ICCS404: Week09 – Marker-based AR

Part 1: Marker Detection (45 minutes)

1.1 Basic Setup

Create `marker_tester.html`:

```html
<!DOCTYPE html>
<html>
  <head>
    <title>A-Frame AR.js Marker Tester</title>
    <meta charset="utf-8" />
    <!-- A-Frame: Core 3D/VR/AR Framework -->
    <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
    <!-- AR.js: Adds AR Capabilities -->
    <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script>
  </head>

  <body style="margin: 0; overflow: hidden;">
    <!-- Scene Configuration -->
    <a-scene
      embedded
      vr-mode-ui="enabled:false"
      arjs="sourceType: webcam;
            detectionMode: mono_and_matrix;
            matrixCodeType: 3x3_PARITY65">

      <!-- Single Marker Test -->
      <a-marker type="barcode" value="0">
        <a-box position="0 0.5 0" color="red" opacity="0.5"></a-box>
      </a-marker>

      <!-- Camera -->
      <a-entity camera></a-entity>
    </a-scene>
  </body>
</html>
```

1.2 Understanding the Code

Let's break down each major component:

1. **Scene Configuration**

```html
<a-scene embedded vr-mode-ui="enabled:false"
         arjs="sourceType: webcam; detectionMode: mono_and_matrix;
               matrixCodeType: 3x3_PARITY65">
```

- `embedded`: Fits AR scene within webpage
- `vr-mode-ui="enabled:false"`: Disables VR mode
- `arjs` parameters:
    - `sourceType: webcam`: Uses live camera feed
    - `detectionMode: mono_and_matrix`: Enables marker detection
    - `matrixCodeType: 3x3_PARITY65`: Specifies marker format

2. **Marker Definition**

```
<a-marker type="barcode" value="0">
  <a-box position="0 0.5 0" color="red" opacity="0.5"></a-box>
</a-marker>
```

Position breakdown:

```
position="0 0.5 0"
          │ │   │
          │ │   └── z (forward/backward)
          │ └───── y (up/down) 0.5 units up
          └─────── x (left/right)
```

Exercise 1: Single Marker Interaction

Modify the box properties to understand positioning:

1. Try different positions:

```
<!-- Test these positions -->
position="1 0.5 0"    <!-- Right -->
position="0 1.5 0"    <!-- Higher -->
position="0 0.5 1"    <!-- Forward -->
```

2. Try different shapes:

```
<!-- Replace a-box with -->
<a-sphere position="0 0.5 0" color="red" radius="0.5" opacity="0.5"></a-sphere>
<!-- or -->
<a-cylinder position="0 0.5 0" color="red" radius="0.5" opacity="0.5"></a-cylinder>
```

Exercise 2: Multiple Markers

Add these additional markers to your scene:

```
<!-- Marker 1: Green -->
<a-marker type="barcode" value="1">
  <a-box position="0 0.5 0" color="green" opacity="0.5"></a-box>
</a-marker>

<!-- Marker 2: Blue -->
<a-marker type="barcode" value="2">
  <a-box position="0 0.5 0" color="blue" opacity="0.5"></a-box>
</a-marker>
```

Continue with markers 3-8 using these colors:

- Marker 3: magenta
- Marker 4: cyan
- Marker 5: yellow
- Marker 6: white
- Marker 7: grey
- Marker 8: black

Part 2: Line Drawing (60 minutes)

Now we'll create lines between markers, introducing key concepts in stages.

2.1 Marker Visibility Tracking

Create `line_drawing.html` starting with visibility tracking:

```
/**
 * Understanding Marker States
 *
 * Marker visibility can be:
 * ┌──────────┐
 * │ Found    │──> markerFound event ──> visible = true
 * │ Lost     │──> markerLost event  ──> visible = false
 * └──────────┘
 */

// Global visibility state
const markersVisibility = { marker0: false, marker1: false };

// Component to track marker visibility
AFRAME.registerComponent("marker-visibility-handler", {
  init: function () {
    const markerElement = this.el;

    // When marker is detected
    markerElement.addEventListener("markerFound", () => {
      markersVisibility[markerElement.id] = true;
      console.log(`${markerElement.id} found`);
    });

    // When marker is lost
    markerElement.addEventListener("markerLost", () => {
      markersVisibility[markerElement.id] = false;
      console.log(`${markerElement.id} lost`);
    });
  }
});
```

2.2 Understanding Three.js Concepts

Before implementing line drawing, let's understand key Three.js concepts:

```
/**
 * Three.js Key Concepts
 *
 * 1. Vector3: Represents a point in 3D space
 *    new THREE.Vector3(x, y, z)
 *
 * 2. BufferGeometry: Holds geometry data
 *    └── position attribute: Array of coordinates
 *
 * 3. Position Array Structure:
 *    For a line between points A and B:
 *    [Ax, Ay, Az,  Bx, By, Bz]
 *     0   1   2    3   4   5
 *
 * Example:
 * Point A(1,0,0) to Point B(0,1,0):
 * [1, 0, 0,  0, 1, 0]
 */
```

2.3 Line Drawing Component

Now let’s implement the line drawer:

```
AFRAME.registerComponent("line-drawer", {
  schema: {
    marker0: { type: 'selector' },
    marker1: { type: 'selector' }
  },

  init: function () {
    // Create line material
    const material = new THREE.LineBasicMaterial({
      color: 0xffffff,    // White color
      linewidth: 5,       // Line thickness
      opacity: 0.8,       // Slight transparency
      transparent: true   // Enable transparency
    });

    // Create line geometry
    const geometry = new THREE.BufferGeometry();
    const positions = new Float32Array(6); // 2 points × 3 coordinates
    geometry.setAttribute('position',
      new THREE.BufferAttribute(positions, 3));

    // Create the line
    this.line = new THREE.Line(geometry, material);
    this.el.object3D.add(this.line);

    // Initialize position trackers
    this.marker0Pos = new THREE.Vector3();
    this.marker1Pos = new THREE.Vector3();
  },

  tick: function () {
    if (!this.data.marker0 || !this.data.marker1) return;

    if (markersVisibility.marker0 && markersVisibility.marker1) {
      // Get current marker positions
      this.data.marker0.object3D.getWorldPosition(this.marker0Pos);
      this.data.marker1.object3D.getWorldPosition(this.marker1Pos);

      // Update line positions
      const positions = this.line.geometry.attributes.position.array;

      // Start point
      positions[0] = this.marker0Pos.x;
      positions[1] = this.marker0Pos.y;
      positions[2] = this.marker0Pos.z;

      // End point
      positions[3] = this.marker1Pos.x;
      positions[4] = this.marker1Pos.y;
      positions[5] = this.marker1Pos.z;

      // Tell Three.js to update
      this.line.geometry.attributes.position.needsUpdate = true;
      this.line.visible = true;
    } else {
      this.line.visible = false;
    }
  }
});
```

Exercise 5: Scene Setup

Set up the scene to read any two of the provided markers (e.g. 0 and 8):

```
<a-scene
  embedded
  vr-mode-ui="enabled:false"
  arjs="sourceType: webcam;
        detectionMode: mono_and_matrix;
        matrixCodeType: 3x3_PARITY65">

  <!-- Line drawing container -->
  <a-entity line-drawer="marker0: #marker0; marker1: #marker1"></a-entity>

  <!-- Marker 0: Red box -->
  <a-marker type="barcode" value="0" id="marker0" marker-visibility-handler>
    <a-box position="0 0.5 0" color="red" opacity="0.5"></a-box>
  </a-marker>

  <!-- Marker 8: Blue box -->
  <a-marker type="barcode" value="8" id="marker1" marker-visibility-handler>
    <a-box position="0 0.5 0" color="blue" opacity="0.5"></a-box>
  </a-marker>

  <!-- Camera -->
  <a-entity camera></a-entity>
</a-scene>
```

Part 3: Triangle Drawing (45 minutes)

Overview

Building on our line drawing experience, we'll now create a triangle that connects three markers. We'll use markers 0, 2, and 8 specifically.

3.1 Scene Setup

Create `triangle_drawing.html` with this marker configuration:

```
<a-scene embedded
  vr-mode-ui="enabled:false"
  arjs="sourceType: webcam; detectionMode: mono_and_matrix; matrixCodeType: 3x3_PARITY65">

  <!-- Marker 0: Red Vertex -->
  <a-marker type="barcode" id="marker0" value="0" marker-visibility-handler>
    <a-box position="0 0.5 0" color="red" opacity="0.5"></a-box>
  </a-marker>

  <!-- Marker 2: Green Vertex -->
  <a-marker type="barcode" id="marker1" value="2" marker-visibility-handler>
    <a-box position="0 0.5 0" color="green" opacity="0.5"></a-box>
  </a-marker>

  <!-- Marker 8: Blue Vertex -->
  <a-marker type="barcode" id="marker2" value="8" marker-visibility-handler>
    <a-box position="0 0.5 0" color="blue" opacity="0.5"></a-box>
  </a-marker>

  <!-- Triangle drawer entity -->
  <a-entity
    triangle-drawer="marker0: #marker0; marker1: #marker1; marker2: #marker2">
  </a-entity>

  <!-- Camera -->
  <a-entity camera></a-entity>
</a-scene>
```

3.2 Understanding Triangle Geometry

Before implementation, let’s understand how triangles differ from lines:

```
/**
 * Triangle Structure in Three.js
 *
 * Points:        Marker0 (Red)
 *                   •
 *                  / \
 *                 /   \
 *                /     \
 *               /       \
 *              •─────────•
 *    Marker1 (Green)    Marker2 (Blue)
 *
 * Position Array Structure:
 * [M0x, M0y, M0z,  M1x, M1y, M1z,  M2x, M2y, M2z]
 *  0    1    2     3    4    5     6    7    8
 *
 * Example triangle vertices:
 * Marker0: (0, 1, 0)  // Top
 * Marker1: (-1, 0, 0) // Bottom Left
 * Marker2: (1, 0, 0)  // Bottom Right
 */
```

3.3 Visibility Tracking

```
// Global visibility state for three markers
const markersVisibility = {
    marker0: false,
    marker1: false,
    marker2: false
};

// Same visibility handler from before
AFRAME.registerComponent("marker-visibility-handler", {
  init: function () {
    const markerElement = this.el;
    markerElement.addEventListener("markerFound", () => {
      markersVisibility[markerElement.id] = true;
      console.log(`${markerElement.id} found`);
    });

    markerElement.addEventListener("markerLost", () => {
      markersVisibility[markerElement.id] = false;
      console.log(`${markerElement.id} lost`);
    });
  }
});
```

3.4 Triangle Component Base

```
AFRAME.registerComponent("triangle-drawer", {
  schema: {
    marker0: { type: 'selector' }, // Red vertex
    marker1: { type: 'selector' }, // Green vertex
    marker2: { type: 'selector' }  // Blue vertex
  },

  init: function () {
    // Create triangle geometry
    const positions = new Float32Array([
      0, 0, 0, // Vertex 0 (Red)
      0, 0, 0, // Vertex 1 (Green)
      0, 0, 0  // Vertex 2 (Blue)
    ]);

    // Set distinct colors for each vertex
    const colors = new Float32Array([
      1, 0, 0, // Red
      0, 1, 0, // Green
      0, 0, 1  // Blue
    ]);

    // Create and configure geometry
    const geometry = new THREE.BufferGeometry();
    geometry.setAttribute('position', new THREE.BufferAttribute(positions, 3));
    geometry.setAttribute('color', new THREE.BufferAttribute(colors, 3));

    // Create material with vertex colors
    const material = new THREE.MeshBasicMaterial({
      vertexColors: true,
      side: THREE.DoubleSide,
      transparent: true,
      opacity: 0.5
    });

    // Create and add mesh to scene
    this.triangleMesh = new THREE.Mesh(geometry, material);
    this.el.object3D.add(this.triangleMesh);

    // Initialize position vectors
    this.marker0Pos = new THREE.Vector3();
    this.marker1Pos = new THREE.Vector3();
    this.marker2Pos = new THREE.Vector3();
  }
});
```

Exercise 6: Implement tick Function

Now implement the tick function for the triangle-drawer component:

```
tick: function () {
  /**
   * Your Task: Implement the tick function that:
   * 1. Checks if all markers exist
   * 2. Verifies all markers are visible
   * 3. Updates vertex positions based on marker positions
   * 4. Shows/hides triangle appropriately
   *
   * Use the line-drawer's tick function as reference
   * Remember to update all 9 position values (3 vertices × 3 coordinates)
   */
}
```

Implementation Tips

2. Pattern from line-drawer:

```
if (markersVisibility.marker0 && markersVisibility.marker1 && markersVisibility.marker2) {
  // Update positions
} else {
  // Hide triangle
}
```

3. Position Update Pattern:

```
this.data.marker0.object3D.getWorldPosition(this.marker0Pos);
// Then update corresponding position array elements
```

3. Remember to mark geometry for update:

```
this.triangleMesh.geometry.attributes.position.needsUpdate = true;
```

Lab Submission

- You can submit your lab individually or as a pair.
- Save your completed code of *Part 3: Triangle Drawing* including the working tick function (*Exercise* 6) as a single HTML file.
- Name the file as *week09_firstname.html* or *week09_firstname1_firstname2.html*, all in lowercase (example: *week09_pisut_tanaboon.html*).
- Submit the file through Google Classroom.
- Do not submit other files.
- Do not forget to add your names as a comment at the beginning of your code.