Tutorial and Laboratories

11464 – 11524PG AR/VR for Data Analysis and Communication

Week 4

Introduction

In this computer lab we will continue practice data visualisation, but in this case, we will learn about the use of **Augmented Reality (AR)**, and how we can use R to create AR content. These technologies can be a great tool for live presentations and very relevant to the communication of results.

It is assumed that you are now familiar with base R to create plots, how to subset data from data frames, how to enter data or read data files, how to plot data; topics that were covered in our previous tutorials. It is recommended to complete the tutorials in week 1,2, and 3 before attempting this tutorial.

Skills Covered in this tutorial include:

- Basics of HTML
- A-frame.
- AR for data science.
- Generating AR content from R.
- Using markers to visualise AR content

Before starting with the tutorial, you need to print out the markers (preferred option) or use your cell phone to visualise the AR content. The markers can be obtained in Canvas in Week 4 page. The two markers (Kanji and Hiro) are presented below:

1. Basics of HTML

Hypertext Markup Language (HTML) is a computer language that displays human-readable information and uses tags (e.g., <table>, <h1>,) to define elements within a document. Its structure is quite simple and understanding its basics is fundamental for this tutorial. If you do not have much experience with HTML, no problem! It is fairly easy to pick up. The html instructions of a simple website look like the following example:

```
<!DOCTYPE html>
<html>
    <head>      <!-- Head -->
        <title>Page Title</title>
    </head>
    <body>      <!-- Body -->
      <h1>Augmented and Virtual Reality</h1>
        <p>This is a paragraph.</p>
      <h3>For Data Analysis and Communication</h3>
        <p>This is another paragraph.</p>
    </body>
</html>
```

Augmented and Virtual Reality

This is a paragraph.

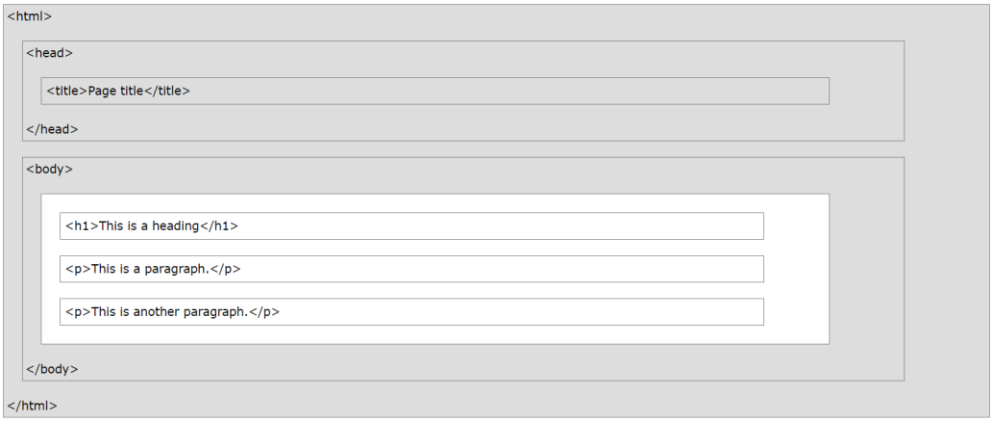
For Data Analysis and Communication

This is another paragrpah.

As you can see in the example above, we define HTML elements using a start tag (e.g.,), the content (e.g., “This is a paragraph”), and an end tag (e.g.,). Thus, an HTML element is everything from the start tag to the end tag. For a complete list of HTML tags and their description, please visit this link: https://www.w3schools.com/TAGS/default.ASP. The elements used in the example above are:

- <!DOCTYPE html> – this declaration defines that this document is an HTML document.
- <html> – this element is the root element of an HTML page
- <head> – it contains meta information about the HTML page
- <title> – it specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- <body> – this element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- <h1> – it defines a large heading, you can also try with h2, h3, h4, h5
- – this element defines a paragraph

We can visualise the content of an HTML file using a web browser (e.g., Chrome, Edge, Safari, etc.). The web browser reads HTML documents and display them correctly. Web browsers do not display the HTML tags, instead the HTML tags are used to determine how to display the content accurately as defined by the designer/programmer. The figure below shows an example of an HTML page structure, note that only the content inside the <body> section (in white) is displayed in the browser.



In order to start writing HTML code, you can use a text editor or source code editor to create and/or edit your html files. You can start with **Notepad** if you are using Windows, or **TextEdit** if you are using Mac. There are other open-source options that come with extra functionalities which can help you with the visualisation of your code. Some great options are:

- Notepad++ - https://notepad-plus-plus.org/downloads/
- Sublime - https://www.sublimetext.com/
- Brackets - http://brackets.io/

Exercise 1. Practise HTML. In this exercise we will learn and practice the basic structure to create a simple HTML file. Do the following:

1. Visit the following website (https://www.w3schools.com/html/) and get familiar with the content.
2. Then, let's create a simple html document that contains heading (<h1>) and paragraphs ( ), follow this link: https://www.w3schools.com/html/tryit.asp?filename=tryhtml_default.
3. Then, let's add more paragraphs ( ) to the example. Please add the following, click on the green button that says "Run" and observe the result:
   <h3>AR/VR for Data Analysis and Communication</h3>
   This is another paragraph.
4. Try with different heading sizes (e.g., h2, h4, h5) and observe the result.
5. Now, let's add an image to your page. HTML images are defined with the <img> tag. The source and name of file (src), width and height are attributes to change the size of the displayed image. Try the following and observe the result:
   <img src="pic_trulli.jpg" width="550" height="400">

Exercise 2. Create a HTML file. Now, let's create our own html document using some of the structures that we just used in the previous exercise, do the following:

1. Open an empty document in your preferred text/code editor.
2. Find an image from the internet that you would like to use for your web page, save it in your working folder.
3. Select and copy the structure that you created in the previous exercise and paste it into you empty new document. Make sure to change the name of your image and the extension (e.g., jpeg, png, gif, etc.)
4. Save the file as "myfile.html", in Notepad/Notepad++ you should make sure to select "save as type: All files", in TextEdit you should change it to plain text (Format > Make Plain Text). Once your file is saved, close the file, and open the folder where you saved it. Then, double click on your html file to see the result.
5. How would you include a paragraph that says: Hello world!

Once you pick up the general structure or syntax of HTML (opening tags, attributes, closing tags), then you are good to go.

2. A-Frame

A-Frame is a web framework that helps us build virtual reality and augmented reality experiences. A-frame can be developed from a plain HMTL file without having to install anything. A-frame was developed to be easy to use and powerful enough to develop AR/VR content. A-frame supports most VR headsets such as HTC Vive, Oculus Rift, Windows Mixed Reality, Daydream, Samsung GearVR, Google Cardboard, Oculus Go. You can learn more about A-frame using this link: https://aframe.io/.

A-Frame is based on top of HTML, making it simple to get started. The easiest way to start with A-frame is by creating an .html file, including A-Frame in the <head> section of your html file, and by adding a scene <a-scene> in the <body> section, as the following template:

```
<html>
  <head>
     <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  </head>
  <body>
     <a-scene>
          <!--Actual scene elements here -->
     </a-scene>
   </body>
</html>
```

<a-scene> is a primitive tag from A-Frame, and all A-Frame elements must be placed inside these tags (<a-scene> </a-scene>). It is important to remember closing all tags (e.g., </p>) otherwise your VR/AR application will not work as expected.

2.1. A-Frame for VR

Exercise 3. Create HTML file to run A-Frame. Let's create our Hello World!! VR example by adding a <a-text> tag in our html document. Open a new document in your preferred text/code editor and do the following:

```
<html>
 <head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
 </head>
 <body>
  <a-scene>
   <a-text value="Hello, World!" color="#AA145D" position="0 2 -3"></a-text>
  </a-scene>
 </body>
</html>
```

After loading the browser, you will see a blank page with the "Hello World!" on the centre of the screen. In addition, you will see a VR icon on the bottom right (two examples below). If you click on the icon, you will get an immerse VR experience that can be used in conjunction with a VR headset. Obviously, if you watch this on your computer's screen won't have an immersive experience.

Try to change the configuration of your scene and see the result, for instance:

- Add a background colour: <a-scene background="color: #000066"> You can find a HTML color picker in this link: https://www.w3schools.com/colors/colors_picker.asp
- Change the size of your text: width="26"
- Change the position of your text (x,y,z): position="0 2 -3"

Exercise 4. A-Frame custom elements. A-Frame has a number of custom elements (also called primitives) that wrap the entity-component pattern to make it easy to understand for beginners. It is possible to create your own primitives, however that is beyond this unit, for now let's use some of the basic primitives. Please do the following:

```
<html>
 <head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
 </head>
 <body>
  <a-scene>
   <a-box height="1" width="2" depth="3" position="0 0 -4" rotation="0 45 0" color="blue"></a-box>
  </a-scene>
 </body>
</html>
```

Congratulations, you have build your first VR environment. You can use the keyboard (arrow keys) and the mouse (click and drag) to interact with your VR environment.

In the examples above, there are primitives to create different components in your <a-scene>. These primitives are shorthand versions for complex but common types of entities (e.g., <a-box>), so they provide a familiar interface for beginners. Let's understand how these work, for instance the <a-box> primitive is composed of.

- <a-box> – will draw a box in our HMTL page
- height – refers to how tall (in meters) the box will be (x axis)
- width – how wide the box will be (y axis)
- depth – it gives a 3D perspective (z axis)
- position – to set where to place the element using three parametes (x,y,z)
- rotation – to set rotation of the elment relative to the viewer using three parameters (x,y,z)
- color – it specifies the color of the box, you can use hexadecimal (e.g., #4CC3D9) or its name (e.g., "blue").

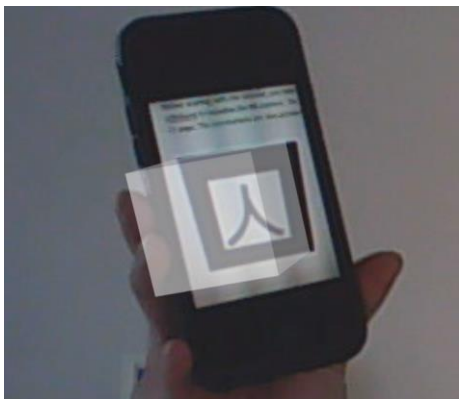
2.2. A-Frame for AR

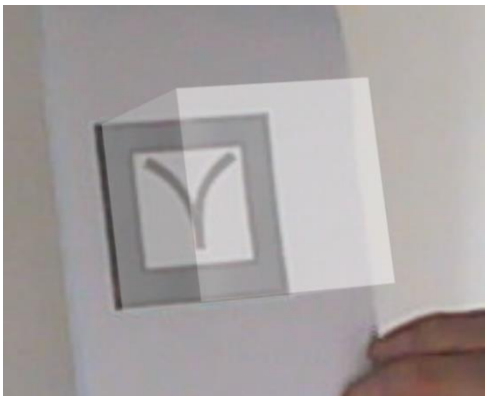
Now that we know how A-Frame works, we can learn how to use A-Frame for Augmented Reality (AR). In this case, we need to add the AR.js library in the <body> section of our html file. Then, we need to initialise ar.js in our scene (e.g., <a-scene embedded arjs>). Finally, we need to tell A-Frame that we want *arjs* to control the camera, so we need to add the <a-marker-camera preset='kanji'> primitive to identify your marker **using your webcam**.

Exercise 5. Create AR content. Now let's create our first example using the kanji maker (kanji_marker.png) to control the interaction with our AR content. You will use a transparent <a-box> entity for our environment. Please try the following within your HTML tags <html> </html>:

```
<head>
        <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
        <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script></head>
<body>
  <a-scene embedded arjs='sourceType: webcam;'>
        <a-marker preset='kanji'>
              <a-box position='0 0.5 0' material='opacity: 0.5;'></a-box>
        </a-marker>
        <a-entity camera></a-entity>
  </a-scene>
</body>
```

The expected output is presented bellow. The image on the left panel was obtained by using the Kanji marker on the screen of my cell phone, while the image on the right panel was obtained using the marker printed on paper.

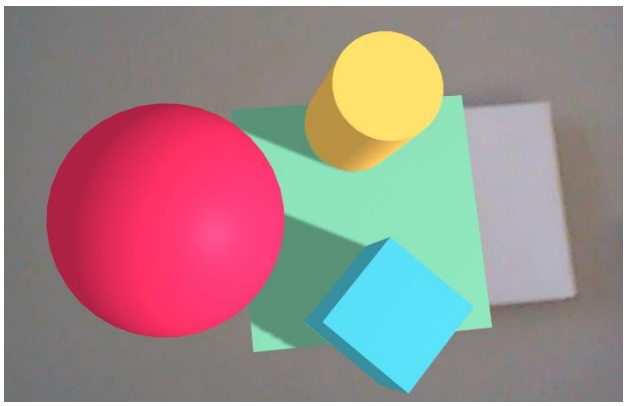


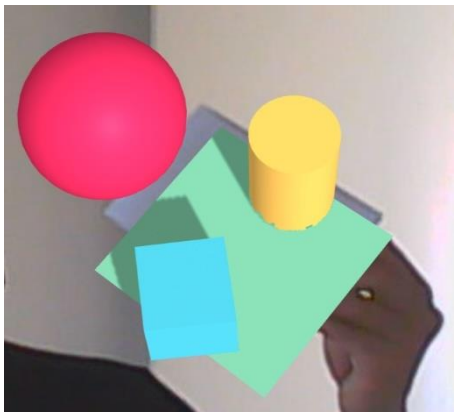


Exercise 6. Adding more entities. Now that we implemented our first AR experience, lets add more entities to our environment. For this example, we will use one of our previous examples with some little changes for better visualisation. Please do the following:

```
<head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script>
</head>

<body>
  <a-scene embedded arjs='sourceType: webcam;'>
    <a-marker preset='kanji'>
      <a-box position="-1 0.5 0" rotation="0 45 0" color="#4CC3D9" shadow></a-box>
      <a-sphere position="0 1.25 -1" radius="0.5" color="#EF2D5E" shadow></a-sphere>
      <a-cylinder position="1 0.75 0" radius="0.5" height="0.5" color="#FFC65D" shadow></a-cylinder>
      <a-plane position="0 0 0" rotation="-90 0 0" width="2.5" height="2.5" color="#7BC8A4" shadow></a-plane>
    </a-marker>
    <a-entity camera></a-entity>
  </a-scene>
</body>
```

Expected output below.

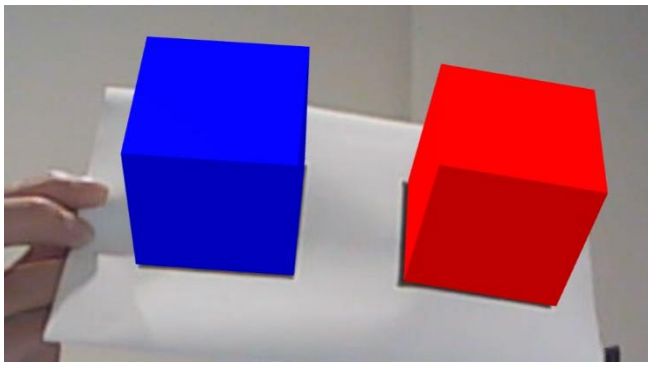




Exercise 7. Using different markers simultaneously. We can use multiple markers (hiro and kanji) in the same scene, so we can interact with different elements/primitives simultaneously in our scene. Please try the following:

```
<head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script>
</head>

<body>
  <a-scene embedded arjs='sourceType: webcam;'>
      <!-- define your first marker -->
      <a-marker preset="hiro">
          <!-- here define the content to display on top of the marker -->
          <a-box position="0 0.5 0" material="color:red;"></a-box>
      </a-marker>
      <!-- define your second marker -->
      <a-marker preset="kanji">
          <!-- here define the content to display on top of the marker -->
          <a-box position="0 0.5 0" material="color:blue;"></a-box>
      </a-marker>
      <!-- define the camera -->
      <a-entity camera></a-entity>
  </a-scene>
</body>
```

Expected output below.

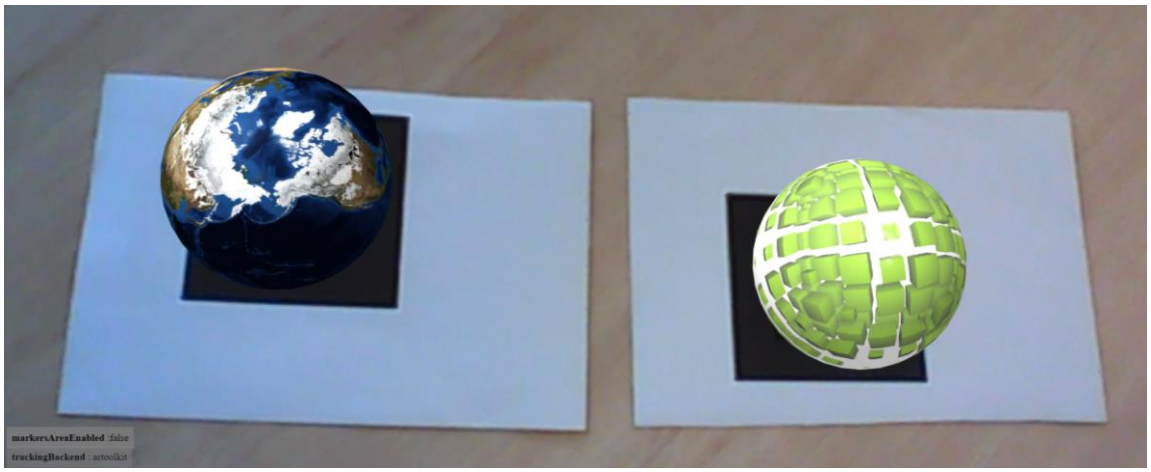


Exercise 8. Now, let's try to create two <a-sphere> entities with our own texture. Using the previous example, remove both <a-box> entities, and add one of following <a-sphere> entities inside one of the <a-marker> entities and add the other one inside the other <a-marker>. Observe the results.

```
<head>
  <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
  <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script>
</head>

<body>
  <a-scene embedded arjs='sourceType: webcam;'>
    <!-- define your first marker -->
    <a-marker preset="hiro">
      <!-- here define the content to display on top of the marker -->
```

```
        <a-sphere src="https://raw.githubusercontent.com/aframevr/sample-
assets/master/assets/images/space/earth_atmos_4096.jpg" radius="0.5" position="0 0.5 0" segments-height="53"
            animation="property: rotation; dur: 10000; to: 0 360 0; easing: linear; loop: true">
        </a-sphere>
      </a-marker>

      <!-- define your second marker -->
      <a-marker preset="kanji">
        <!-- here define the content to display on top of the marker -->
        <a-sphere src="https://cdn.aframe.io/360-image-gallery-boilerplate/img/cubes.jpg" radius="0.5" position="0
0.5 0" segments-height="53" animation="property: scale; dur: 1000; from: 1 1 1; to: 2 2 2; dir: alternate; easing:
easeInOutCirc; loop: true">
        </a-sphere>
      </a-marker>
      <!-- define the camera -->
      <a-entity camera></a-entity>
    </a-scene>
</body>
```

You should obtain something like this:

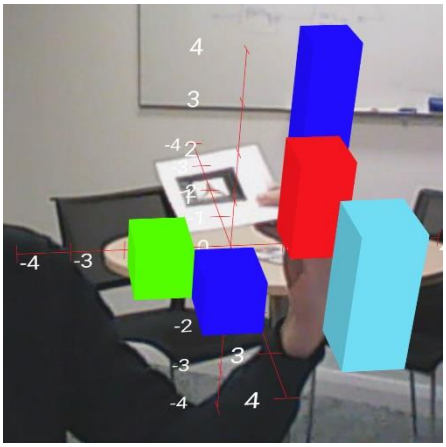


Exercise 9. AR to visualise data. Now let's put all together to visualise data. We will use a marker to interact with the data. Please do the following:

```
<head>
    <script src="https://aframe.io/releases/1.6.0/aframe.min.js"></script>
    <script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar.js"></script>
    <script src="https://unpkg.com/aframe-charts-component/dist/aframe-charts-component.min.js"></script>
</head>

<body>
    <a-scene embedded arjs='sourceType: webcam;'>
      <!-- simulate a json file -->
      <a-asset>
        <a-asset-item id="data" src='[
```

```
        {"x": 1, "y": 2, "z": 0, "size": 1, "color": "#ff0000"},
        {"x": -2, "y": 1, "z": 1, "size": 1, "color": "#00ff00"},
        {"x": -1, "y": 1, "z": 2, "size": 1, "color": "#0000ff"},
        {"x": 2, "y": 3, "z": -3, "size": 1, "color": "#0000ff"},
        {"x": 1, "y": 2.5, "z": 3, "size": 1, "color": "#4CC3D9"}]'>
      </a-asset-item>
    </a-asset>

    <a-marker preset='kanji'>
      <a-entity charts="type: bar; dataPoints: #data; axis_length: 4"></a-entity>
    </a-marker>

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

The expected output is as follows:



3. Using A-Frame with R

R allows us to build VR and AR environment using A-Frame entities. This is achieved using the *aframer* and *arframer* packages to create VR and AR content, respectively. In order to use these packages, we need to install them first. Please do the following.

```
# install devtools, not needed if already installed
install.packages("devtools")

# install aframer and arframer
devtools::install_github("JohnCoene/aframer")      # select option 3 (none)
devtools::install_github("JohnCoene/arframer")     # select option 3 (none)

# load libraries
library(aframer)
library(arframer)
```

Exercise 10. Create A-Frame entities with R. Once the packages are installed, **we can create A-Frame entities in HTML** directly in R using the embed_aframe() function from the *aframer* package. In this example let's generate the same entities used in exercise 5. Please do the following and observe the results in the console:

```
# it creates the scenes in html code
embed_aframe(
  a_scene(
    a_dependency(),
    a_box(position = "-1 0.5 -3", rotation = "0 45 0", color = "#4CC3D9"),
    a_sphere(position = "0 1.25 -5", radius = "1.25", color = "#EF2D5E"),
    a_cylinder(position = "1 0.75 -3", radius = "0.5", height = "1.5", color = "#FFC65D"),
    a_plane(position = "0 0 -4", rotation = "-90 0 0", width = "4", height = "4", color = "#7BC8A4"),
    a_sky(color="#ECECEC")
  )
)
```

The above function returns the following HTML script, that can be used to build your environment:

```
<div>
  <a-scene style="width:100%;height:400px;" embedded>
    <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
    <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
    <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
    <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
    <a-sky color="#ECECEC"></a-sky>
  </a-scene>
</div>
```

As you can see, we can use the generated content in our HTML files to display virtual environments. However, the *aframer* package can also publish VR content directly to your web browser using the *browse_aframe()* function. Let's try the following and observe the results in your browser:

```
# it runs the scene in web browser
browse_aframe(
  a_scene(
    a_dependency(),
    a_box(position = "-1 0.5 -3", rotation = "0 45 0", color = "#4CC3D9"),
    a_sphere(position = "0 1.25 -5", radius = "1.25", color = "#EF2D5E"),
    a_cylinder(position = "1 0.75 -3", radius = "0.5", height = "1.5", color = "#FFC65D"),
    a_plane(position = "0 0 -4", rotation = "-90 0 0", width = "4", height = "4", color = "#7BC8A4"),
    a_sky(color="#ECECEC")
  )
)
```

Exercise 11. Create AR content with R. Similarly, we can publish AR environment in your web browser directly in R using the *browse_aframe()* function from the *arframer* package. Now, lets create the same example than exercise 1, but this time using R. Please do the following and observe the results in your web browser:

```
# Generate AR content
browse_aframe(
  ar_scene(
    a_box(position = "0 0.5 0", material = "opacity: 0.5;"),
    a_primitive("marker-camera", list(preset = "hiro"))
  )
)
```