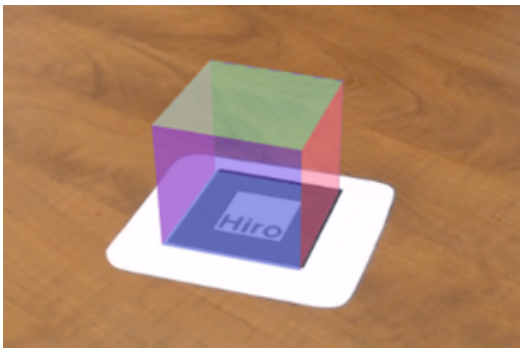
Understanding the nature of ARJS

Welcome to an introduction to using the ARJS platform. ARJS stands for Augmented Reality (AR) in JavaScript (JS). It is a plugin-free, web-based technology, which enables you to use marker or markerless images (such as a picture of some flowers) to show a virtual object.

Below is an example of using the commonly known “Hiro marker” and what could happen when you launch a custom-built AR application to scan this marker (using your webcam or smartphone camera):



Augmented Reality is considered an “improvement on existing reality.” Typically, this type of technology enables you to “see” virtual objects that do not exist in real-life. Remember: you can only see these virtual objects by launching a custom-built AR application and using a webcam (or your smartphone camera) to scan a specific marker.

If you’re lucky enough to have had a go on one, The Microsoft Hololens takes it to the next level by offering the whole experience in a headset.

The University of Gloucestershire own one of these unique devices, enabling you to experience AR in a wireless headset!

It is not just 3D primitive objects that can be displayed (e.g. box, torus or sphere), but anything that nowadays, is in a digital form, e.g. a custom 3D model of a game character, a YouTube video, an audio track or a virtual image gallery.

A well-known example of this in today's society is Pokémon Go – a game enabling you to catch virtual Pokémon in the real-world, using your smartphone and Global Position System (GPS).

A more practical example of AR is the IKEA mobile app – an app enabling you to virtually place IKEA furniture (objects) in your own home to find out how it would look, before purchasing.

In this tutorial, you will learn how to create your first Augmented Reality experience using the ARJS platform.

The technologies involved

There are several ways by which Augmented Reality experiences can be developed. Some use bespoke software applications and plugins to create the experience (e.g. Unity + Vuforia), whilst others provide you with just an Application Programming Interface (API).

ARJS relies on bringing together a range of sophisticated web API frameworks (sometimes known as libraries), using predominantly HTML5 and JavaScript:

a) https://aframe.io (file: aframe.js) – a Web framework for building virtual reality experiences.

b) https://ar-js-org.github.io/AR.js-Docs/ (file: aframe-ar.js) – a lightweight library for Augmented Reality on the Web, coming with features like Image Tracking, Location based AR and Marker tracking.

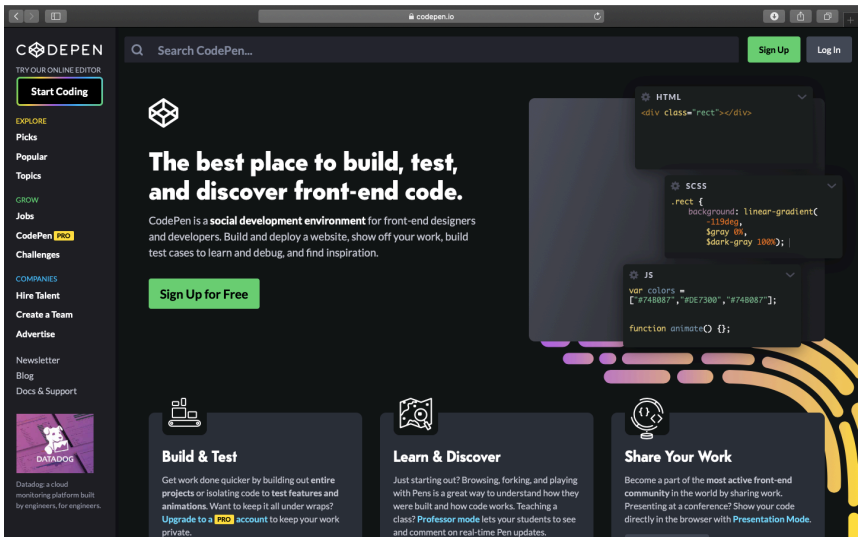
c) https://threejs.org (file: three.js) – a lightweight cross-browser JavaScript library / API used to create and display animated 3D computer graphics on a Web browser.

Feel free to take a moment to discover some of the examples on these web sites.

Setting up your workspace

This tutorial will require you to have access to a **webcam** or your **smartphone camera**.

1. Navigate to the following URL: https://codepen.io

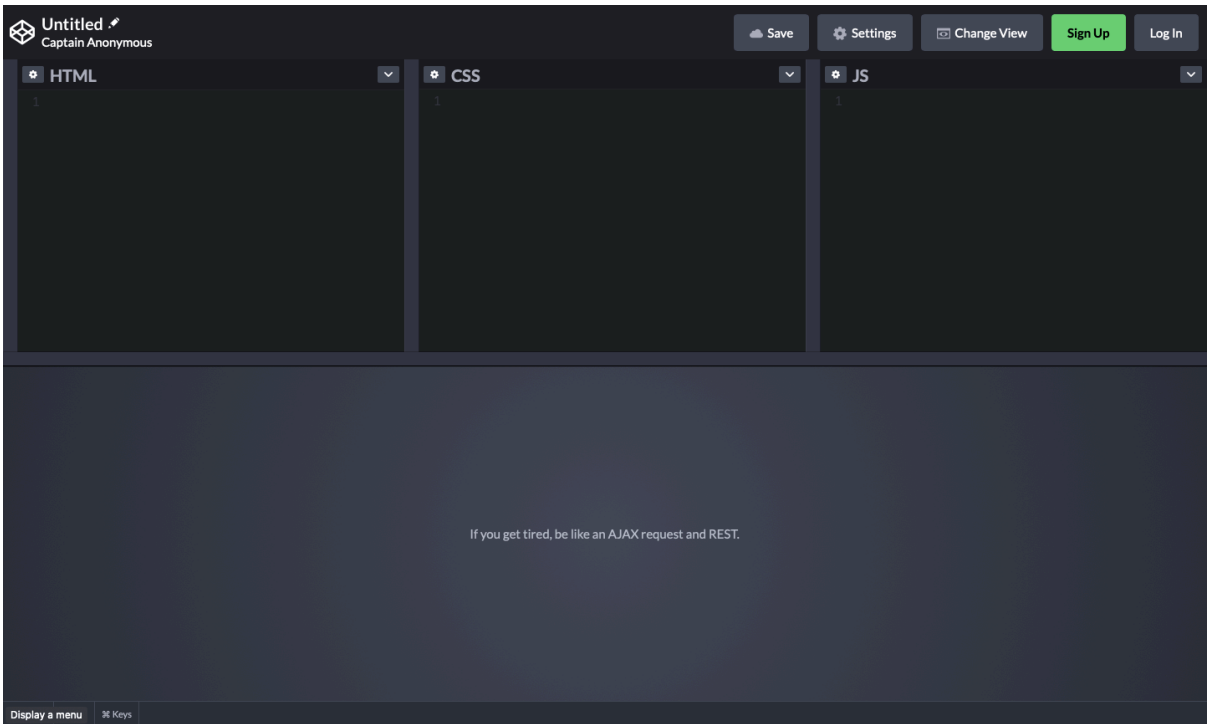


CodePen is a free online tool to let you “build, test, and discover front-end code” in real-time. If you sign up with CodePen, you can also save your “Pens” for future reference. It is a very useful online resource to quickly test the output of any kind of web design and development you undergo.

For the purpose of this example, we can get straight into coding without signing up.

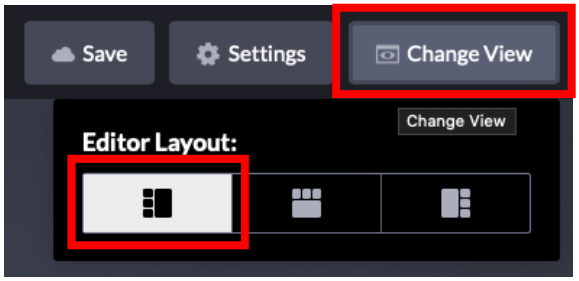
2. Click the **Start Coding** button at the top-left.

3. The default interface will load and present you with three key panels to type in: **HTML**, **CSS** and **JS** and one panel to **Preview** the outcome of your development.

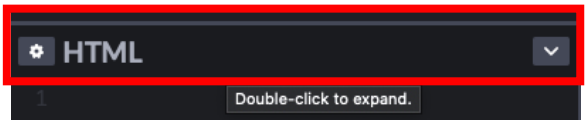


4. For the purpose of this example, we do not need access to the CSS or JS panel. However, we will need a larger Preview panel as it will be previewing your camera.

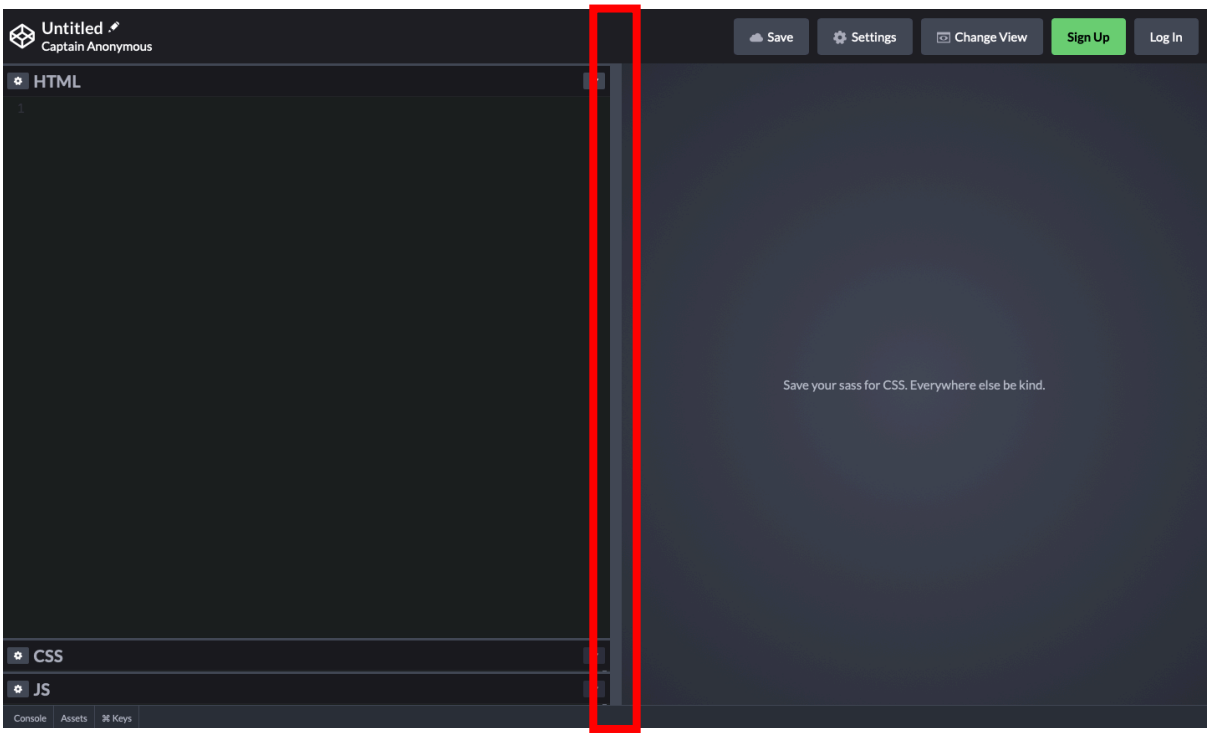
   Click the **Change View** button at the top-right and under the **Editor Layout** select the option to dock the coding panels to the **Left**



5. Following this, collapse the **CSS** and **JavaScript** panels. You can do this quickly, by **double-clicking** the **HTML** bar.



6. Expand the **HTML** panel to give you some room to easily see the code you type. You can do this by **click and dragging** the **middle divider**. Your workspace should now look something like below:



Time to get coding!

7. Type the following code:

```
<script
src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>

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

These two `<script>` tags import two of the key libraries to get you started: **aframe** and **aframe-ar**.

8. Continue typing the following code:

```
        <body style="margin: 0px; overflow: hidden;">
```

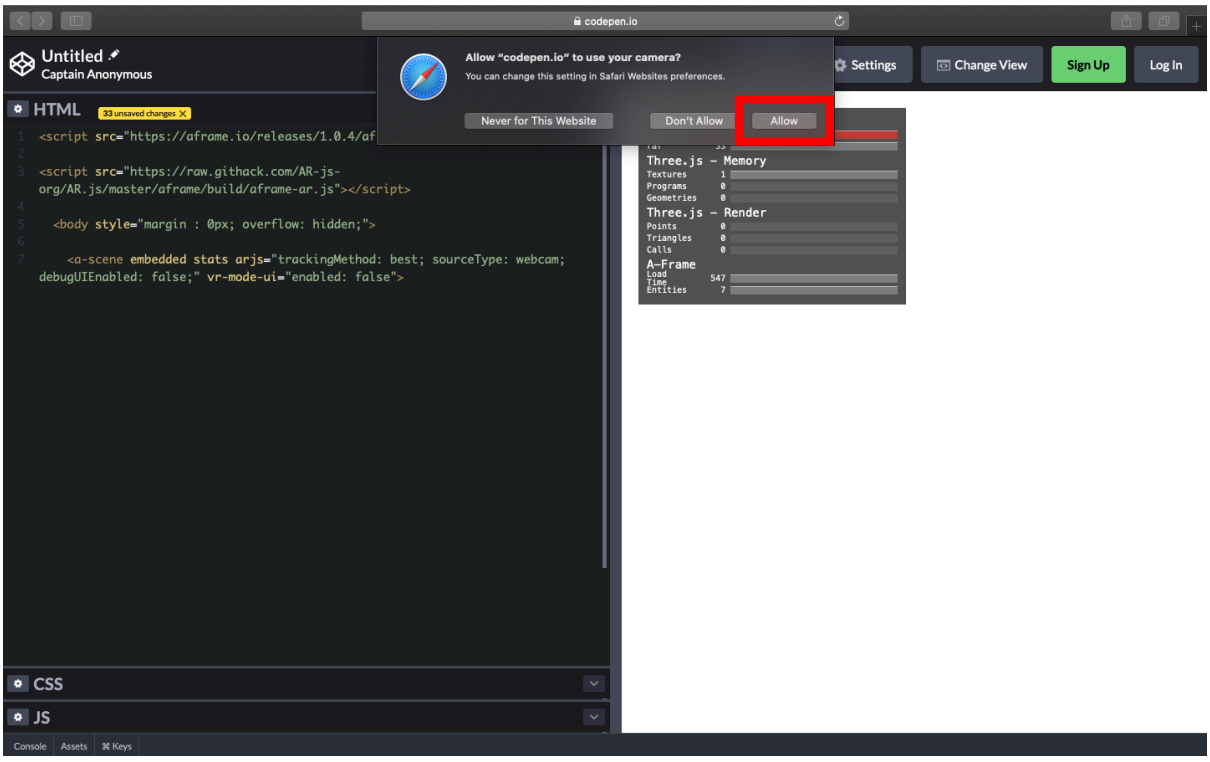
This `<body>` HTML tag initialises the main section of the page – the body of the page. Some additional `style` has been applied to the body to stop the output of the page from being scrollable.

9. Continue typing the following code:

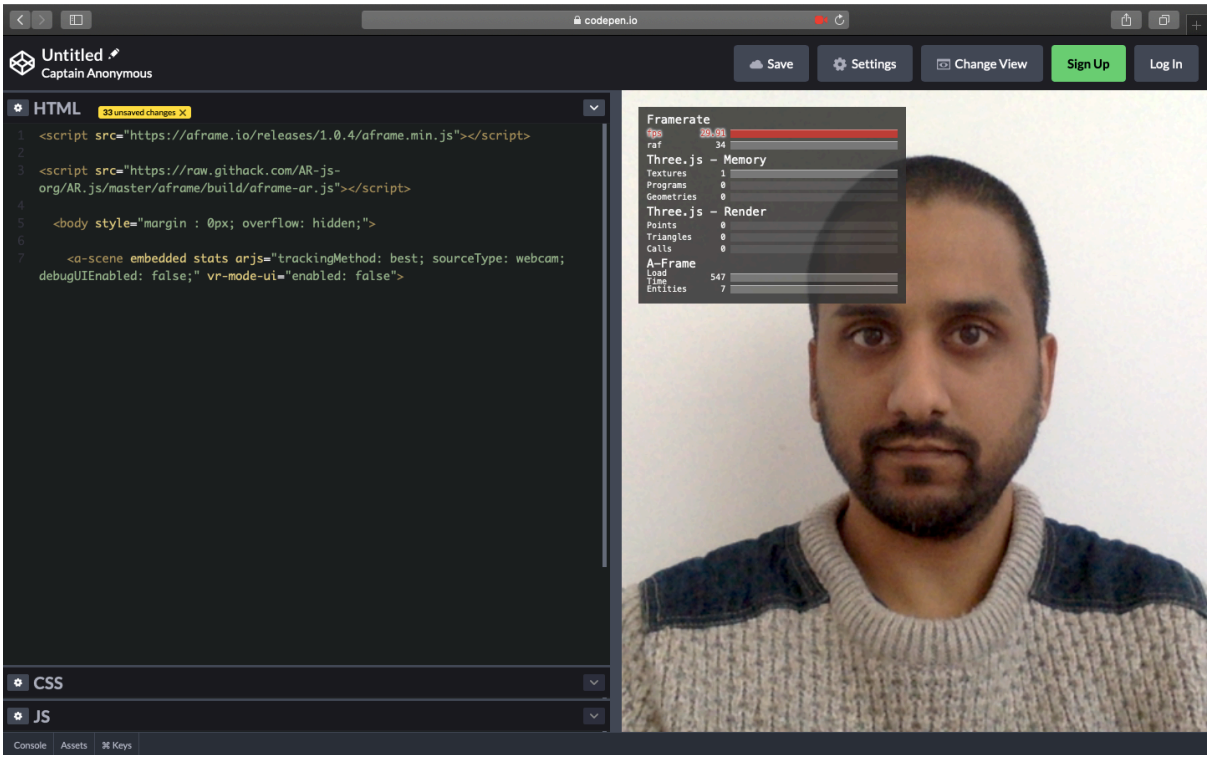
```
            <a-scene embedded stats arjs="trackingMethod: best;
sourceType: webcam; debugUIEnabled: false;" vr-mode-ui="enabled:
false">
```

Using the aframe framework we begin by initialising an aframe specific HTML tag called an `<a-scene>`. This defines a Scene in which virtual objects can exist. Note that several options are being defined to configure aframe's behaviour. One of these is the `stats` option. This provides some useful statistics which will be overlaid on top of the camera, helping to identify any performance issues.

As soon as you finish typing this line of code, CodePen will automatically refresh. Your web browser will request your permission to access the camera. Click **Allow**.



The Preview pane will update with a live stream of your webcam.



10. Continue typing the following code:

```
                    <a-marker preset="hiro">

                        <a-box position="0 0 0" rotation="0 0 0"
color="#F00" opacity="0.5" animation="property: rotation; to: 360
360 0; loop: true; dur: 5000; easing: linear" ></a-box>

                    </a-marker>
```

We now add a reference to an aframe marker using the `<a-marker>` HTML tag. An optional property `preset` is included to tell aframe to use one of aframe's pre-existing marker designs – the Hiro marker.

With the marker now registered by aframe, we ask it to display an aframe box (cube) using the `<a-box>` HTML tag. We also apply several properties to this box. These include:

a) `position` (more information)
b) `rotation` (more information)
c) `color` (hexadecimal colour value)
d) `opacity` (how transparent the box is – ranges from 0 to 1) and finally
e) `animation` (more information) – in this case, to take 5 seconds to rotate around the x and y axis 360º, indefinitely and at a constant speed.

11. Continue typing the following code:

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

We finish off the aframe's Scene by initialising the `<a-entity>` HTML tag and associating the device's camera to it.

12. Continue typing the following code:

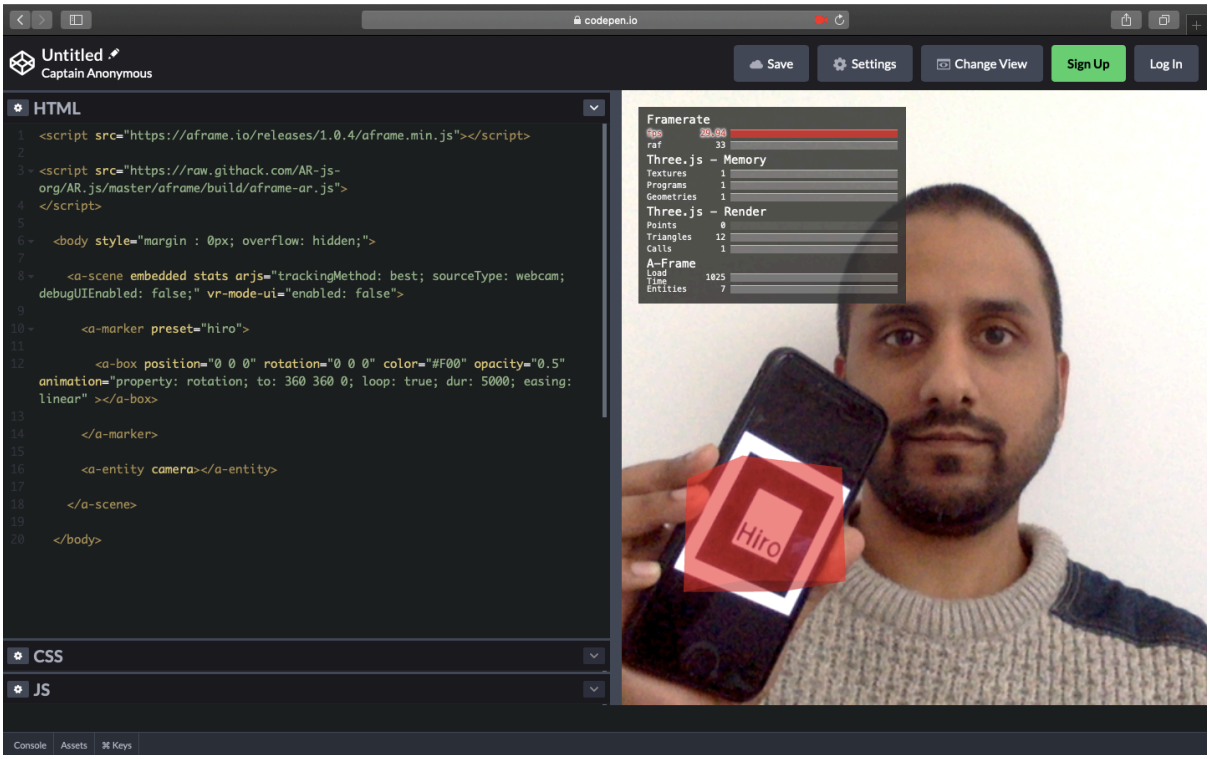
```
        </a-scene>

    </body>
```

The core of ARJS is now complete and we close off any remaining HTML tags that were previously opened.

Testing time!

The project is now complete. The next step is to print off the Hiro marker on some paper (or use your smartphone to load a picture of the Hiro marker on the screen).

Present the marker facing towards the camera and a semi-transparent, rotating AR box should appear in its place. Try casually moving the marker around (whilst facing it towards the camera) and watch the AR box continually be tracked.



Congratulations! You have managed to create your first Augmented Reality experience with marker tracking.

Move the camera or move the marker?

There are two types of tracking algorithms:

a) The method covered in the tutorial above relies on the **camera** being **stationary** (e.g. a built-in webcam on a laptop) and the **marker moving** around.

b) The alternative scenario would be the **camera moving** (e.g. a moving smartphone camera) and the **markers** being **stationary**.

ARJS enables you to use one or the other. See here for more information:
https://aframe.io/blog/arjs/#move-the-camera-or-the-marker

To use type b) tracking, carry out the following instructions:

13. **Replace Steps 10** and **11** with the following code:

```
<a-entity camera rotation="0 0 0" animation="property: rotation; to:
360 360 0; loop: true; dur: 5000; easing: linear">

       <a-marker-camera preset="hiro">

              <a-box position="0 0 0" rotation="0 0 0" color="#F00"
opacity="0.5"></a-box>

       </a-marker-camera>

</a-entity>
```

The key difference here is how the camera has been set up. It has been set up as an `<a-entity>` with the `animation` attached to it.

Also, rather than associate the `preset` Hiro marker to an `<a-marker>`, it gets attached to the `<a-marker-camera>` HTML tag. Finally, the `<a-box>` is contained within the `<a-marker-camera>`

Therefore, as the camera moves around and comes across the Hiro marker, it will map the `<a-box>` co-ordinates to the position of where the stationary Hiro marker is.

Upon testing the above, you will hopefully be able to see the difference for how the box tracking reacts when pointing / moving the camera towards the marker.

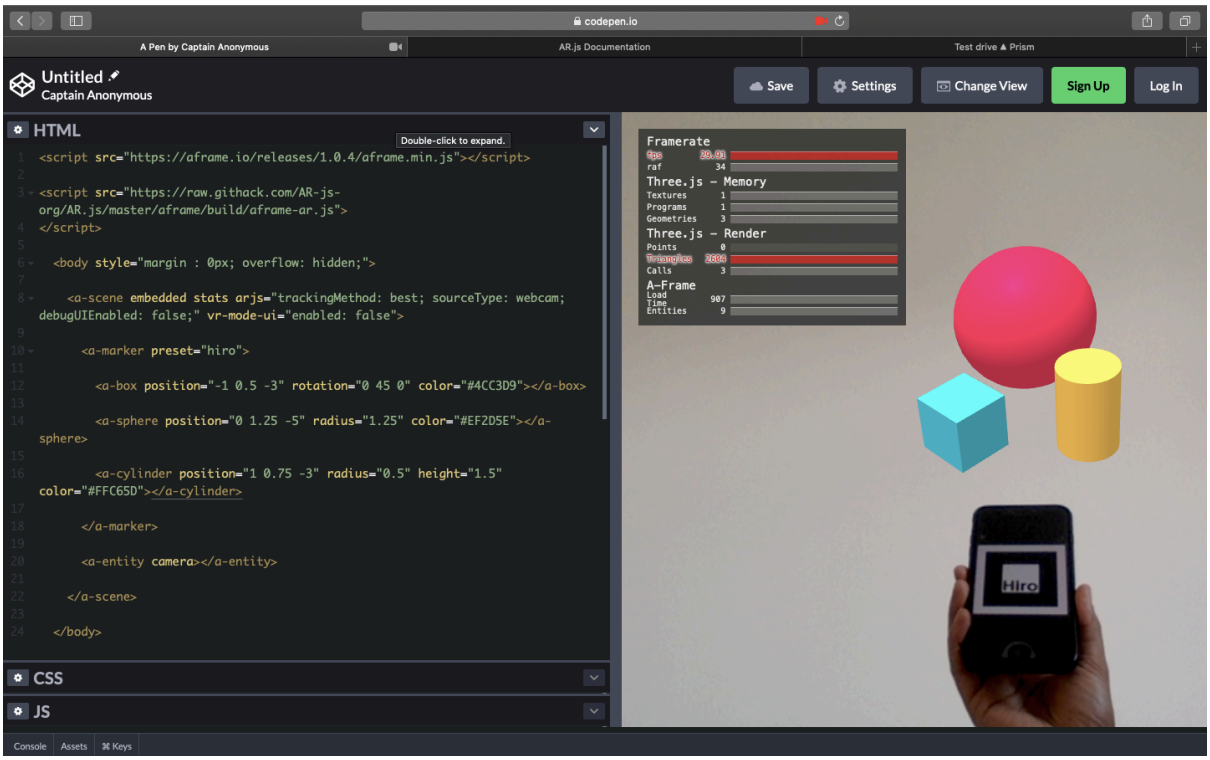
Feel free to switch back to using type a) tracking at any time.

Where to go from here…

1. Consider signing up to CodePen and saving your Pen for future reference. You could also make some additional ARJS examples as separate Pens.

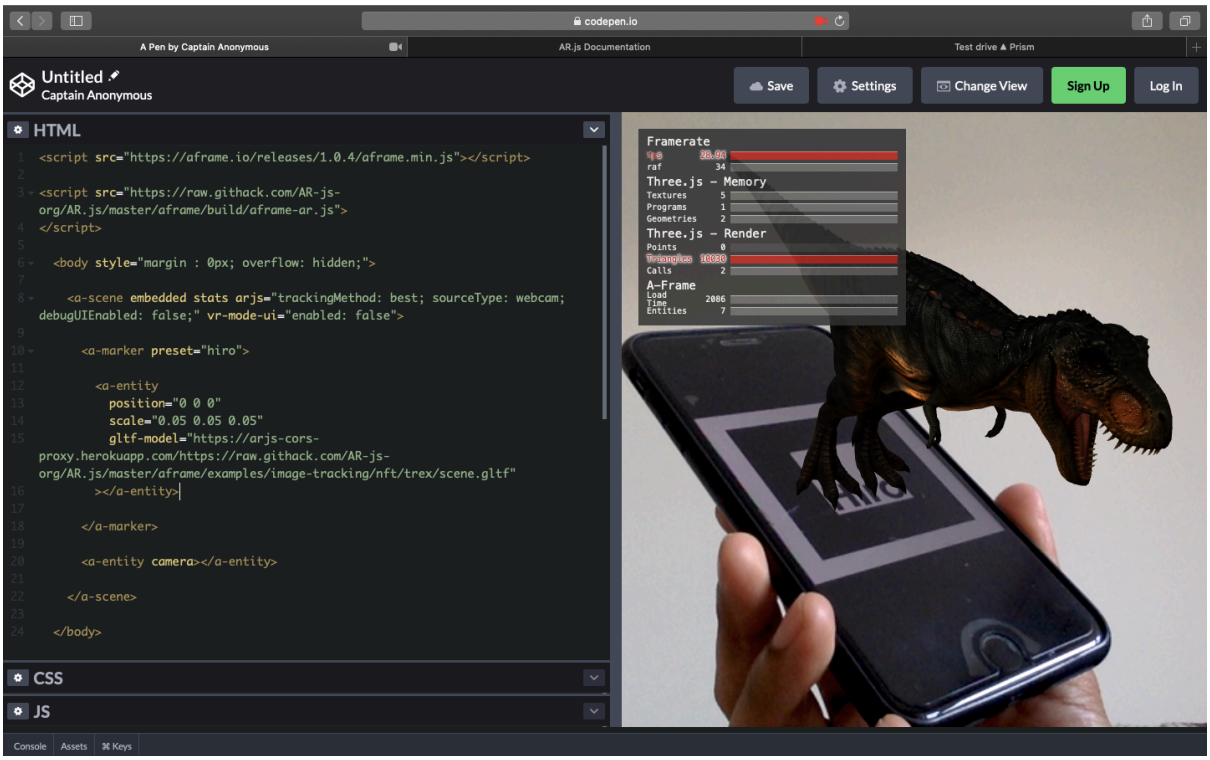
   Once saved, CodePen will provide you with a unique Pen link which you can navigate to on your smartphone. You can test / view the AR object using your smartphone camera.

2. Try changing the `<a-box>` to a different primitive that comes built in as part of the aframe framework – you can even load more than one at the same time and space them out using their `position` property. Take a look here for some additional examples: https://aframe.io/docs/master/introduction/html-and-primitives.html

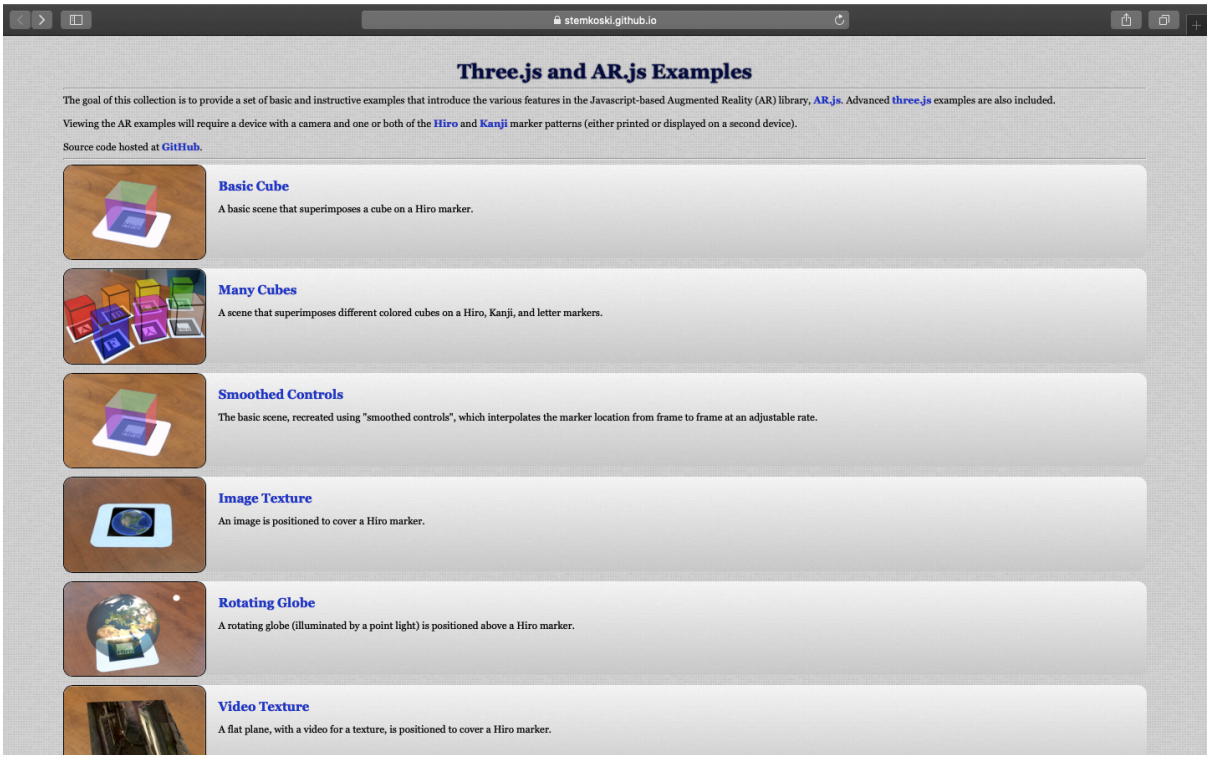


3. You can even replace the `<a-box>` with this `<a-entity>` to load in a custom model instead (more information):

```
<a-entity position="0 0 0" scale="0.05 0.05 0.05" gltf-
model="https://arjs-cors-
proxy.herokuapp.com/https://raw.githack.com/AR-js-
org/AR.js/master/aframe/examples/image-
tracking/nft/trex/scene.gltf">
</a-entity>
```

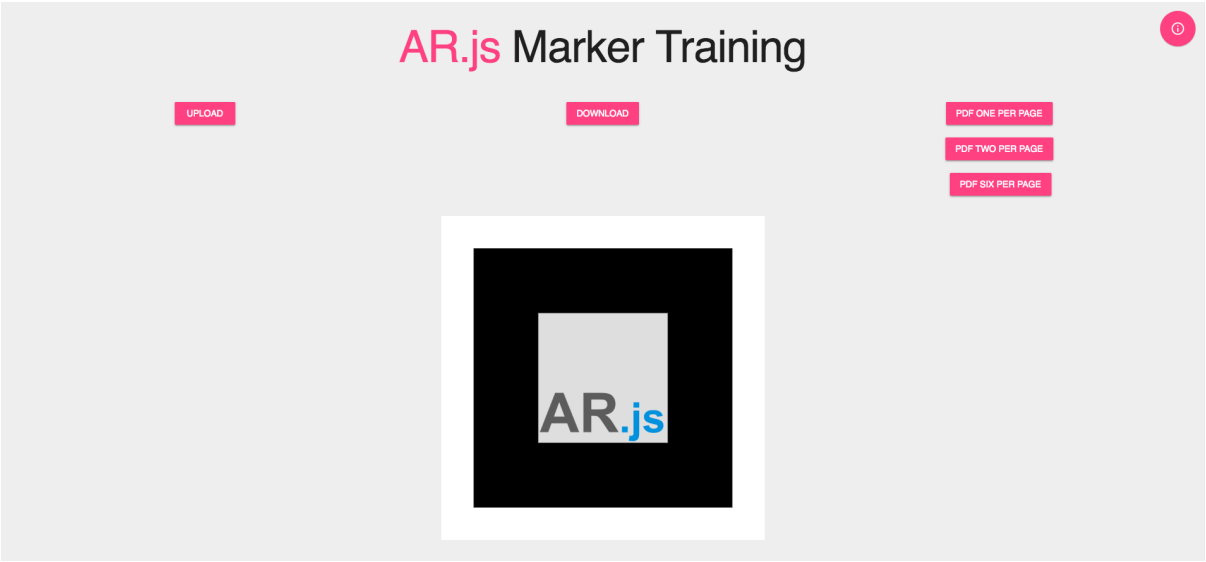


4. Take a look at these more advanced examples: https://stemkoski.github.io/AR-Examples/



You can try them out using the Hiro marker and / or the Kanji marker. The source code is a little more advanced, but interesting to glance at to get an idea of what it takes to create such examples.

5. Finally, perhaps explore how you would go about creating your own custom designed markers to trigger AR objects: https://medium.com/arjs/how-to-create-your-own-marker-44becbec1105



6. Bonus challenge: can you create a markerless AR experience? See here for more information: https://ar-js-org.github.io/AR.js-Docs/image-tracking/