CS + Social Good: VR Workshop

# IntroductioN

One of the benefits of virtual reality is the ability to bring important issues to light in innovative ways for a call to action. In this workshop, we are going to use A-Frame and WebVR to build a virtual reality-enabled website to animate disintegrating garbage in the ocean from a fishes’ point of view. These sites can be viewed in a standard browser, or, if you have a desktop VR device such as an Oculus Rift, you can download Firefox Nightly to view the site in VR.

In addition to working on desktop browsers, WebVR sites will also render *stereoscopically* on mobile devices. Combined with a Cardboard (or similarly styled mobile headset), you can turn any smart phone into a VR headset!

Firefox Nightly Download: <https://nightly.mozilla.org/>

Wallenberg – 323 (3:30)

# OUR INDEX.HTML File

We are going to start with two files: index.html, and aframe.min.js, both of which are located in the sample project directory. To start, we’ll open up index.html and add the first few lines of markup – these will be familiar if you’ve written a website before:

<!doctype HTML>

<html>

<head>

</head>

<body onload = “OnSuccessfulLoad()”>

</body>

</html>

We will first load in the A-Frame scripts in our head tags with the line:

<script src = “aframe.min.js></script>

Once we’ve done that, we’ll be able to use all of the A-Frame elements in our own page. This includes all of the support for virtual reality rendering and Three.JS – there’s a lot included in that one JavaScript file!

# Setting Up The A-FRame Scene

The next thing that we’re going to want to do is set up our initial A-Frame scene. All of the components in A-Frame are prepended with an “a-“ **tag**, such as <a-scene>, the first element that we’ll add.

Almost all HTML (or, in this case, A-Frame) tags require a closing tag </> at the end of the declaration. This lets us specify what elements get nested within one another.

Within the body tags, we’ll start by adding <a-scene> **tags** to surround our other elements. We’ll also give this an id (**sceneElement***)* so that we can access it later from a script to change the elements within the scene:

<body onload = “OnSuccessfulLoad()”>

<a-scene id = “**sceneElement**”>

</a-scene>

</body>

All of the elements that we will put into our application will go in between our <a-scene> tags.

The first element in our scene is going to be a camera. This will define what in our scene is visible within the frame of our computer screen, and will represent a user’s field of view into the scene when in VR. The camera element in A-Frame, like most computer graphics software, is highly customizable, but we’ll just use a basic example:

<a-scene id="sceneElement">

<a-camera **position = "0 5 0"**>

</a-scene>

If you are familiar with HTML, you’ll notice that the *attributes* of the A-Frame objects are written out the way you would write in standard CSS attributes for a regular webpage. In this case, we added the **position = “0 1 0”** attribute to define where in our scene the camera would be placed.

In the A-Frame coordinate system, +X can be roughly thought of as “right”, +Y is “up”, and “+Z” is towards the default camera.

Now that we have our camera, we can start adding in new items to our scene. We’ll start by putting in our background to our scene using the <a-sky> element.

Wikipedia’s video game graphics series has a good overview of skyboxes: <https://en.wikipedia.org/wiki/Skybox_(video_games)>

For our skybox, we’ll actually be using an *equilrectangular projection* of an underwater scene. This is the file **underwater.jpg** in our project directory. Since immersive experiences are all about covering multiple senses, we’ll also add in an ambient audio file to play sound. Lastly, to make our scene objects have textures, we’ll include two lights in our scene as well:

<a-scene id="sceneElement">

<a-camera **position = "0 1 0"**>

<a-sky src="underwater.jpg"></a-sky>

<a-entity sound="src:ocean.mp3; autoplay:true; loop:true; volume:0.5"></a-entity>

<a-entity light="type: directional; intensity: 0.3; color: #fff" position="-1 2 1"></a-entity>

<a-entity light="type: ambient; color: "></a-entity>

</a-scene>

## About the A-FRAME Entity-Component System

While there are some pre-defined objects that can be used within A-Frame (such as the a-camera and a-sky elements), what makes A-Frame so powerful is its *entity-component* system. This system, as seen above when we load in our sound file, will let us build more complex elements to add to our scenes. The *entity* is the overarching container of all of the *components* that we are adding, so in the above example, our entity is composed of a “sound” component with multiple characteristics: src [source], autoplay, loop, and volume.

# The Garbage

We are going to add an animation to our web page that shows the degeneration of a can in the ocean and how over time, massive amounts of ocean pollution break down into invisible particles that impact marine life and build up on the sea floor. Our perspective will be that of a small fish as we watch a single soda can break down into tiny particles.

Our scene is going to be made of a few things:

1. A soda can A-Frame entity
2. A container for our particles to explode out of the can
3. JavaScript functions to create and animate the particles and “dissolve” effect

## Adding & Animating a Soda Can

Although we won’t go into too many details about 3D graphics here, there are a few terms to know before we start working on our 3D objects in our scene:

**Geometry:** the shape that our object’s *mesh* will take

**Material:** the way that the object will look – how lights will reflect

**Texture:** any print or visual image that the object will take on

For our can, we’ll be creating a new <a-entity> with the following components. The format for the component-entity pair is generally component=string. Some components may have multiple attributes defined, separated with a semi-colon.

Can Entity-Component Attribute Table

|  |  |
| --- | --- |
| **Component** | **String** |
| id | “can” |
| geometry | “"primitive:cylinder; radius:2; height:5" |
| position | "0 2 -10" |
| rotation | "-30 30 12" |
| material | "color:#9E9F9F; roughness:0.2; metalness: 0.7; src:url(label.jpg)" |

Our composed entity element will look like this:

<a-entity id="can" geometry="primitive:cylinder; radius:2; height:5" position="0 2 -10" rotation="-30 30 12" material="color:#9E9F9F; roughness:0.2; metalness: 0.7; src:url(label.jpg)">

Notice, though, that there isn’t a closing tag for our can entity – this is important! We don’t want our can to just sit in one place, so we’re going into introduce a new type of A-Frame entity: the a-animation.

AFrame animation elements work in the same way as other components do, with a few small changes. They also work quite similarly to CSS animations, so these may look familiar! We’ll be adding in two different animations to our soda can:

**FLOAT ANIMATION ENTITY-COMPONENT ATTRIBUTE TABLE**

|  |  |
| --- | --- |
| **Component** | **String** |
| **id** | **“float”** |
| **attribute** | **“position”** |
| **from** | **“"0 2 -10"** |
| **to** | **"0 -2 -10"** |
| **dur** | **“10000”** |

**<a-animation id="float" attribute="position" from="0 2 -10" to="0 -2 -10" dur="10000"></a-animation>**

**SHRINK ANIMATION ENTITY-COMPONENT ATTRIBUTION TABLE**

|  |  |
| --- | --- |
| **Component** | **String** |
| **attribute** | **“scale”** |
| **to** | **“0 0 0”** |
| **begin** | **“resize”** |
| **dur** | **“10000”** |

**<a-animation attribute="scale" to="0 0 0" begin="resize" dur="10000"></a-animation>**

**Straightforward animations are just one of the cool features of A-Frame. All combined, our can entity will look like this:**

**<a-entity id="can" geometry="primitive:cylinder; radius:2; height:5" position="0 2 -10" rotation="-30 30 12" material="color:#9E9F9F; roughness:0.2; metalness: 0.7; src:url(label.jpg)">**

**<a-animation id="float" attribute="position" from="0 2 -10" to="0 -2 -10" dur="10000">**

**</a-animation>**

**<a-animation attribute="scale" to="0 0 0" begin="resize" dur="10000"></a-animation>**

**</a-entity>**

**Add that directly following your sound, and take a look at the scene in the browser! You might notice that our can isn’t actually shrinking – we’ll get to that in just a minute!**

# **The groundwork for particles**

To represent the degrading particles in our system, we’re going to need to add in another A-Frame entity. However, we don’t want this to show just yet, so we’re going to introduce another aspect of the A-Frame framework: The a-asset component.

A-asset tags surround other A-Frame elements that we’ll reference later on in our code, but don’t want to include in our scene immediately.

We’re going to add one new entity into our assets: a *container* entity to hold our particles. Our container entity will have two animations as part of it to animate the exploding apart and sinking effects.

**CONTAINER ENTITY-COMPONENT ATTRIBUTION TABLE**

|  |  |
| --- | --- |
| **Component** | **String** |
| **id** | **“container”** |
| **position** | **“0 2 -10”** |
| **scale** | **“0 0 0”** |
| **Dissolving Animation** | |
| **attribute** | **“scale”** |
| **from** | **“0 0 0”** |
| **to** | **“1 1 1”** |
| **dur** | **“10000”** |
| **begin** | **“breakApart”** |
| **Sinking Animation** | |
| **attribute** | **“position”** |
| **from** | **“0 2 -10”** |
| **to** | **“0 -30 -10”** |
| **dur** | **“50000”** |
| **begin** | **“sink”** |

**The overall entity will look like the following:**

**<a-entity id="container" position="0 2 -10" scale="0 0 0">**

**<a-animation attribute="scale" from="0 0 0" to="1 1 1" dur="10000" begin="breakApart"></a-animation>**

**<a-animation attribute="position" from="0 2 -10" to="0 -30 -10" dur="50000" begin="sink"></a-animation>**

**</a-entity>**

# **Putting it all together**

**With that, we have our basic A-Frame scene! The total scene should look like this:**

**<a-scene id="sceneElement">**

**<a-assets>**

**<a-entity id="container" position="0 2 -10" scale="0 0 0">**

**<a-animation attribute="scale" from="0 0 0" to="1 1 1" dur="10000" begin="breakApart"></a-animation>**

**<a-animation attribute="position" from="0 2 -10" to="0 -30 -10" dur="50000" begin="sink"></a-animation>**

**</a-entity>**

**</a-assets>**

**<a-camera position="0 5 0">**

**<a-sky src="underwater.jpg"></a-sky>**

**<a-entity sound="src:ocean.mp3; autoplay:true; loop:true; volume:0.5"></a-entity>**

**<a-entity id="can" geometry="primitive:cylinder; radius:2; height:5" position="0 2 -10" rotation="-30 30 12" material="color:#9E9F9F; roughness:0.2; metalness: 0.7; src:url(label.jpg)">**

**<a-animation id="float" attribute="position" from="0 2 -10" to="0 -2 -10" dur="10000"></a-animation>**

**<a-animation attribute="scale" to="0 0 0" begin="resize" dur="10000"></a-animation>**

**</a-entity>**

**<a-entity light="type: directional; intensity: 0.3; color: #fff" position="-1 2 1"></a-entity>**

**<a-entity light="type: ambient; color: "></a-entity>**

**</a-scene>**

# **adding JavaScript**

You may recall that when we added in our **<body>** tags earlier, we added in a line within those tags to define our page’s **onload** function (in our case, **onload = “OnSuccessfulLoad()”**. We’re going to add an on load function, as well as 3 additional helper functions, in JavaScript on our page now.

The first thing that we are going to do is add in **<script>** tags to our page directly underneath the **</a-scene>** tag, and include the outlines of our four functions:

<script>

function OnSuccessfulLoad(){

}

function ShrinkCan(){

}

function EmitParticles(){

}

function SinkParticles(){

}

</script>

The function **OnSuccessfulLoad()** will be called once our page has finished loading in the browser, but the other three will be called as the previous animation completes. Before we add anything to our functions, we want to create a JavaScript variable that represents our container entity above, so we’ll put the following two lines of code in directly under the <script> tags, before our first function:

var entityElement = document.getElementById("container")

document.getElementById("sceneElement").appendChild(entityElement)

This code will create a variable for us to add additional A-Frame elements to within our JavaScript code to programmatically populate our particle effect in our A-Frame scene. The first line creates the element by finding our container A-Frame entity, and the second adds this asset to our scene. We’ll be using the **appendChild** JavaScript function frequently as we add in more elements to our scene.

## Function OnSuccessFulLoad()

When we load in our page, we are going to immediately want to populate our **entityElement** object with particles to give the effect of the can disintegrating into smaller pieces. To do this, we will have two loops and set the **x** and **z** elements for each new particle sphere, create the sphere itself, and add it to our **entityElement,** which can be thought of as a collection of all of the tiny particles.

Copy the following code into your **OnSuccessfulLoad()** function:

var i;

for (i = 1; i < 10; i++) {

var r, x, z;

for (r = -10; r < 10; r++) {

// Create our metal particles

var particleElement = document.createElement("a-entity")

particleElement.setAttribute("geometry", "primitive:sphere; radius:0.1")

particleElement.setAttribute("material", "color:#cccccc; roughness:0.2; metalness:0.7")

x = Math.PI \* r \* Math.cos(i)

z = Math.PI \* r \* Math.sin(i)

particleElement.setAttribute("position", x \* 1.5 + " 2 " + z \* 1.5)

entityElement.appendChild(particleElement)

}

}

setTimeout(function(){ShrinkCan()}, 5000)

## Function ShrinKCan()

Our animation to create the effect of a dissolving can will be two-fold: First, we want the can to shrink in size as particles begin to fly off of it. In our above function, we call **ShrinkCan()** after 5 seconds, so let’s write it! Our function will be short, just two lines of code:

document.querySelector("#can").emit("resize")

EmitParticles()

## Function EmiTParTicles()

The second half of our dissolve animation will shoot out the particles in our **entityElement** by calling our animation function on them. Our function to emit the particles will look like this:

entityElement.emit("breakApart")

setTimeout(function () { SinkParticles() }, 5000)

## Function SinkParticles()

Lastly, we will sink our particles over the course of several seconds to showcase how garbage particles in the ocean accumulate on the ocean floor to do this, we will call our **emit** function on our last animation within our SinkParticles() function with the line:

entityElement.emit("sink");

# Running the App

To run the app, load up your index.html file in a browser! If you use a VR-enabled browser with a VR device attached, you will be able to view your file in stereoscopic 3D. You can also view it in VR mode on a mobile phone by setting up a web server and navigating to the file that way.

Enjoy!

# The Full Code

<!doctype HTML>

<html>

<head>

<script src="aframe.min.js">

</script>

</head>

<body onload="OnSuccessfulLoad()">

<a-scene id="sceneElement">

<a-assets>

<a-entity id="container" position="0 2 -10" scale="0 0 0">

<a-animation attribute="scale" from="0 0 0" to="1 1 1" dur="10000" begin="breakApart"></a-animation>

<a-animation attribute="position" from="0 2 -10" to="0 -30 -10" dur="50000" begin="sink"></a-animation>

</a-entity>

</a-assets>

<a-camera position="0 5 0">

<a-sky src="underwater.jpg"></a-sky>

<a-entity sound="src:ocean.mp3; autoplay:true; loop:true; volume:0.5"></a-entity>

<a-entity id="can" geometry="primitive:cylinder; radius:2; height:5" position="0 2 -10" rotation="-30 30 12" material="color:#9E9F9F; roughness:0.2; metalness: 0.7; src:url(label.jpg)">

<!-- Add in animations: a "float" animation, always on, and a "scale" element, triggered on click" -->

<a-animation id="float" attribute="position" from="0 2 -10" to="0 -2 -10" dur="10000"> </a-animation>

<a-animation attribute="scale" to="0 0 0" begin="resize" dur="10000"></a-animation>

</a-entity>

<a-entity light="type: directional; intensity: 0.3; color: #fff" position="-1 2 1"></a-entity>

<a-entity light="type: ambient; color: "></a-entity>

</a-scene>

<script>

// Create a variable to store the A-Frame container our particles will go in

var entityElement = document.getElementById("container")

document.getElementById("sceneElement").appendChild(entityElement)

// When we load the page, add the particles to the container to prep for our animations

function OnSuccessfulLoad() {

var i;

for (i = 1; i < 10; i++) {

var r, x, z;

for (r = -10; r < 10; r++) {

// Create our metal particles

var particleElement = document.createElement("a-entity")

particleElement.setAttribute("geometry", "primitive:sphere; radius:0.1")

particleElement.setAttribute("material", "color:#cccccc; roughness:0.2; metalness:0.7")

x = Math.PI \* r \* Math.cos(i)

z = Math.PI \* r \* Math.sin(i)

particleElement.setAttribute("position", x \* 1.5 + " 2 " + z \* 1.5)

entityElement.appendChild(particleElement)

}

}

setTimeout(function(){ShrinkCan()}, 5000)

}

// The first animation will shrink the can

function ShrinkCan() {

console.log("Called ShrinkCan() with success")

document.querySelector("#can").emit("resize")

EmitParticles()

}

// The second animation will create tiny particles around the camera

function EmitParticles() {

console.log("Called EmitParticles() with success")

entityElement.emit("breakApart")

setTimeout(function () { SinkParticles() }, 5000)

}

// The third animation will sink the tiny particles

function SinkParticles() {

entityElement.emit("sink");

}

</script>

</body>

</html>