);
102     geometry.setAttribute('position', positionAttribute);
103
104     const lineMaterial = new THREE.LineBasicMaterial({ color: 'gray', linewidth: 1 });
105     return new THREE.LineSegments(geometry, lineMaterial);
106 }
107
108 // Create the unit cell group
109 const unitCellGroup = new THREE.Group();
110
111 // Define the radius of the spheres
112 const radius = Math.sqrt(3) / 4;
113
114 // Define positions and rotations for the eight corners of the unit cell
115 const positions = [
116     [0, 0, 0],
117     [1, 0, 0],
118     [0, 1, 0],
119     [0, 0, 1],
120     [1, 1, 0],
121     [1, 0, 1],
122     [0, 1, 1],
123     [1, 1, 1]
124 ];
125

```

```

126 const rotations = [
127   new THREE.Quaternion().setFromEuler(new THREE.Euler(0, 0, 0)),
128   new THREE.Quaternion().setFromEuler(new THREE.Euler(0, -Math.PI / 2, 0)),
129   new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, 0, 0)),
130   new THREE.Quaternion().setFromEuler(new THREE.Euler(0, Math.PI / 2, 0)),
131   new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, -Math.PI / 2, 0)),
132   new THREE.Quaternion().setFromEuler(new THREE.Euler(-Math.PI / 2, -Math.PI / 2, 0
    )),
133   new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, Math.PI / 2, Math.
    PI)),
134   new THREE.Quaternion().setFromEuler(new THREE.Euler(Math.PI / 2, 0, Math.PI))
135 ];
136
137 // Create and position the corner spheres in the unit cell
138 positions.forEach((pos, index) => {
139   const color = (index === 7) ? 0xff0000 : 0xFFCC00; // Red color for the atom at
    (1,1,1)
140   const cornerSphereGroup = createCornerSphereGroup(radius, color);
141   cornerSphereGroup.quaternion.copy(rotations[index]);
142   cornerSphereGroup.position.set(pos[0], pos[1], pos[2]);
143   unitCellGroup.add(cornerSphereGroup);
144 });
145
146 // Add cube edges to the unit cell group
147 const cubeEdges = createCubeEdges();
148 unitCellGroup.add(cubeEdges);
149
150 // Add a whole sphere in the center of the unit cell
151 const centerSphere = createWholeSphere(radius, 0xFFCC00); // Yellow color for the
    center sphere
152 centerSphere.position.set(0.5, 0.5, 0.5);
153 unitCellGroup.add(centerSphere);
154
155 // Add the unit cell group to the scene
156 scene.add(unitCellGroup);
157
158 // Function to update the distances between unit cell groups
159 function updateDistances(distance) {
160   const translations = [
161     [2 * distance, 0, 0],
162     [0, 0, 2 * distance],
163     [2 * distance, 0, 2 * distance],
164     [0, 2 * distance, 0]
165   ];
166   translations.forEach((trans, index) => {
167     const clonedGroup = clonedGroups[index];
168     clonedGroup.position.set(trans[0], trans[1], trans[2]);
169   });
170 }
171
172 // Clone and translate copies of the unit cell group
173 const clonedGroups = [];
174 const distanceSlider = document.getElementById('distanceSlider');
175 const initialDistance = distanceSlider.value;
176
177 const initialTranslations = [
178   [2 * initialDistance, 0, 0],
179   [0, 0, 2 * initialDistance],
180   [2 * initialDistance, 0, 2 * initialDistance],
181   [0, 2 * initialDistance, 0]
182 ];
183
184 const rotationAngles = [
185   new THREE.Euler(0, -Math.PI / 2, 0),

```

```

186     new THREE.Euler(0, Math.PI / 2, 0),
187     new THREE.Euler(0, Math.PI, 0),
188     new THREE.Euler(0, 0, -Math.PI / 2)
189 ];
190
191 initialTranslations.forEach((trans, index) => {
192     const clonedGroup = unitCellGroup.clone();
193     clonedGroup.position.set(trans[0], trans[1], trans[2]);
194     const quaternion = new THREE.Quaternion().setFromEuler(rotationAngles[index]);
195     clonedGroup.applyQuaternion(quaternion);
196     scene.add(clonedGroup);
197     clonedGroups.push(clonedGroup);
198 });
199
200 // Event listener for the distance slider
201 distanceSlider.addEventListener('input', (event) => {
202     const distance = event.target.value;
203     updateDistances(distance);
204 });
205
206 // Function to update the visibility of unit cell groups based on checkbox state
207 function updateVisibility() {
208     clonedGroups.forEach((group, index) => {
209         const checkbox = document.getElementById(`cell${index + 1}`);
210         group.visible = checkbox.checked;
211     });
212 }
213
214 // Add event listeners for the checkboxes
215 const checkboxes = document.querySelectorAll('.controls-container
input[type="checkbox"]');
216 checkboxes.forEach(checkbox => {
217     checkbox.addEventListener('change', updateVisibility);
218 });
219
220 // Set the camera position
221 const mag = 5;
222 const initialCameraPosition = new THREE.Vector3(mag * 0.75, mag * 0.75, mag);
223 camera.position.copy(initialCameraPosition);
224
225 // Function to reset the camera position and show all cells
226 function resetCameraPosition() {
227     camera.position.copy(initialCameraPosition);
228     controls.update();
229     checkboxes.forEach(checkbox => {
230         checkbox.checked = true;
231     });
232     updateVisibility();
233     distanceSlider.value = initialDistance;
234     updateDistances(initialDistance);
235 }
236
237 // Add event listener for the reset view button
238 const resetViewButton = document.getElementById('resetViewButton');
239 resetViewButton.addEventListener('click', resetCameraPosition);
240
241 // Update distances initially
242 updateDistances(initialDistance);
243
244 // Set initial camera position
245 camera.position.set(5, 5, 5);
246 camera.lookAt(controls.target);
247
248 // Animation loop

```

```
249     function animate() {
250         requestAnimationFrame(animate);
251         controls.update();
252         renderer.render(scene, camera);
253     }
254     animate();
255
256     setupResizeHandler(camera, renderer, containerId);
257 }
```