Workshop A-Frame

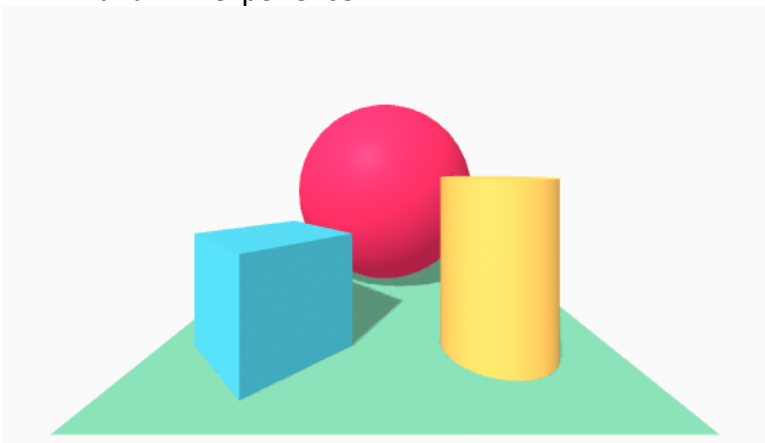
A-frame is a web framework for building virtual reality experiences. It work with basic web technology such as HTML and Javascript and it can be run on your computer or smartphone. For the latter, support for head-mounted VR (e.g. Google Cardboard) is built into the framework. The underlying technology behind A-frame is webXR (the successor of webVR), a technology to run Virtual Reality and Augment Reality experiences in browser.

The point to this workshop is to get acquainted with A-frame and how to build virtual environments with it. And to find where you can learn more about the framework so you can apply it to your own projects. We will cover the topics of: shapes, textures, animation and interaction.

01 Hello World

Let's get started by setting up all you need to develop you VR environments. Some of the functions of A-frame only work when the files are hosted on a webserver. The easiest way to do so is to use Github and Github Pages.

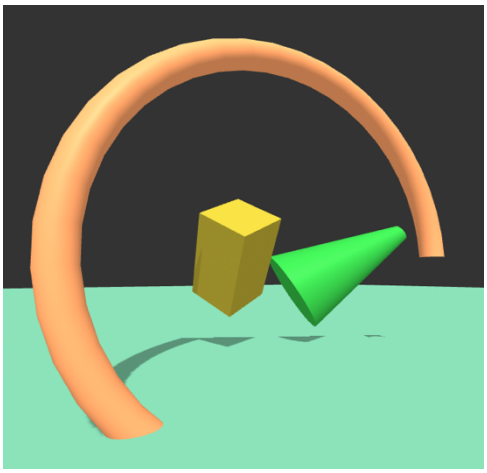
1. Make a new public Github repository for this workshop. Please pick a descriptive name for your repository.
2. Copy the workshop files to this new repository. Make sure you don't change anything to the filenames or folders.
3. Commit and publish.
4. In your repository on the Github Website, click on the settings tab. Scroll down to the *Github Pages* section and choose *master branch* as your source. (If this option is unavailable, check if your repository is public. This can be changed under setting > Danger zone)
5. A URL to the hosted website should be available now. Open the webpage in your Google Chrome browser and then click on the link for *01HelloWorld.* A VR environment should now op in your browser.
6. Open the same page on your smartphone. Move your phone around to see the whole environment. Click the VR button and load your phone in your VR headset for a full VR experience.



02 Shape

You have now seen a basic example of some shapes in VR. Let's start making our own shapes.

1. In the *index.html* file for this exercise, you will find the basic structure for building a virtual environment. In the head section, you can see a link to the A-frame JS library. This library makes all the A-frame functionality available to us.
   Also notice that the *body* of this page only (directly) contains one element. Namely the *a-scene* element. This *a-scene* element includes the entire virtual environment and contains all the 3D shapes and others elements.
2. Open the file in the browser. You should see a box. If not, you might need to move around a bit. You can use the *W,A,S,D*-keys to move.
3. The *a-box* element has a *width*, *height*, *depth* and *color* attribute. Change the values to make this box your own.
4. To move the box around, add a position attribute. `position="-1 3 1"`. This attribute expects a list of three values. For moving in the X, Y and Z direction respectively. Change the box's position so it is in the center of the screen when the scene is loaded.
5. In the same way position moves the box around, rotation and scale also manipulate the box's properties. They also expect a list of three values for X, Y and Z. Try them out.
6. Now add some more primitive shapes. Choose the shapes from the list below. Not every shape has the same attributes. Look up how to use the shape you want on https://aframe.io/docs/1.0.0/introduction/ (scroll down in the left menu to find the primitives section)
   - <a-circle>
   - <a-cone>
   - <a-cylinder>
   - <a-dodecahedron>
   - <a-octahedron>
   - <a-plane>
   - <a-ring>
   - <a-sphere>
   - <a-text>
   - <a-tetrahedron>
   - <a-torus-knot>
   - <a-triangle>



tip: to make it look like there is ground beneath your feet, use a rather large plane.

03 Textures

To make your virtual world look more interesting, you can wrap images around your objects. In the files for this exercise, you will find an *index.html* file which contains a box and a plane. Also in the images folder you will find a texture for wood and one for grass.

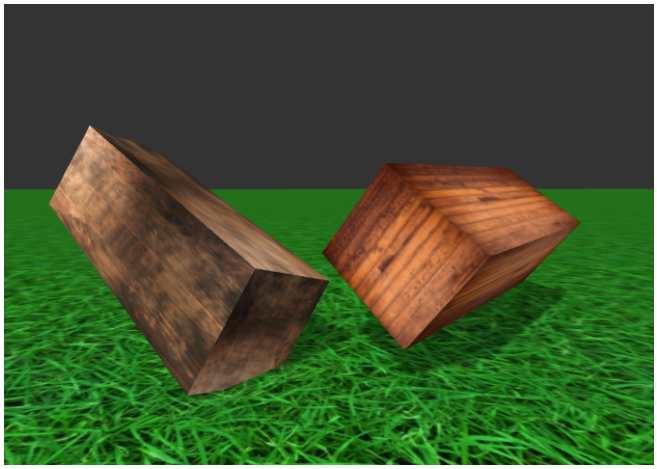
1. In the index.html file you will find the new *a-assets* tag and in it you will see an *img* tag. With the *a-assets* tag we can load images we use in the rest of the code. We give the images we load an id, so we can reference them later. This way we can load it once and refer to it many times. (this does not only apply to images, there are many assets you can use in A-frame)
2. Apply the wood by adding a *material* attribute to the box. `material="src: #wood"`. Here the # is used to signal that we are referencing an id, just like in CSS. Test your code in the browser. If it doesn't work, make sure you are using the webserver (eg. Github Pages) to view your page. You Will run into issues when viewing your site locally.
3. You will now apply the grass texture to the ground. Load the grass.jpg file as an asset and then apply the appropriate material to the plane.
4. If you applied the grass texture correctly, you might notice the blades of grass look way to big. That's because the image is stretched over the entire plane. Change the material attribute to `material="src: #grass; repeat: 15 15"` to remedy this. The texture image is now repeated 15 times in both directions over the entire plane. When you are looking for textures for your own projects, make sure you use seamlessly repeatable images if you are going to use them in this way.

There are many more ways in which you can use materials. More information here https://aframe.io/docs/1.0.0/components/material.html

04 Animations

Until now you have created some static objects. You can bring these objects to life using animation. There are multiple ways to do that, we will explore a few in this exercise.

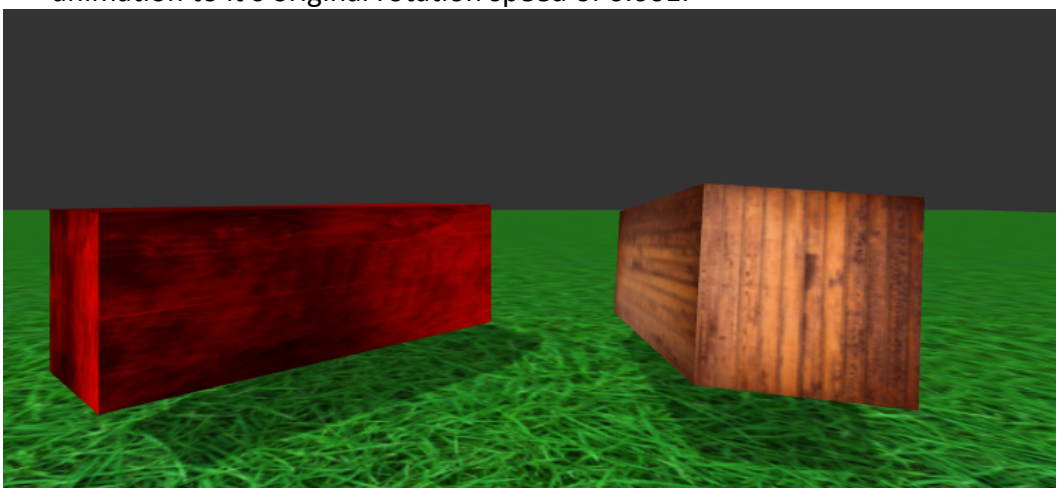
1. The first way you will use is the way provided in A-frame. In the index.html file you will see an animation is applied to the box with id="myBox". In the animation property, a couple of things are set: *property* determines the property to animate, *to* describes the state of that property at the end of the animation, *dur* is the duration in milliseconds, *easing* determines how the animation progresses and *loop* determines how many times the animation should loop or if it loops indefinitely. You can find more info about animation here https://aframe.io/docs/1.0.0/components/animation.html
2. Play around with the animation. Change values and see what happens. Make the box move in an interesting way. If you feel up for a challenge, try to combine multiple animations (https://aframe.io/docs/1.0.0/components/animation.html#multiple-animations)
3. This form of animation works fine, but sometimes you need more versatility. For instance, this way does not allow you to use variables very easily. In the *scripts/main.js* file, you will find some commented code. Uncomment it, test it and see what happens with the other box in the scene. In the next few exercises you will change the animation to get a firm grip on the way you could use JavaScript to manipulate the virtual environment.
4. The box rotates clockwise, change the code so it rotates counter clockwise.
5. The box only rotates around the X axis. Add to the code so it revolves around all the axis's
6. Change the speed at which the box rotates. Speed it up and then slow it down. Make sure you don't change the number that is set in the setInterval function.
7. Play around. See what other interesting animations you can come up with.



05 Interaction

In the virtual environment, we can interact with the objects. This exercise will show you two ways. The first way anly uses the A-frame tags trigger animations when the mouse is over a specific object. The second way uses JavaScript to add event listeners to specific events.

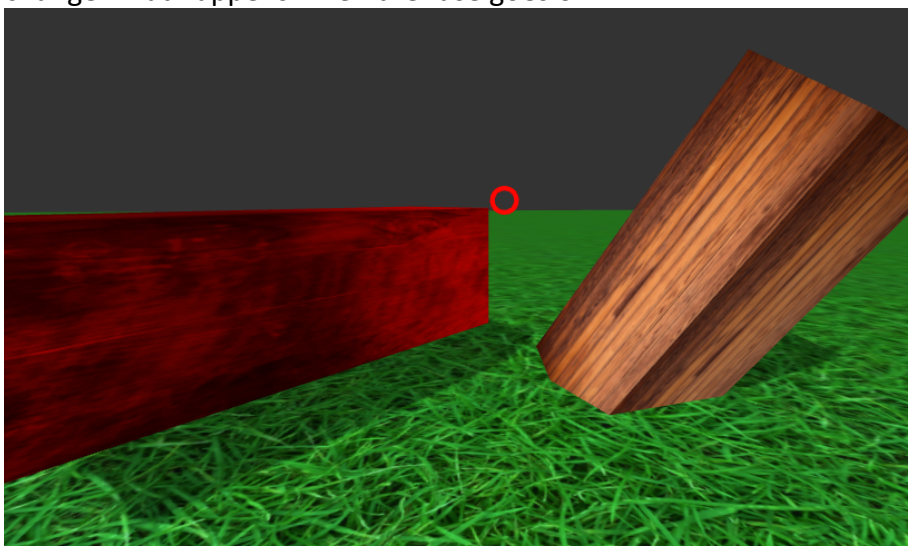
1. Try out the files for this exercise in your browser. Move your mousecursor over the left box and see what happens. In the index.html file you will see two animations, much like we saw in the previous exercise. One is called `animation__mouseenter` and the other is called `animation__mouseleave`. What makes this work is that both animations have `startEvents` specified. Here the events are set that trigger the start of the animation.
2. Play around with this interaction. For example, change to an other event or change the animation.
3. A second way of adding interaction is use JavaScript. In the scripts/main.js file, you will find a piece of commented code. Uncomment and run it. In the virtual environment, hover your mouse over the right box and see what happens.
4. The animation speeds up when the mouse enters the box, but then it will keep on animating at that speed even when the cursor leaves the box again. Add the code that adds a function to the mouseLeave event. This function should return the animation to it's original rotation speed of 0.001.



06 Interaction Headset

In the previous exercise you saw how interaction can be achieved by using your mouse. Interaction using the keyboard can be achieved in a similar manner. But that does not help us when we are experiencing the VR environment on our smartphone mounted in a headset when mouse and keyboard are unavailable. In this exercise you will get acquainted with two ways you can interact while wearing the headset[1].

1. Test the files for this exercise on your browser and with your headset. You will notice a small circle in the centre of your screen that moves together with your field of view. This is the virtual cursor that replaces the mouse and it works similar to crosshairs in videogames. The *mouseenter* and *mouseleave* event are now triggered by this virtual cursor.
2. In the *index.html* file, you will find an *a-camera* element is added. This is the virtual camera we look through. All elements you put inside the a-camera element, will become tied to the camera view. They basically become a heads-up-display. Inside this *a-camera* element you find an *a-entity* that represents the cursor and an *a-ring* element that visualises that cursor. Play around with the visualization of the cursor to make your own custom cursor.
3. In the *script/main.js* file you will find a part of code commented. Uncomment this and try it out. You will notice that the right box won't grow immediately when the cursor hits it. Instead it changes only then the cursor is over the box for 1.5 seconds. This interaction is called a fuse and is very handy for head mounted displays with limited controls. The fuse is attached to the *click* event. And in the *index.html* file it is stated how long the fuse is by setting a value for *fuseTimeout* in milliseconds. Play around with this fuse functionality. For example, change the length of the fuse and change what happens when the fuse goes off.



[1] . If you do not have access to a headset, try holding your phone close to your face (approximately 5cm) and focus your eyes somewhere in the distance until you see one 3d image.

Where to go from here

This is the end of the workshop. If you want to continue learning about VR with A-frame, I can highly recommend looking into the topics of lighting, skybox and importing OBJ models.

I also recommend watching this youtube playlist of tutorials. They have partially been the basis for this workshop. A small disclaimer is that these video tutorials have been made for an older version of A-frame. Hence, some of the code may not work in the current version.
https://www.youtube.com/watch?v=dv6_C4UqTfs&list=PLRtjMdoYXLf4inSULAHyCMqpIUj4cmBTr&index=1