Mobile HCI: Introduction to XR Development

Euan Freeman (euan.freeman@glasgow.ac.uk)

1. Introduction

During this lab exercise, you will begin implementing a **mobile mixed reality experience**, using the **A-Frame** web framework. This exercise only requires a basic understanding of web development technologies: your previous experience with Java development should be enough for you to figure out how to use Javascript and A-Frame.

Mixed reality is a term used to describe interactive experiences where virtual content is added to the real world (augmented reality) or used to create a wholly virtual world (virtual reality). Mobile mixed reality experiences can be created for smartphones, using their integrated motion sensors and cameras to capture the real world, render virtual content on top of it, and ensure virtual content is placed in an appropriate 3D position whilst the user moves their device.

To simplify the process of developing and deploying web applications, we will be using **Glitch** for this lab exercise (and again in later weeks). Glitch is a browser-based development environment that enables you to develop web apps on your desktop browser, instantly deployed so that you (or anyone else) can test it using another device (i.e., your smartphone).

Most modern smartphone web browsers have some support for web-based mixed reality platforms (e.g., Safari and Chrome for iOS, Chrome and Firefox for Android). You can check your own device compatibility by going to https://immersive-web.github.io/webxr-samples/ on your mobile device. If you don't have a compatible device, you can still test your prototypes using your desktop web browser – instead of moving a smartphone to 'look around' and view 3D content, you can look using the mouse pointer instead.

1.1. Helpful resources

- https://aframe.io/docs/
- https://aframe.io/docs/1.5.0/introduction/best-practices.html
- https://glitch.com/

1.2. Intended Learning Outcomes

- **ILO4:** develop and evaluate prototypes of mobile interactive systems using a variety of prototyping methods and evaluation techniques.
- **ILO5:** discuss cutting edge developments in mobile human-computer interaction, such as context-aware systems, sensor-based interaction, location-based interaction, and mixed reality.

2. Lab Exercise

2.1. Introduction

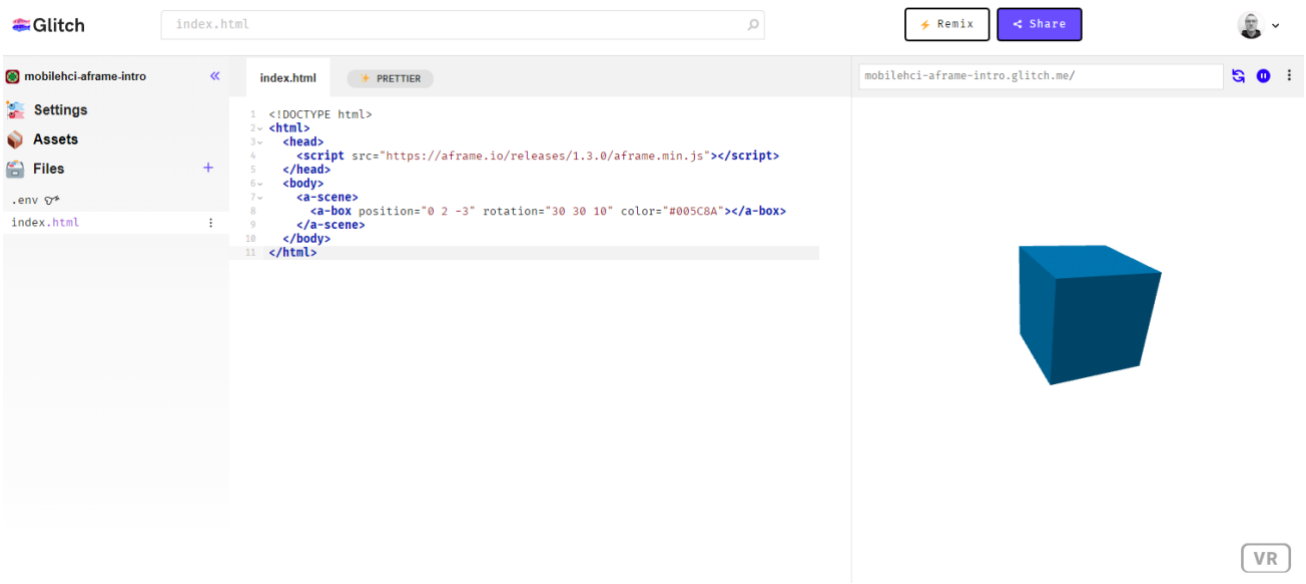
We are going to be using the **Glitch** development environment (https://glitch.com/) for this lab exercise and for other labs later in the semester. You may also prefer to develop your coursework projects using Glitch.

Glitch is a browser-based web development environment that allows you to edit code on your computer then run it on a different device by visiting a unique URL. This simplifies deployment and makes it easy to share prototypes with others, e.g., for evaluating your Mobile HCI coursework projects with other people.

You do not require an account to use Glitch, but we recommend you create one so you can keep working on your projects at a later date, since temporary projects are deleted after five days.

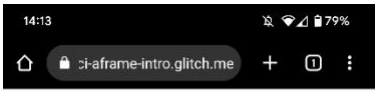
2.2. Access the Starter Project

Go to https://glitch.com/edit/#!/mobilehci-aframe-intro to access the starter project for this lab exercise. Press the Preview button in the bottom toolbar and open the preview pane. This page should now look something like the following screenshot. There are three main panels in this view: a list of project files on the left, a file editor in the middle, and a live preview of the project on the right. In the preview panel, you should see a blue cube in the centre.



2.3. Remix the Starter Project

Press the Remix button in the top toolbar to create an editable copy of the starter project. This clones the project and creates a unique instance for you to edit. Open the Preview pane again and note the unique URL shown at the top. Using your mobile device, open a web browser and navigate to that unique URL – you need to include **https://** at the beginning of the URL because most browsers only allow access to XR features via a secure https (not http) connection.



When viewing your project via a mobile device, the content will be anchored in a real-world position relative to your device – the start position is typically based on the orientation of your device at the time the website loads. When you move your device, notice the virtual content also moves so that it stays in its original position and orientation in the real world.

If you are using your desktop to preview your prototype, try clicking and dragging in the preview pane with the mouse pointer. This has a similar effect to 'looking around' with a mobile device.

Depending on browser compatibility, you may see buttons in your mobile web browser labelled AR and VR (like in the screenshot on the left). If these buttons are visible, try tapping the AR button. This should enable a pass-through feed from the camera, so that virtual content appears to be rendered over the real world. You may need to give the website permission to access the camera.

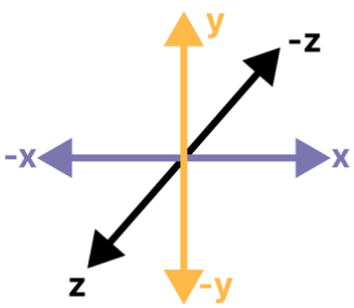
Congratulations – you've just deployed your very first mixed reality prototype!

2.4. Add Virtual Objects

If not already visible, open index.html in the file editor by selecting it from the project files panel. Note the **<script>** element on Line 4 – this effectively adds the A-Frame framework to your web app and will be necessary if you are creating other web apps using A-Frame.

When using A-Frame, mixed reality content is added to a **scene**, which you create in a web page using an **<a-scene>** element (see Line 7). A scene is a container for other A-Frame entities, so everything in your web app will be defined inside this container.

In the starter project, there is already one virtual object in the scene: an **<a-box>** entity, which in this case creates a blue cube (as pictured above). Note the syntax used to specify the virtual cube and its properties. Three properties are specified (position, rotation, color), with their values given as strings. For a complete list of properties for the **<a-box>** entity, refer to the documentation (https://aframe.io/docs/1.5.0/primitives/a-box.html).



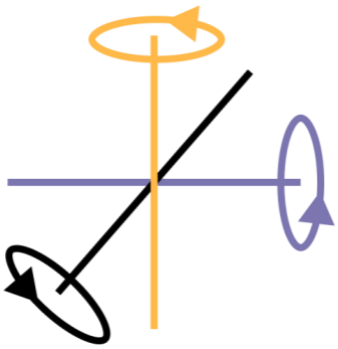
Since we are creating 3D content, object positions are specified using a three-dimensional coordinate system: **position="x y z"**. All positions are defined relative to the origin (0, 0, 0). Note that all units are specified in metres, e.g., a y coordinate of 2 would be 2m above the origin. As shown in the image on the left, the x and y axes can be thought of as horizontal and vertical respectively, whereas the z-axis controls the *depth* of the content, relative to the camera.

In the starter project, the blue cube is positioned at (0, 2, -3) which means 2m higher than the origin (y-axis) and 3m forward from the origin (z-axis). Note that the positive direction of the z-axis would be *behind* the initial camera position.

Task 1 Copy and paste the line of code that creates the cube (i.e., the **<a-box>** element) to create a second cube. Change it to a different colour by modifying the **color** property (e.g., #7D2239), and change the **position** property of the new cube. Try changing one axis at a time, to help familiarise yourself with the 3D coordinate system. Explore both positive and negative values in each axis, to help orient yourself.

If you are using a mobile device to preview your web app, try position the cube *behind* you, *above* you, *under* you, etc. You can use the blue cube as a reference to help orient yourself. Play around with the coordinate system to help familiarise yourself with the 3D content space and to help understand how position affects the perspective of different objects.

Note that colours can be specified using named colours (e.g., black, red) or using hex codes (e.g., see https://www.gla.ac.uk/myglasgow/staff/brandtoolkit/brandelements/colours/ for examples from the University of Glasgow colour palette). Also note that A-Frame (like HTML and CSS more generally) uses the US-English spelling of color.



Task 2 The **rotation** property also uses a three-dimensional coordinate system, illustrated by the image on the left. Instead of specifying translation along each axis (like position), the rotation property specifies the amount of angular rotation around each axis.

For this task, change the rotation properties of each cube to further familiarise yourself with the coordinate system. It is important that you become familiar with the position and orientation of 3D content, before moving on to more complex projects.

Task 3 By default, **<a-box>** entities have a width, height, and depth of 1 unit (i.e., 1 metre). You can change the dimensions of your cube using the **width**, **height**, and **depth** properties. Add these properties then change their values to create different sized boxes.

Note that you could specify a depth of 0 to create a 'flat' box (i.e., a 2D rectangle) instead of a 3D one, although the **<a-plane>** entity is better suited to creating flat objects.

Task 4 Boxes are cool, but you can also create other 2D and 3D objects. Now add a few more virtual objects to your scene using different shapes. If you are using a mobile device that supports the AR view using your mobile device camera, perhaps you could try position these objects on top of surfaces in your surroundings: e.g., a box positioned on top of your monitor, a sphere positioned beside the keyboard, etc.

Refer to the docs (https://aframe.io/docs/master/introduction/) for this task, e.g., by scrolling down to the list of primitives at the bottom of the menu or using the following links:

- https://aframe.io/docs/master/primitives/a-box.html
- https://aframe.io/docs/master/primitives/a-circle.html
- https://aframe.io/docs/master/primitives/a-cone.html
- https://aframe.io/docs/master/primitives/a-cylinder.html
- https://aframe.io/docs/master/primitives/a-plane.html
- https://aframe.io/docs/master/primitives/a-ring.html
- https://aframe.io/docs/master/primitives/a-sphere.html
- https://aframe.io/docs/master/primitives/a-triangle.html

The documentation for each primitive will identify the properties you may need to specify to control the appearance of your virtual objects. For example, an **<a-sphere>** entity uses the **radius** property to determine sphere size and an **<a-plane>** entity (i.e., a rectangle) uses the **height** and **width** properties to determine rectangle size.

2.5. Summary

You've now completed the main tasks for this lab exercise, which was an introduction to mixed reality prototyping using the Glitch development environment and the A-Frame mixed reality web framework.

You should hopefully now feel comfortable using Glitch for implementation and testing on a mobile device (if available). You should also hopefully now feel comfortable with the basic concepts of adding virtual objects to a three-dimensional scene and controlling their appearance via their properties. Later in the course, we will be using A-Frame to create interactive mixed reality prototypes, so it is important that you pick up these basic scene-creation skills now.

2.6. Extra Tasks

In our starter project, we did not explicitly specify any camera properties within our A-Frame scene. In this case, the A-Frame framework uses the default camera settings to determine the relationship between device view and virtual content. However, you can explicitly specify camera properties and in doing so, change the relationship between user and content. You can read more about the camera at the following page: https://aframe.io/docs/1.5.0/components/camera.html

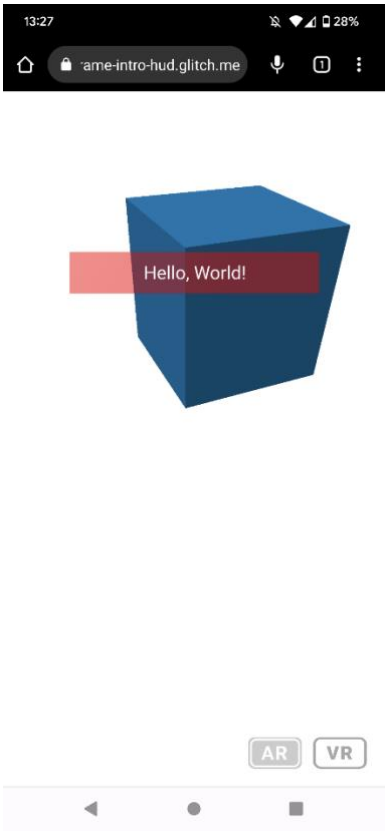
Task 5 In the previous tasks, all virtual objects were defined relative to the camera position in 3D space. This means that when the camera moves (e.g., by looking towards the left), the virtual content will shift on screen so that it appears to remain in its original position.

When prototyping augmented reality content, you may prefer virtual content to always be visible in the same region of the screen, e.g., information in a heads-up display (HUD). You can create such always-visible content in an A-Frame scene by explicitly adding a camera component and then adding elements as a child element of the camera:

```
<a-scene>
  <!-- Dynamic position on screen (normal) -->
  <a-box position="0 2 -3" rotation="30 30 10" color="#005C8A"></a-box>

  <!-- Define camera at head height (1.6m) -->
  <a-entity camera look-controls wasd-controls position="0 1.6 0">
    <!-- Fixed position on-screen (HUD) -->
    <a-entity id="hud" position="0 0.2 -0.5" height="0.2" width="0.2">
      <a-plane scale="0.3 0.05 0.1" color="red" opacity="0.5"></a-plane>
      <a-text value="Hello, World!" scale="0.1 0.1" align="center"></a-text>
    </a-entity>
  </a-entity>
</a-scene>
```

Integrate the above code with your project so that you have explicitly specified a camera object, with a virtual object included as a child element. Note that we have added only one child element, which we have given the “hud” identifier. By adding virtual objects as children of this container element, it will be easier to later reposition and resize the HUD, without needing to alter the position and scale of each individual element.



Look around the virtual scene on your mobile device (or in the preview pane) and note how the HUD contain remains in position, versus the other objects in the scene (e.g., as shown in the screenshot on the left).

This is not a perfect solution because it is still possible for virtual objects to end up in the space between the camera and the HUD objects, but this is unlikely - as any virtual content would not be fully visible from such a short distance away. Also note that since we are placing the HUD content so close to the camera, it is necessary to reduce the size, to avoid the HUD content filling the user’s field of view and occluding the rest of the virtual scene.

Task 6 Create a HUD that includes various text widgets in different regions of the display, e.g., time in the top left, battery level in the top right. Play around with the position and scale properties, both for the HUD object and its children objects - try to figure out the effects these have on each other and on the appearance of the content.

Task 7 Modify your HUD prototype to create a mock augmented reality glasses display – e.g., use several <a-plane> entities to create the outline of the glasses frame and position the HUD widgets within those boundaries.