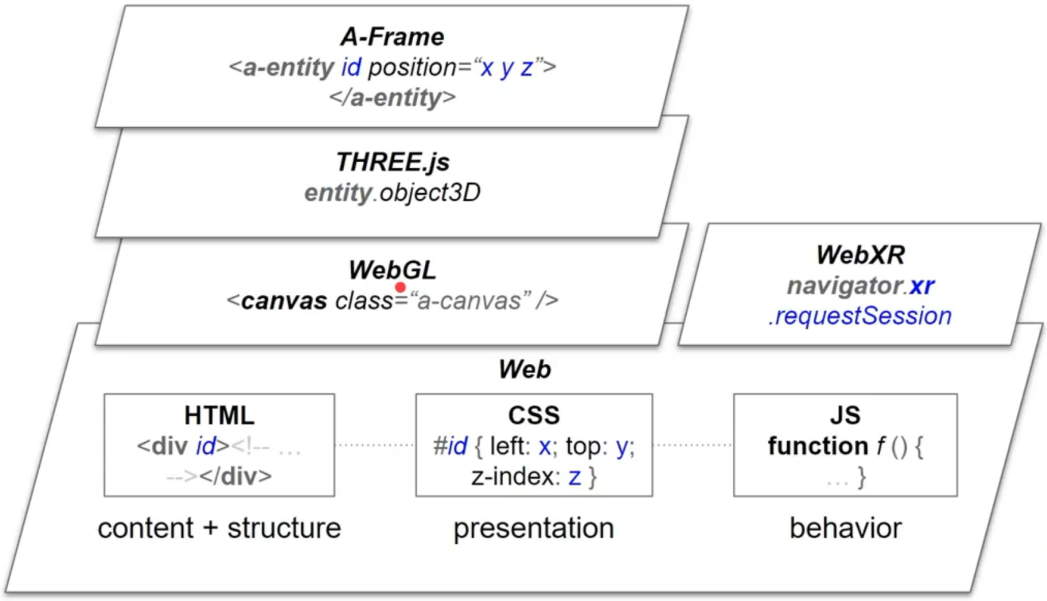
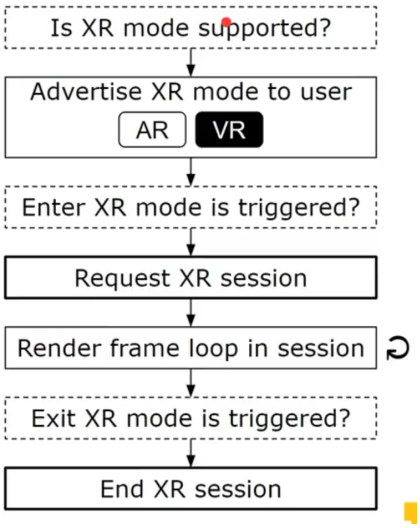
Week 1: XR Development Approaches

1. XR Development Approaches:
    - Web vs A-Frame:
        - Web:
            - Structure & Content: HTML (Organization of page content & hierarchy).
            - Presentation: CSS (Definition of page content presentation).
            - Behavior: JavaScript (Specification of interactive behavior).
        - A-Frame:
            - Structure: Entities (Organization of 3D scene hierarchy).
            - Content & Presentation: Components (Definition of 3D scene content & presentation).
            - Behavior: Systems & Scripts (Specification of interactive behavior).
    - WebXR:



Enable XR applications on the web by allowing pages to:

- **Detect** XR capabilities
- **Query** XR device capabilities
- **Poll** XR device and associated input device state
- **Display** imagery on XR device at the appropriate frame rate



    - WebXR vs Unity vs Unreal:
        - Unity:

        - Unity is a game engine and XR dev platform:
            - De facto standard for XR apps.
            - Increasingly built-in support.
            - Most "XR people" will as you about your Unity skills.
        - Support for all XR devices:
            - Basically all AR and VR device vendors provide Unity SDKs.
    - A-Frame:
        - A-Frame is a declarative WebXR framework:
            - Emerging XR app development framework on top of THREE.js
            - Good for novice XR designers with web dev background.
        - Support for most XR devices:
            - Full WebXR support in Firefox, Chrome and Oculus Browser.
    - Unreal:
        - Unreal offers high-quality visuals straight out of the box while Unity won't produce quite the same quality.
        - Developers often prefer Unity while designers opt for Unreal.
        - Unity enables to create complex project for low-end devices while Unreal requires a powerful PC and broadband internet setup.
        - Unreal supports XR but there is some overhead while Unity has native XR.

2. XR Development Toolkits:
    - A-Frame: WebXR, Marker-less AR, ARCore
    - AR.js: WebXR, Marker-based AR, Webcam
    - SteamVR: Unity
    - MRTK: Unity, Unreal, ARCore, ARKit, HoloLens
    - Vuforia: Unity, Webcam, ARCore, ARKit, HoloLens
    - AR Foundation: Unity, ARCore, ARKit, HoloLens
    - XR interaction: Unity, ARCore, ARKit
3. Toolbox:
    - For VR:
        - From 2D to 3D: 3D objects & transforms, working with 3D models, 3D interactions.
        - Designing for VR: environmental design, lights & shadows, animations, spatial sound.
        - Basic VR Interactions: travel, object selection, gaze, gesture, speech input.
        - Advanced VR Interactions: menu design, object manipulation, physics & particle systems.
    - For AR:
        - From VR to AR: types of AR displays, registration, tracking and calibration, mapping physical to virtual objects.
        - Designing for AR: motion tracking, environmental understanding, light estimation.
        - Marker-based AR: marker-based design, marker-based interactions, from marker-based to marker-less.
        - Marker-less AR: marker-less design, marker-less interactions, from hand-held to head worn.
4. From 2D to 3D:
    - 3D objects:
        - Geometry: position (x,y,z), size (width,height,depth), renderOrder: n, rotation (Xdeg, Ydeg, Zdeg).

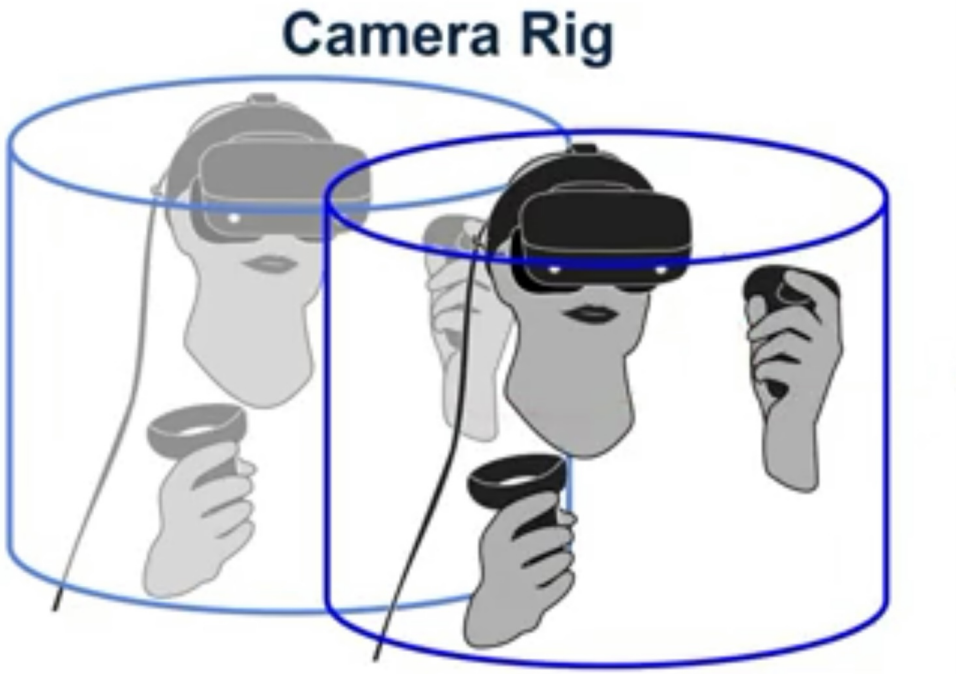
        - Material: visible [**true**, false], opacity [0,...,**1.0**], color: RGBa, src: texture, side (**front**, back, double), wireframe [true, **false**].
        - Transform = position, rotation, scale.
        - Materials are enhancement of texture mapping.
        - **Texture mapping** is a graphic design process in which a two-dimensional (2D) surface, called a **texture map**, is "wrapped around" a three-dimensional (3D)object.
    - Coordinate system: Unity: left-handed, Unreal: right-handed.
    - 3D interactions:
        - clicking in 3D: cast ray from mouse into scene, return all intersected elements.
    - 3D models:
        - Components:
            - Geometry/mesh/skin: Vertex position, normals and faces, texture coordinates.
            - Material: shader, bump/normal ... maps.
            - Texture: Image sprites.
            - Skeleton / rig & animations: Bones, poses/keyframes.
        - 3D model file formats:
            - glTF: common on web.
            - fbx: common in Unity and Unreal.
            - dae (digital asset exchange)
            - obj (wavefront object file)
            - mtl (material file with obj)
        - Places to find 3D model: Sketchfab, Google Poly, SketchUp's 3D Warehouse, Clara.io, Adobe Mixamo.
        - Places to create 3D models: Blender, Tinkercad, Maya or Maya LT, 3ds Max, Cinema4D.
    - From 360 to 3D:
        - 3D objects have width, height and depth while 360 has no depth.
        - 3D objects are positioned along x, y & z axes while 360 is represented in 2D.
        - 3D objects react to light, cast shadows and can be interactive while 360 photos and videos are quite immersive and can also be interactive.
5. First steps:
    - First step in WebXR:
        - A-Frame Development Environment: Glitch, CodePen, GitHub.
        - A-Frame and VR: A-Frame (Development Environment) => WebXR (Browser) => VR (Device)
        - A-Frame and AR: A-Frame (Development Environment) => WebXR (Browser) => AR (Device with ARCore)
    - First step in Unity:
        - AR Foundation for marker-less AR and Vuforia for marker-based AR.
        - Vuforia:
            - obtained from package manager
            - We can create AR camera and marker image and use webcam to display AR.
        - AR Foundation:
            - Insert an AR Session Origin, which gives an AR camera at the beginning.
            - Switch Android as default device.
            - Enable ARCore in Build Settings.

Week 2: Developing VR Applications

1. Designing a Virtual Reality:
    - Virtual Reality Content:
        - 3D models and animations: The 3D characters of your story. The main "ingredients". Can be in the foreground and background.
        - Lights and shadows: Required for rendering. But also key to creating atmosphere and guiding users (visual cues).
        - Environment: The background content. Vital for immersion and the user's feeling of being present in a virtual world.
        - Physics: Make or break the VR experience. More believable if objects behave similar to the real world.
        - Spatial sound: Key to immersion. VR experiences with sound are more believable (audio cues).
        - Menus and HUDs: Work differently in VR. Allow users to trigger functions, navigate and customize the VR experience.
    - Virtual Reality Interactions:
        - Selection (or pointing): To activate virtual objects:
            - Virtual hand: use controller as hand proxy to activate object.
            - Laser pointer: cast ray from controller to activate object.
        - Manipulation (or grasping): To manipulate (change) objects:
            - Hover: focus when in range.
            - Drag: move and rotate object.
            - Resize: scale while grabbing.
        - Navigation (or travel): To move through the virtual world:
            - Gaze-based: use gaze (head/eye) to activate waypoint.
            - Teleporting: use controller to activate travel function.
        - Menus: To travel/customize the experience:
            - Fixed: anchored in world.
            - Following: keep in user's view.
            - Attached: mount on head/hand.
    - Object placement:
        - World Space/ Diegetic:
            - Objects exists within the 3D world.
            - Anchored in a fixed position in the 3D world.
        - DOM/Page:
            - Object in 2D in the web page DOM/canvas of the mobile screen.
            - Separate from the canvas that renders the 3D scene.
        - Screen Space/Non-Diegetic:
            - Object in 2D over the 3D scene.
            - Think Heads-Up Display (HUD) in VR/2D screen overlay in AR.
        - Attached:
            - Combination of the above.
            - Object is anchored within the user's gaze/controller/hand.
    - Object Positioning and Sizing:
        - Position:

        - The location of the object in absolute 3D coordinates.
        - Think meters away from the camera's calibrated startup location.
        - Centered around pivot point.
    - Rotation:
        - The angle of the object.
        - Think degrees (or radians) difference from model's orientation.
        - Rotated around pivot point.
    - Scale:
        - Relative to model size (1.0 = default = model's original size).
        - Scaled on pivot point.
- Designing in Layers:
    - Foreground: 3D models, spatial sound.
    - Midground: Terrain, Ambient music.
    - Background: Skybox, Background sound.
- Interaction Design of XR:
    - Implicit Interactions:
        - Camera-based: gaze, eye.
        - System-based: location, time, counter, event, physics.
    - Explicit Interactions:
        - User-based: controller, hand, finger pose/gesture, voice command.
        - Environment- based: maker, object.
    - Combination of the above:
        - System-based + user-based (mixed initiative): dwell on target.
        - User-based + user-based (multi-modal): gesture + speech command.

2. Menus and Navigation in VR:
    - Options of Menus:
        - Fixed: anchored in the world.
        - HUD: attached to the head (camera).
        - Controller: attached to the tracked controller,
        - Hand: attached to the tracked hand.
    - Ways to travel:
        - Menus
        - Teleportation:
            - Example code: https://codepen.io/mnebeling/pen/JxboWj
            - Camera Rig: moves when teleporting. (not just moving the camera itself while teleporting)



    - Walking (6 DOF)
    - Redirected Walking (6 DOF)

3. Object Selection and Manipulation:
    - Object Selection:
        - For far objects: use ray-casting and hit-testing for object selection.
        - For near objects: use collision detection for object selection.
    - Object Manipulation:
        - Hover: change object appearance.
        - Grab: change object position.
        - Rotate: Change object orientation.
        - Stretch: Change object scale.
4. VR with WebXR:
    - Example:
        - A-Frame Animation: https://codepen.io/mnebeling/pen/EGvxgp.
        - A-Frame Lights: https://codepen.io/mnebeling/pen/yZbwOd.
        - A-Frame Cardboard Template: https://codepen.io/mnebeling/pen/vvoOOj.
        - A-Frame Environment Component: https://codepen.io/mnbeling/pen/PXmNXZ.
        - A-Frame Sound: https://codepen.io/mnbeling/pen/dyMPLRN.
        - A-Frame Physics: https://codepen.io/mnbeling/pen/vvmEOe.
        - A-Frame Particle System: https://codepen.io/mnbeling/pen/RvVMwL.
        - A-Frame Teleport Controls: https://codepen.io/mnbeling/pen/poygYpb.
        - A-Frame Laser Controls: https://codepen.io/mnbeling/pen/LqpaRN.
        - A-Frame Hand Controls: https://codepen.io/mnbeling/pen/Yzqeyme.
        - A-Frame Super Hands: https://codepen.io/mnbeling/pen/qLQjLJ.
        - A-Frame HUD: https://codepen.io/mnbeling/pen/aPeqRV.
        - A-Frame Menu: https://codepen.io/mnbeling/pen/VgWvdy.
        - A-Frame Gaze-based Travel: https://codepen.io/mnbeling/pen/JxboWj.
    - Animations in A-Frame:

```
<a-entity id="gifaffe" position="0 0 -5" rotation="-90 -90 0"
scale="0.0025 0.0025 0.0025" gpoly="polyid: aTAr4AayOrB; APT_KEY:
AIzaSyDsovdrekNH2A-NiKfMtMWA8LXebZQ41nk" shadow animation="property:
rotation; to: 360 180 0; loop: true; dur: 2500; dir: alternate; easing:
linear" event-set__click="visible: false"></a-entity>
```

        - Defined in the `<a-scene>` block.
        - To use entities defined in A-Frame: can use `<a-plane>`, `<a-sky>` etc.
        - Poly id and APT_KEY is needed to use Google Poly.
        - In animation: different properties can be accessed and the list can be found on https://aframe.io/docs/1.1.0/components/animation.html#sidebar.
        - Easing defines the accelerations and speed throughout the cycle of the animation. Linear easing makes the movement smoother.
        - Shadow of entities can be added in `<a-entity>` with light casting shadow.
        - `event-set` can add the entity to listen to certain event and perform certain react.
    - Lights in A-Frame:

```
<a-light id="alight" type="ambient" color="#BBB" castShadow="true"></a-
light>

<a-light id="dlight" type="directional" color="#FFF" intensity="0.6"
castShadow="true" shadowCameraVisible="true" angle="20" target="giraffe"
position="-0.5 1 1"></a-light>
```

        - Defined in the `<a-scene>` block.
        - Ambient lights globally affect all entities in the scene. The `color` and `intensity` properties define ambient lights. Additionally, `position`, `rotation`, and `scale` have no effect on ambient lights.
    - Directional lights are like a light source that is infinitely far away, but shining from a specific direction. Thus, absolute position do not have an effect on the intensity of the light on an entity. We can specify the direction using the `position` component.

- Adding `shadowCameraVisible: true` creates a visual aid for debugging: objects outside the camera's view cannot cast or receive shadows.
- Cardboard Template (cursor) in A-Frame:

```
<a-cursor color="white" fuse="true" animation__click="property: scale;
startEvents: click; easing: easeInCubic; dur: 150; from: 0.1 0.1 0.1; to: 1
1 1" animation__fusing="property: scale; startEvents: fusing; easing:
easeInCubic; dur: 1500; from: 1 1 1; to: 0.1 0.1 0.1"
animation__mouseleave="property: scale; startEvents: mouseleave; easing:
easeInCubic; dur: 500; to: 1 1 1">
```

    - Defined in the `<a-camera>` block.
    - Different animations can be defined in `<a-cursor>`.
    - `fuse` defines whether cursor is fuse-based (gaze-based).
- Environment Component in A-Frame:

```
<script src="https://unpkg.com/aframe-environment-component/dist/aframe-environment-component.min.js"></script>
...
<a-entity environment="preset: forest; groundColor: #f0f0f0; groundTexture: none; horizonColor: silver; skyColor: skyblue; dressingColor: #fcfcfc; shadow: true; lighting: distant"></a-entity>
```

  - Environment can be set as an `<a-entity>` component.
  - Needs to install `"https://unpkg.com/aframe-environment-component/dist/aframe-environment-component.min.js"` before using environment.
  - `preset` are some preset environments in A-Frame with the default value `default`.
  - `lighting` contains `none`, `distant`, and `point` with the default value `distant`. The color and intensity of lighting are estimated automatically.
  - `skyType` contains `color`, `gradient` and `atmosphere` with the default value `atmosphere`.
  - `fog` can add fog to the environment and the value can be adjusted.

- Sound in A-Frame:

```
<audio id="rainforest"
src="//cdn.rawgit.com/michaelnebeling/src/master/rainforest_ambience-GlorySunz-1938133500.mp3" crossorigin="anonymous"></audio>
```

  - Defined in the `<a-asset>` block.

```
<a-scene cursor="rayOrigin: mouse" sound="src: #rainforest; on: click; loop: true; volume: 0.2; rolloffFactor: 0" environment="preset: forest; groundTexture: walkernoise; fog: 0.9">
```

  - Sound can be used in the cursor to make some sound effect when interacting.

- Physics in A-Frame:

```
<a-scene physics="debug: true">
<a-sphere dynamic-body position="0 1.25 -5" radius="1.25" color="#EF2D5E">
</a-sphere>
<a-plane static-body click-drag position="0 0 -4" rotation="-90 0 0"
width="4" height="4" color="#7BC8A4"></a-plane>
```

  - In `<a-scene>` physics needs to be enabled.
  - Objects can be set to `dynamic-body` or `static-body` to distinguish whether it can move with physics.

- Particle System in A-Frame:

```
<script src="https://unpkg.com/aframe-particle-system-component@1.0.x/dist/aframe-particle-system-component.min.js"></script>
...
<a-entity position="0 0 -25" particle-system="preset: stars; particleCount: 1000; opacity: 0.8; color: red"></a-entity>
```

  - Particle system is an `<a-entity>` in A-Frame and the value of `particle-system` can be set (similar to lights and environment).

    - Needs to install `"https://unpkg.com/aframe-particle-system-component@1.0.x/dist/aframe-particle-system-component.min.js"` before using the particle system.
- Teleport Controls in A-Frame:

```
<script src="https://unpkg.com/aframe-teleport-controls/dist/aframe-teleport-controls.min.js"></script>
...
<a-entity id="camera-rig">
<a-entity id="head" camera position="0 1.6 0" wasd-controls look-controls>
</a-entity>
<a-entity hand-controls="hand: left" teleport-controls="cameraRig: #camera-rig; teleportOrigin: #head; button: trigger"></a-entity>
<a-entity oculus-touch-controls="hand: right" teleport-controls="cameraRig: #camera-rig; teleportOrigin: #head; button: grip" raycaster="objects: #giraffe" line="color: red"></a-entity>
  </a-entity>
```

    - Camera rig, head and oculus touch controls an be set as `<a-entity>`.
    - Needs to install `"https://unpkg.com/aframe-teleport-controls/dist/aframe-teleport-controls.min.js"` before using teleport controls.
    - A ray-cast can be added to cast a ray that hits on certain object. If we want the ray to hit on everything that is interactable, we can use `.interactable` in the `objects` of the `raycaster` while defining the objects that we want to interact with to be : `class="interactable"`.
    - `hand-controls` shows a hand in the scene while `oculus-touch-controls` shows the oculus touch in the scene.
    - In the `button` section, `grip` enables the user to choose objects while `trigger` can move the user to the selected point.
    - For the head (camera), we can define the `look-controls` and `wasd-controls` to make it able to use on both VR headset and the desktop.
- Super Hands in A-Frame:

```
<script src="//cdn.rawgit.com/donmccurdy/aframe-physics-system/v3.1.2/dist/aframe-physics-system.min.js"></script>
<script src="https://unpkg.com/aframe-physics-extras/dist/aframe-physics-extras.min.js"></script>
<script src="https://unpkg.com/aframe-event-set-component@4.0.1/dist/aframe-event-set-component.min.js"></script>
<script src="https://unpkg.com/super-hands@3.0.0/dist/super-hands.min.js">
</script>
```

    - Needs to install the above libraries to use super hands.

```
<script>
  // turn controller's physics presence on only while button held down
  AFRAME.registerComponent("phase-shift", {
    init: function() {
      console.warn("installing phase shift");
      var el = this.el;
      el.addEventListener("gripdown", function() {
        el.setAttribute("collision-filter", {
          collisionForces: true
```

```
        });
      });
      el.addEventListener("gripup", function() {
        el.setAttribute("collision-filter", {
          collisionForces: false
        });
      });
    }
  });
</script>
```

- Needs to add the above code to turn controller's physics presence on only while button held down.

```
<a-assets>
       <a-mixin id="cube" geometry="primitive: box; width: 0.33; height:
0.33; depth: 0.33" hoverable grabbable stretchable draggable event-
set__hoveron="_event: grab-start; material.opacity: 0.7" event-
set__hoveroff="_event: drag-end; material.opacity: 1"
         dynamic-body shadow></a-mixin>

       <a-mixin id="hand" physics-collider phase-shift collision-
filter="collisionForces: false" static-body="shape: sphere; sphereRadius:
0.02" super-hands="colliderEvent: collisions;     colliderEventProperty:
els; colliderEndEvent: collisions; colliderEndEventProperty: clearedEls;">
       </a-mixin>
     </a-assets>
```

- Define the above two materials for objects and hands.

```
<!-- hand controls -->
    <a-entity id="left-hand" mixin="hand" hand-controls="left"></a-
entity>

    <a-entity id="right-hand" mixin="hand" hand-controls="right"></a-
entity>
  </a-entity>
```

- Use the above code to define the hand controls of the left and right hand entities.

- Menu in A-Frame:

```
<script src="https://rawgit.com/fernandojsg/aframe-teleport-
controls/master/dist/aframe-teleport-controls.min.js"></script>

<script src="https://unpkg.com/aframe-event-set-component/dist/aframe-event-
set-component.min.js"></script>

<script src="https://unpkg.com/aframe-layout-component/dist/aframe-layout-
component.min.js"></script>
```

- Needs to install the above libraries to use menu.

```
<a-mixin id="menu-item" text="align: center" geometry="primitive: plane"
material="opacity: 0.8; color: white" event-
set__mouseenter="material.opacity: 1.0" event-
set__mouseleave="material.opacity: 0.8" event-
set__mousedown="material.color: silver" event-
set__mouseup="material.color: white"></a-mixin>
```

- Define the above mixin as menu item.

```
   <!-- fixed menu -->
         <a-entity id="menu" position="0 0 -1" scale="0.25 0.25 0.25"
layout="type: box; plane: xy; columns: 3; margin: 1.1; align: center"
visible="false">

           <a-text mixin="menu-item" class="menu-item clickable" value="box"
color="#4CC3D9"></a-text>

           <a-text mixin="menu-item" class="menu-item clickable"
value="sphere" color="#EF2D5E"></a-text>

           <a-text mixin="menu-item" class="menu-item clickable"
value="cylinder" color="#FFC65D"></a-text>

         </a-entity>
```

- Define the entity of menu with `<a-text>` to display the menu.
- `visible="false"` means the menu is not visible at the beginning and can be brought out.
- In the JavaScript script below: make the objects hidden/shown when clicking the button.

```
function(e) {
  if (!menu.object3D.visible) return; // if menu not visible, shouldn't
do anything (click events would still be triggered without this check)

  let el = document.getElementById(this.getAttribute("value"));
  toggle(el);

  e.preventDefault();
  return false;
});

function toggle(el) {
  el.setAttribute("visible", !el.object3D.visible);
}
```

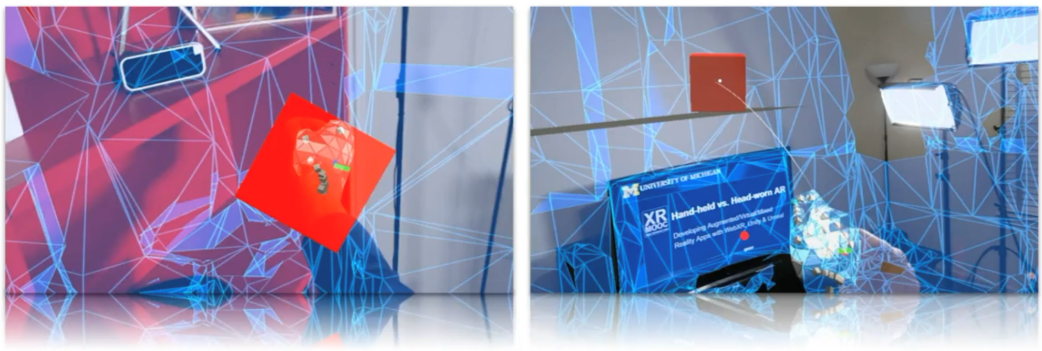
Week 3: Developing AR Applications

1. Designing an Augmented Reality:
   - Marker-based AR:
     - Marker Tracking: Default fiducials often work best. But can customize/randomize. Trade-off between blending with the environment and tracking.

        - Marker-based Interactions: To react to detected marker.
            - Found: show marker content.
            - Lost: hide when not tracked.
            - Extended: switch to marker-less when marker lost.
    - Marker-less AR:
        - Environmental Understanding: To detect physical objects & place virtual objects in environment.
            - Plane detection: gives surface.
            - Spatial mapping: depth sensing for occlusion rendering.
        - Light Estimation: To match lighting on virtual object.
            - Ambient intensity: apply average lighting to object.
            - Color temperature: use white balance to color correct object.
    - Marker-based vs. Marker-less AR:
        - Environment: Consider the scale of the experience.
        - Content: Consider the design of the augmentations.
        - Interactions: Consider explicit and implicit interactions.
2. Marker-based AR:
    - Marker-based Tracking:
        - Original image: Extract frame by frame.
        - Threshold: Segment out the marker.
        - Connected components: Identify marker components.
        - Contours: Get rough shape.
        - Marker edges & corners: Extract marker.
        - Fitted square: Estimate marker pose.
    - Marker Design: (default: Hiro marker)
        - Continuous black borders around marker are key.
        - Don't cut/occlude them or the marker won't be detected.
        - Smaller markers are harder to detect.
        - Size of marker determines object scale.
        - Content inside marker can be customized (with constraints).
    - Custom Markers (Vuforia): Basically any object can be a marker.
        - Codes: QR code or barcode scanner.
        - Faces, images and photos: Face and image recognition.
        - 3D models: Object recognition of intricate 3D objects.
        - Indoor locations: Environment recognition of 3D scanned physical areas and spaces.
        - Buildings: Combination of above to recognize facade, especially principle front.
    - Marker-based Interactions: Place, Point, Tap, Drag.
    - Marker-based AR Applications: Low-end Devices, Rapid Prototyping, Collaborative Experiences.
3. Marker-less AR:
    - Marker-less Tracking:
        - Plane detection: Horizontal and vertical planes.
        - Spatial mapping: 3D mesh reconstruction and texture mapping.
        - Scene Understanding:
            - Classification: face, walls, floors, tables, seats, windows, ceilings, ...
            - Segmentation: foreground/background, sky, people, ...

- Marker-less Interactions: Place, Point, Tap, Drag.
- Marker-less AR Applications: High-end Devices, Scene Understanding, Believable Experiences.

4. Hand-held vs. Head-worn AR:
    - Hand-held:
        - Varying display sizes.
        - Mix of 2D / 3D content + touch:
            - 2D screen + touch
            - 3D world + touch
            - Device motion gestures
            - Voice commands
        - Marker-based or marker-less:
            - Marker-based tracking
            - Marker-less tracking
            - Extended tracking
    - Head-worn:
        - Usually small display (FOV)
        - HUD + 3D content + gesture:
            - HUD + hand/voice
            - 3D world + hand gestures
            - Head / eye gaze
            - Voice commands
        - Advanced tracking with:
            - Inside-out 6 DOF tracking
            - Spatial mapping
            - Scene understanding
    - Near and Far Manipulation:



    - Head-worn AR Applications: Very High-end Devices, Direct Manipulation, Multimodal Interaction.

5. Marker-based AR with WebXR:
    - Examples:
        - A-Frame AR.js Template: https://codepen.io/mnebeling/pen/Jxvzyy.
        - Marker-based AR Interactions: https://xrmooc.glitch.me/arjs/interactions/.
    - AR.js:
        - AR.js is a web-based marker-based AR toolkit based on ARToolKit.
        - There are versions based on THREE.js and A-Frame.
        - It is relatively popular for its ease of use and portability.

- A-Frame AR.js Template:

```
<a-scene embedded arjs="sourceType: webcam" ar-mode-ui="enabled: false"
vr-mode-ui="enabled: false">

   <a-marker preset="hiro">
      <a-box id="box" position="0 0.5 0" rotation="90 0 0" scale="1 0.25
1" color="blue" opacity="0.8"></a-box>
   </a-marker>

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

  - First enable AR.js in the `a-scene` component and set the `sourceType` to `webcam`.
  - Define a marker in the `a-marker` component.
  - Place an `a-box` on the top of the marker as a child of `a-marker`.
- Interaction in marker-based AR:

```
...
   <a-entity camera>
       <a-cursor></a-cursor>
   </a-entity>
</a-scene>
<script>
    funciton changeColor(el, color) {
        el.setAttribute("color", color);
    }
    const boxEl = document.getElementById("box");
    boxEl.addEventListener("mouseenter", function() {
        changeColor(this, "blue");
    });
    boxEl.addEventListener("mouseleave", function() {
        changeColor(this, "red");
    });
</script>
...
```

  - The above code makes the box to change color when it is targeted by the cursor and the color changes back to red when the cursor leaves.
- Multi-marker in Marker-based AR:
  - Define Multi markers as `<a-marker>` components and have different components of the markers.
- AR Cube:
  - Has 6 sizes of different markers.
  - Ensure that at least one marker will not be occluded when the user is holding the cube.
- Tips and Tricks in marker-based AR:
  - Don't occlude the marker or the marker borders.
  - Smaller markers are harder to detect.
  - Size of marker determines object scale.
  - If you are using CodePen/Glitch, disable Auto-Updating Preview.
  - Use opacity: 0.5 for debugging / 1.0 to "hide" marker.
  - Use laptop webcam to develop AR.js interface.

        - You could use your phone to show the marker.
        - Use browser options to change the webcam that is being used.

6. Marker-less AR with WebXR:

    - Examples:
        - A-Frame Template: https://codepen.io/mnebeling/pen/wRJxNx.
        - Single object: https://xrmooc.glitch.me/less/single.
        - Lights on the object: https://xrmooc.glitch.me/less/lights/.
        - Hit Testing: https://xrmooc.glitch.me/less/hit-test/.
    - A-Frame Template:
        - Run the template on the phone, enter the debug mode and select AR mode, we can run the program in AR scene.
    - Hit Testing:

```
<a-scene background="color: black" webxr="optionalFeatures: hit-test,
local-floor, viewer; referenceSpaceType: local">
    ...
    <a-entity ... hide-in-ar-mode ..>
        ...
    </a-entity>
    ...
</a-scene>
```

        - Needs to use `webxr` specification in the `a-scene` component.
        - We can make certain entities hide in AR mode by `hide-in-ar-mode` property in `a-entity` component.
        - JavaScript codes to perform hit testing:

```
<script>
<!-- When the user does a select action, such as tapping on the
screen in handheld AR it will update the position -->
  const reticle = document.getElementById('reticle');
  reticle.addEventListener('select', function (event) {

    if (
      event.detail.inputSource.profiles[0] === "generic-touchscreen"
&&
      event.detail.inputSource.gamepad
    ) {
      // Handle input if you want via the axes -1, -1 is top left,
1, 1 is bottom right
      const x = event.detail.inputSource.gamepad.axes[0];
      const y = event.detail.inputSource.gamepad.axes[1];
      console.log(x,y);
    }

    // Get the next hit event to position the element
    reticle.addEventListener('hit', function (event) {
      if (document.querySelector('a-scene').is("ar-mode")) {
        const position = this.getAttribute('position');
        document.getElementById('world').setAttribute('position',
position);
        document.getElementById('world').setAttribute('visible',
true);
```

```
      }
    }, {
      once: true
    });
  });
</script>
```

- Tips and Tricks:
  - Provide a preview of the content during placement.
  - Incorporate light and shadow to blend with the environment.
  - Plan for the impact of occlusion rendering and light estimation.
  - Prototype the experience using 360, VR or marker-based AR.
  - Consider allowing users to adapt the scale of the experience.
  - Consider adapting the layout to fit the detected plane.

Week 4: Special Topics in XR

1. Advanced Techniques:
   - Advanced VR:
     - Procedural generation: Gradually load more and more VR content.
     - Redirected walking: Let the users feel that they are walking straight while they are actually walking in a circle.
     - Custom controllers:
       - Haptic feedback to make users feel like they are using real object
       - Physical affordances to make controller look and feel like real object.
   - Advanced AR:
     - 3D reconstruction:
       - Spatial mesh to scan the physical world as dense 3D point cloud.
       - Object recognition to identify objects in mesh and assign classification label.
     - Object tracking:
       - Object recognition to detect physical object in real world.
       - Registration and tracking: register object in 3D and track it in scene.
     - Custom displays:
       - Spatial / projective: Lightform, Looking Glass.
       - Room-sized: AR mirrors.
   - Accessibility: Scene Understanding.
   - Text input:
     - Virtual Keyboard: Usually not very efficient in neither VR not AR.
     - Physical Keyboard: Usually not part of the AR/VR setup unless in developer mode.
     - Speech Commands: Usually not clear what commands are available.
     - Multimodal Input: Usually the safest but hardest to execute text input method.
   - Collaboration: Co-located, Remote.
   - Adaptive layout / Customization: Move between tabletop, room and world scale.
   - Progressive XR.
   - Mixed reality capture / virtual production.
2. XR Research:

- Topics: Applications, Sensing and Input, Human Factors, Design and Accessibility, Prototyping, Collaboration.
- Skills: UX/UI Skills, Programming Skills, Research Skills.
- Questions:
    - Good Questions:
        - Has this been done?
        - What will this enable in the future?
        - What are the open issues?
        - What if we had this?
        - What does it enable?
        - How can we design with them?
    - Bad Questions:
        - What has been done?
        - What is currently possible?
        - What are the low hanging fruits?
        - What does it take to have this?
        - What is cool about it?
        - Who are we designing for?