Social VR for the Metaverse
With A-Frame WebXR -PART 1 of 2 PARTS

By Michael McAnally, © Copyright August 22, 2022

Rocket Virtual Blog  On Twitter  Funbit64 VRserver  On Medium

This cover is composed of actual screenshots in browser VR immersive spaces created during the multi-year development of this book.
# Table of Contents

Social VR for the Metaverse..............................................................................................................1
Book Credits.........................................................................................................................................3
Book Corrections.................................................................................................................................3
Introduction...........................................................................................................................................4
  What is WebVR?.................................................................................................................................4
  What is WebXR?.................................................................................................................................4
  Why do VR inside a browser?...........................................................................................................5
  What is A-Frame?..................................................................................................................................5
  How to begin?.......................................................................................................................................6
The Technology Stack Visualized......................................................................................................6
The Tick And Rendering Loop.............................................................................................................7
Note To Developers (Or The Browser Cache Can Be A Pain)..............................................................7
Future Tech..........................................................................................................................................8
The Book Approach............................................................................................................................9
  Glitch Link:........................................................................................................................................10
  GitHub Link:......................................................................................................................................10
  RocketVirtual.com Code Example Links:..........................................................................................10
Chapter 1 - Hello World VR!.............................................................................................................12
  Geometric Shapes, Position, Rotation And Color............................................................................13
  Using The A-Frame Inspector...........................................................................................................15
  Changing Colors..............................................................................................................................17
  Creating A Custom Sky....................................................................................................................19
  More Geometric Primitives.............................................................................................................21
  Versions, Documentation, Bug Issues And Inspect Console............................................................22
Chapter 2 – Animation, Lights And Shadows....................................................................................27
  Mixins And Animation......................................................................................................................27
  Lights And The Shadow Camera.....................................................................................................28
Chapter 3 - VR Environment, Movement And Selection....................................................................31
  Creating A VR Environment.............................................................................................................31
  Movement While Inside VR..............................................................................................................33
  Loading And Displaying 3D glTF Models.........................................................................................36
  Selection With Raycaster..................................................................................................................37
  Controlling Audio............................................................................................................................40
  Dramatic Effects (Moving The Camera)............................................................................................40
  Dramatic Effects (Toggling The Environment)..................................................................................41
Chapter 4 - Inside The Sphere............................................................................................................42
  Painting The Sphere..........................................................................................................................43
  Problems With This Photo.................................................................................................................44
  Teleporting Through Multiple 360° Scenes.......................................................................................46
  Solving The Large 360° Video File Problem.....................................................................................48
  Hosting 360° Video On YouTube......................................................................................................49
Book Credits

It takes a community, especially for all the things learned over the years about Virtual Reality inside of browsers over the internet, leading up to and during the writing of this work. I would be remiss if I didn’t mention the individuals who helped me out along the way, in one way or another. Whether it was through support on Discord, Slack, GitHub, Stack Overflow, Glitch, Twitter, Medium, YouTube, Open Source Code, Documentation, and examples, a video call, an in person meeting, or just simply an encouraging email.

Bravo to all, I am in your dept: Diego Marcos (Supermedium, and A-Frame, Support on Discord and emails), Vincent Fretin (Networked-Aframe), Ada Rose Cannon (W3C WebXR GitHub XRBoilerplate example, twitter txt, simple-navmesh), Kevin Ngo (A-Frame, GitHub, meetups 2016-2017), Don McCurdy (A-Frame, Navmesh, GitHub, email), and Kieran Farr (GitHub examples, discussion of book approach, and subject matter), Diarmid Mackenzie (GitHub examples), and Brandon Jones (Early WebVR examples, meetups 2016-2017). Also for your technical discussion, and greatly appreciated code testing assistance, experimentation, and support, my good friends Shawn Sweet, and Frank Weiss.

Thank you all very very much!

Book Corrections

Before I finally begin, it is too optimistic to think that I can write a perfect book the first time! I’m only human, and make mistakes just like everyone else. So with that in mind, some things can be corrected or improved after the book is completed, and published. I’m including a link to a corrections page below. I will endeavor to keep it updated.

No corrections yet. https://funbit64.com/drupal/index.php/node/10
Introduction

Although Virtual Reality (VR) as a technology has been around for some time, it is only recently that consumer hardware, more commonly VR headsets have become available at an affordable price. At the same time, open-source repositories providing for the easy coding of VR with HTML and JavaScript inside a browser are now available for free download. The browser is undeniably the killer app of the internet.

What is WebVR?

Historically there was a technology called WebVR, which is a JavaScript API for creating immersive 3D virtual reality experiences in your browser. WebVR has since been deprecated. Meaning it is now superseded by WebXR. This can be somewhat confusing because there are still many older examples of WebVR on the internet.

What is WebXR?

WebXR is the new W3C specification for browsers. In addition to Virtual Reality, it includes technology specs for Augmented Reality (AR). Hence the "X" in WebXR can be variably a "V" or an "A" representing VR or AR respectively (or sometimes MR referred to as Mixed Reality). XR is officially referred to as EXtended Reality. There is a spectrum of virtuality called the Virtuality Continuum in computer science. Use this link to determine which browsers support the WebXR API.

This book will focus on virtual reality, and will explore coding inside a browser, taking into consideration the basics, and some new developments. We will assume you know HTML and some lite JavaScript as a prerequisite. If you don’t, there are many free articles and videos to learn from on the internet. Just search for them, study up, and come back here.
Why do VR inside a browser?

Well many web developers already create websites for the internet, and know and use HTML and JavaScript daily. Also, most every device these days (smartphones, computers, tablets) comes with a browser installed, or it can usually be downloaded free. Plus the software tools for browser development and 3D modeling are diverse, powerful, and with many of them, again absolutely free (example: Blender).

Finally, given the ease with which you can deliver an immersive VR experience to someone, by simply sharing a URL address with a friend or social media; doing VR inside a browser just makes good sense. It’s also possible WebXR is the breakthrough spec upon which a large part of the future hypothetical “Metaverse” may be built.

What is A-Frame?

First, recent versions of A-Frame are WebXR compatible. A-Frame is an open-source web framework for building virtual and augmented reality experiences. It is a framework based off an entity component design pattern which leverages the power of THREE.js (sometimes capitalized in code or literature) for the easy creation of 3D immersive experiences, simulations, scenes, and games.
How to begin?

If you are just starting out with A-Frame, you should probably first learn the basics for creating primitive objects, positioning them, their rotation, visibility / transparency, controlling lights and shadows, simple animation, manipulating your environment, loading assets and 3D glTF models, controlling audio, and implementing selection / movement while immersed inside your VR app. Be assured, we will go through all of this to some detail inside of this book.

The Technology Stack Visualized

*For those interested in a high level view of the overall technology stack used, the figure above can be helpful.*
Since A-Frame wraps an instance of THREE.js and follows the WebXR specification, it allows for VR and AR inside a headset device browser (as an example: the Meta Quest 2 Browser, which I understand is based off of Chromium Blink).

VR/AR can be accessed from a variety of vendor headset hardware devices, browsers, and shared across the internet by a simple URL address. The only downside being the time it takes to download heavy graphic assets into a browser (but remember when you install a game or other app the first time, you are incurring an install download time, we just tend to forget about it). However after the browser has cached the assets, they will usually load much more quickly the next time the page is loaded.

Some of this will be mitigated in the future, Internet Access speeds are always increasing (for example 5G), as well as more powerful mobile graphic processors, and with enhanced resolutions and edge computing AI capabilities. A tether-less VR headset has basically all the hardware components of a smartphone, plus optics, controllers, all in a head mounting case.

**The Tick And Rendering Loop**

There is such a thing called a tick, or a frame of the scene’s render loop. Meaning the scene is drawn before it’s pushed into the VR headset on each frame (or tick) of a rendering loop. This usually happens many many times per second. It gives an appearance of smooth animation of objects, while not causing nausea to the VR headset wearer. This rendering loop is found inside of all game engines. You can find out more about A-Frame tick and it’s rendering loop here.

**Note To Developers (Or The Browser Cache Can Be A Pain)**

Caching and sessions can be complex to understand. I recommend changing the name of your HTML filename while testing and debugging changes in the code. This will usually resolve any confusion as to whether you’re testing an older cached version or the actual code change you just made. This won’t be a problem with our Glitch examples (discussed further on). This advice is mostly for when you are using your own webserver development environment. I self host, usually on a VPS, but there are many cloud options.

Ultimately, if you are unsure, try it in a different browser, such as FireFox, or Edge that you haven’t used with that filename before. Clear all the cache, and when you are done debugging, change the filename back to the desired filename. It works for me.

Finally, I might be stepping out on a shaky computer science limb here by saying, A-Frame programming of WebXR with its entity component system, and attributes, might be considered Attribute programming, or Declarative programming, but when you add JavaScript it becomes Imperative programming. So, I don’t really know, call it what you want, it’s programming VR in a browser to me!
Future Tech

Given some time (probably in the mid-to-late 2020's) I truly believe VR headsets and AR ones will merge and become as indispensable as the mouse is for 2D screens. No more sticking smartphones into cardboard, that time will have long come and gone. Also, think no need for controllers in the future (hand tracking), just VR hand gestures for manipulating, and interacting, as well as emerging eye tracking hardware.

So the future is bright. Let's start learning and building it today!
The Book Approach

The easiest way that I know how to teach VR in the browser is with lots of example code, slowly starting from the basics to increasingly more complex. That’s how the next chapters will be taught. We will start with the original A-Frame Hello World and progress to our more advanced example Hello World VR.

Along the way, I’ll refer to each example and point out specific details along the way. You will be allowed to modify the examples yourself and experiment with things as we go. In fact, I encourage it!

The best way to learn is by trying, and the best teacher is making mistakes along the way, and then learning what is necessary to correct them. Of course avoiding them is most desirable, but not always possible. We live in an imperfect world, and I wouldn’t have it any other way. Which in actuality makes it perfect. Pardon, that I got a little philosophical there.
How the last example code in Chapter 3 will look when loaded into Glitch with preview on right.

The documented code will be presented in Glitch which allows for easy remixing, and modification. Refer to Glitch documentation on how to use. I also provide the code on GitHub. With links to the working examples.

I have made every effort to make sure the source code and its assets are original or open source, with no copyright infringement. If this is found to not be the case, I sincerely apologize, because I wish to respect the rights of all creators and their original work, as you should always do.

Glitch Link:
https://glitch.com/edit/#!/handy-enormous-nightshade

GitHub Link:
https://github.com/Mike-McAnally/VRBook

RocketVirtual.com Code Example Links:
https://rocketvirtual.com/HelloWorld.html
https://rocketvirtual.com/HelloWorld2.html
https://rocketvirtual.com/HelloWorld3.html
https://rocketvirtual.com/HelloWorld4.html
https://rocketvirtual.com/HelloWorld5.html
https://rocketvirtual.com/HelloWorld7.html
https://rocketvirtual.com/HelloWorld8.html
Chapter 1 - Hello World VR!

Let’s start with the basics for A-frame Hello World.

```html
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Hello World VR</title>
  <meta name="description" content="First example, basic hello world for VR using A-frame WebVR.">
  <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0, shrink-to-fit=no">
  <meta name="mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-capable" content="yes"/>
  <meta name="apple-mobile-web-app-status-bar-style" content="gray-translucent"/>

  <!-- A-frame minimized JavaScript version component entity system -->
  <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
</head>

<body>
  <div id="scene" style="background-color: #444444; width: 100%; height: 100%; z-index: 0;">
    <!-- Used to create a scene, like on the stage of a play, but in 360 ° -->
    <a-scene>
      <!-- 3D primitive objects inside our scene, positioned x, y, z and rotation, and color -->
      <!-- from our default created camera at view point 0, 1.6, 0 which is 1.6 meters above the ground -->
      <a-box position="-1.0 0.0 -3" rotation="0 45 0" color="#44C3D8"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#FF206E"></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>

      <!-- The plane upon which the objects are virtually positioned -->
      <a-plane position="0 0 -4" rotation="0 0 0" width="4" height="4" color="#7DCEAA"></a-plane>

      <!-- We are inside a gigantic hollow sphere, looking at its inner surface painted a light color of gray -->
      <a-sky color="#E8ECEC"></a-sky>
    </a-scene>
  </div>
</body>
</html>
```
Geometric Shapes, Position, Rotation And Color

Look at the source code HelloWorld.html and its browser rendition (HTML displayed in the browser). It’s HTML with comments, things are indented with extra lines for readability. Now notice <a-scene>, line 18, its entities, <a-box>, <a-sphere>, <a-cylinder>, lines 22-24, <a-plane>, line 27, <a-sky>, line 30, and a closing </a-scene>, line 31. Also notice <a-box> has a closing </a-box>, as do the other shapes.

The attributes within <a-box> are position, rotation and color.

The position of the box is defined as x, y, z coordinates separated by a space between quotes. Position is defined as in a Cartesian Coordinate System. Where the z-axis is a positive value coming outward from a 2D screen and negative value going backward into the screen, if you imagine we are standing at the origin of 0, 0, 0.

With A-Frame by default we are actually looking at things with our virtual eyes from a height of standing 1.6 meters tall. So that would mean our view point is really at position 0, 1.6, 0 for our x, y and z axis values in this first scene. <a-scene> being the virtually defined area created by A-Frame, sort of like a 3D 360 degree play stage.

Because our <a-box> has position=-1 0.5 -3 means it appears -1 to the left, raised 0.5 up and -3 back into the screen, if that makes sense. The same goes for the other geometry shapes in our scene which all have their own positions defined.

With our <a-box> rotated at 45 degrees counter clockwise on the y-axis, giving it an angle on our 2D screen, and the appearance of depth.
If you drag with a mouse or touch pad in the center of the screen you can rotate our view of things much like turning our virtual head.

If you press the W A S and D keys on the keyboard, you can move (step) forward, left, right and back inside the scene.

Amazing, right?
Using The A-Frame Inspector

A-Frame provides for a way to change the attribute values in a scene dynamically inside the inspector. You’ll then have to copy those changed values back into the source code and save them, for them to take effect.

To enter the inspector, press the Ctrl Alt and i keys simultaneously ( <Ctrl> + <Alt> + i ).

In the left panel of the inspector you will find your <a-scene> and all objects including <a-box>. Select <a-box> and notice a right panel appears with the attributes for the box, along with positional arrows, red, green, blue. These allow you to move the position of the box while affecting the position values in the right panel. Try it.

Notice the controls on the top, just slightly left of the right panel. Select the rotation circle control, like this.
You will now notice a change to the <a-box>.

Now select holding down while dragging on the green circular line. Notice when dragging left or right the box (or cube) rotates. Also the rotational values for y in the right panel update as well correspondingly.
Changing Colors

Now scroll down on the right panel to the section under MATERIAL called color. Select the blue rectangle. A color picker will appear allowing you to change the color of the cube. It also displays a hex value #4CC3D9 next to the colored rectangle. This is the value of the color attribute on <a-box> in the source code. By changing that value in the source with an editor of your choice and saving it away, you will change the color as well when the HTML loads in the browser, the next time you refresh the page. In Glitch things are saved for you when you remix and modify the code.
A multitude of color to choose from

So let’s do that now as an exercise. Change the color of each of the geometric shapes, `<a-box>`, `<a-cylinder>`, `<a-sphere>` and save away our source code and refresh the browser.

There are a number of color picking sights on the internet which you can use to generate attractive color palettes to use in your A-Frame scenes. Copying the hex values into your source.

Color Generator link
https://coolors.co

The results will be something like this, depending on the colors you choose.
Creating A Custom Sky

Now we are ready to move on to our next exercise.

1. Take a look at the new source code `HelloWorld2.html` and `browser rendition`. 
Notice the <a-assets> tag contained within the <a-scene> tag. We have an img tag on line 21 which is given a source (src) of landsky.jpg stored in an assets/img/ directory. We have assigned an id equal to the name or label of texture1.

Much further down on line 34, we have the <a-sky> tag which we have given a material of source #texture1. This assigns the landsky.jpg file as a texture material for the sky. So imagine the sky as a gigantic sphere of which we are inside of, and the inside the center of the sphere is painted with the texture of landsky.jpg. That is essentially what is happening here.

The actual landsky.jpg file looks like the image above and the <sky> tag wraps it around us.
You can see this if you enter the A-Frame inspector again ( <Ctrl> + <Alt> + i ) then select <a-sky> in the left panel. Now scroll back away from the screen, keep scrolling a lot, until a sphere will appear with the landsky.jpg texture mapped onto it. You should imagine yourself as inside this painted sphere when you are in true VR. Now scroll back into the normal size view inside the inspector.

**More Geometric Primitives**

We are going to modify the code and add more geometric primitives, maybe even some you have never heard of, I hadn’t.

```html
<a-entity geometry="primitive: dodecahedron; radius: 0.75" position="1.8 2.2 -4.6" rotation="0 45 0" material="color: #F3FFB0"></a-entity>
<a-entity geometry="primitive: box" position="-1 0.5 3" rotation="0 45 0" material="color: #C814A"></a-entity>
<a-entity geometry="primitive: sphere" position="0 1.25 -3" radius="1.25" material="color: #E84260"></a-entity>
<a-entity geometry="primitive: cylinder; height: 1.5; radius: 0.5" position="1 0.75 -3" material="color: #B1E83C"></a-entity>
```

First let’s modify our box, sphere and cylinder by using the <a-entity> tag with the geometry attribute. This is a more general way to create geometric shapes. Notice we are using primitive: box within the geometry attribute. Now let’s add a new geometric primitive, dodecahedron (with 12 faces or sides) to the scene, giving it a radius, position and rotation.
Our modified scene should look something like this. In Glitch this is the HelloWorld4.html.

**Versions, Documentation, Bug Issues And Inspect Console**

Up until this point we haven’t talked about the A-Frame JavaScript framework which we included near the top of the HTML.

```html
<!-- A-frame minimized JavaScript version component entity system -->
<script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
```

First notice the version number 1.3.0 which is important because A-Frame tends to change rapidly and sometimes things can get broken or become buggy along the way. Meaning A-Frame is under rapid development, and because of that some things don’t work in newer versions of A-Frame. This can even be sometimes because of changes to the browser itself. Not to worry, A-Frame seems to be increasing in code stability as version numbers increase, as this is a natural result of any software development.
Issues with latest A-Frame master and releases can be found under the issues tab on GitHub.

A-Frame can be found on GitHub here.

The website for A-Frame can be found here.

The detailed documentation for A-Frame can be found here.

Something else which will be useful for you is the ability to inspect execution of your code while in the browser. You probably won’t have to go into the length of how to debug JavaScript in our exercises, just know how to access the console first, which can be really useful. I use a right click in the center of the screen, select the inspect option, then select the console tab.
Now with all that said, we are going to add many more geometric objects, but first we are going to add a new A-Frame compatible JavaScript component to our HTML to help us with laying out the positions of all those new objects we will create.
The **aframe-layout-component** is very useful and can save a lot of time and effort.

We added it right after A-Frame itself in the code above. See [HelloWorld6.html](#).

We are now going to layout our new geometric shapes in a circle around the existing shapes of box, cylinder and sphere. We do that with an `<a-entity>` and layout attribute with a radius on the xz plane. We position the circle and include all our new shapes inside the closing </a-entity> block, which affects all the shapes within, assigning each of them relative positions equidistant from each other. So we don’t have to actually assign a position attribute to each shape, that’s great because layout nicely takes care of that for us.

We are also going to add one more “sub-attribute” to material called opacity. Opacity allows us to make things partially transparent like glass. A value of 1 is no transparency, which is the default. A value of .5 is half transparent, and so forth fractionally.
Our scene should now look like this.
Mixins And Animation

Now we are going to add something called Mixin. It is tagged as <a-mixin> and goes in the <a-assets> block. It’s sort of a shorthand variable attribute for use throughout the <a-scene> section.

Inside our mixin we are going to do some animation for the first time. Animation is done in A-Frame with the animation attribute, and in this case can do a looping rotation on objects that are assigned the mixin.

Here is the code in Glitch for HelloWorld7.html and the browser rendition of that code.
We added three mixins with the ids of spin-y, spin-x, spin-z. Each one of them spins around a different axis in the Cartesian Coordinate System. You can see that in the parameters within the animation attribute rotation; to: 0 360 0; loop: true means spin continuously around the y-axis for example. The others have 360 on each other axis.

The mixin ids are then used with the mixin= attribute and assigned to each object entity. For example the cone is assigned the spin-x mixin, which will make the cone spin around the x-axis.

Mixins are very powerful short cuts in A-Frame and used to shorten effort, and the readability of code.

Lights And The Shadow Camera

Let’s first extend our plane a little larger so we will be able to see all the shadows the light will cast by extending its width and height to the value of 12. Then we are going to create something that is a light with a shadow camera.

Inside our new entity we create a light with castShadow: true. That light also has attributes for a shadow camera. There are number of parameters for the shadow camera which control the size of where the shadows will fall when the light casts upon them.
The shadow camera light is a fairly complex entity and we give an example of it here, however we are not going to go into too much detail. You can find more in the A-Frame documentation. One thing to point out is that the shadow camera must be adjusted correctly for the shadows to appear correct.

With our light and shadow camera added our scene object shadows have darkened a little bit. To fix that we will add an ambient light to brighten things up, setting its intensity to 0.7.

```html
<body>
  <!-- Used to create a scene, like on the stage of a play, but in 360 ° -->
  <a-scene shadow="type: pcfsoft" renderer="antialias: true; highRefreshRate: true;" shadow="autoUpdate: false">
  <a-assets>
  </a-assets>
</a-scene>
</body>
```

Also we are going to add additional shadow and render control attributes to our `<a-scene>`. This will soften the shadows, reduce their graphic processing load on the GPU and improve aliasing and increase refresh rate of the VR display.

In Glitch [HelloWorld9.html](#).
This is how it should look with the new objects animating around the original box, cylinder and sphere. We changed the opacity back to 1, or you can remove it entirely, since 1 is the default. We can even animate the opacity, but that is outside the scope of this tutorial.

```html
<entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" mixin="spin-y">...
</entity>
```

Now for fun, and a more dramatic effect add the spin-y mixin to the layout circle. It causes the whole layout of the objects in the circle to spin around. That is powerful animation!

```html
<entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" mixin="spin-y" animation="property: position; from: 0 1.6 -5; to: 0 1.6 -5; dir: alternate; loop: true">...
</entity>
```

For a final animation example we are going to add the animation attribute directly to the layout. This will move the spinning layout circle up and down and reverse its direction.

Finally, in Glitch HelloWorld11.html and rendered in the browser like this.
Chapter 3 - VR Environment, Movement And Selection

In this chapter we are going to be focusing mostly on things inside of VR.

Creating A VR Environment

Our original landsky environment was simple and effective. However, persons inside of VR are known to explore more extensively, much as they do in the real world, and usually require a richer, more immersive environment. To accomplish this with the least amount of effort, we will introduce and use the `aframe-environment-component`.

First we add our new component in the `<HEAD>` of the HTML just following the aframe-layout-component we added earlier.

Then near the bottom of the code just above `</a-scene>` we add the entity with attributes for our new environment calling it parkland. Notice we also commented out the ambient light above, the plane and removed the sky, and its texture from assets. This is because the environment controls some of the lighting and sky itself, all automatically for us. As well as creates a ground for us and adds some 3D models into the scene to make it all that more believable. Admittedly we don’t get the clouds in the sky, but that’s a small price to pay for what we do get in return.
So Let’s see. In Glitch HelloEnv.html and in the browser.

If you examine this all in the A-Frame Inspector ( <Ctrl> + <Alt> + i ) you can see these elements, even play with modifying them.

Select parkland in the left panel, and if you look closely at the right panel you will see attributes for environment. Also you will see other entities within the parkland block, such as lights, ground, dressings, and sky. Explore further.
Any changes you make will need to be copied back to the source for them to become permanent. You can copy those attributes back by selecting one of the copy symbols in the right panel, then pasting the text into the source.

Let’s make another environment, an Arizona desert like one by selecting the yavapai preset and adding that to our code. We will also comment out the other parkland preset temporarily.

The result will look like this. [In Glitch HelloEnv2.html](#) and [in the browser](#).

**Movement While Inside VR**

We will be adding a new A-Frame compatible component called [aframe-extras](#) for moving around inside of the VR environment while immersed with our headset on.

*NOTE: At the time of this writing aframe-extras has picked up a few minor bugs, but they won’t affect us directly here in our code examples!* So we will still use the powerful moment-control addon provided. For your information, the bugs are related to the ocean and navmesh. To get these to work you may need to use earlier versions of A-Frame, such as 1.1.0. Which is really fine if you don’t need any newer features of A-Frame added since, and if you can reconcile any version dependencies between components. **RECENT UPDATE:** In a later chapter I found a fix for the ocean [starting line](#).
103 in the JavaScript and line 292 in HTML) and later again a substitution for navmesh called simple-navmesh in this example on GitHub. Just giving you a teaser glimpse into future chapters!

In Glitch HelloMov.html and in the browser. Explaining the code, you will see the new aframe-extras.min.js has been added to the top, right after the other A-Frame components we added earlier.

Further down, you will see an <entity> with an id of cameraRig with movement-controls. It is the movement-controls which allow us to move around inside of VR with usually a toggle stick on our controllers. Within that entity you will find another which is actually the camera. We have given it an id of head instead, because it represents where our eyes and head would be from the cameras perspective.
Inside of the camera head entity you will find two other entities which represent the left and right hands of our virtualized hands (or partial avatar) inside of VR. These hands are presented in VR by left and right low poly 3D models that are colored the hex color of blue.

Our virtual hands as seen inside of a VR headset.

Here is what they should look like inside your VR headset.
Loading And Displaying 3D glTF Models

Next we are going to load a couple of 3D models and put them in the environment for both the desertland and parkland environments. For the desertland we are going to add some cactus, and for the parkland we are going to add some trees.

The models will be in a format called glTF, which is sort of a jpeg compression format, but instead for 3D models. We will load those models first inside our <a-assets> tag and then display them with an entity later in the HTML code. In Glitch HelloMove5.html and in the browser.

```html
<!-- Cactus models 3D glTF models -->
<a-asset-item id="cactusShort" src="assets/gltf/cactusShort.glb"></a-asset-item>
<a-asset-item id="cactusTall" src="assets/gltf/cactusTall.glb"></a-asset-item>
</a-assets>
```

Notice inside the Cactus1 id we are using gltf-model to display the asset we loaded earlier #cactusShort. We give it a position and a scale which is used to transform its size and shape in the three axes. We can make sure it doesn’t receive any shadows to simplify any rendering load on the GPU, since it is not needed in this case.

```html
<!-- Tree models 3D glTF models -->
<a-asset-item id="Tree1" src="assets/gltf/NormalTree_1.glb"></a-asset-item>
<a-asset-item id="Tree2" src="assets/gltf/NormalTree_2.glb"></a-asset-item>
</a-assets>
```

<!-- Place a couple of simple tree models to enhance the scene -->
<entity id="ParkTree1" gltf-model="#Tree1" position="-9.59927 -0.29994 -9.83981" scale=".81 .01 .81" shadow="receive: false"></entity>
<entity id="ParkTree2" gltf-model="#Tree2" position="4.35166 -9.11864 -5.78572" rotation="0 -73.954 0" scale=".81 .01 .81" shadow="receive: false"></entity>
```
We now comment out the desertland environment and do the same again, but adding trees for parkland instead.

After adjusting some colors in the parkland scene to make the greens more vibrant, we get this.

**Selection With Raycaster**

Now that we have two scenes set up, we might want to do something that will allow us in VR to have an effect upon them. This will be a bit more programmatically complex, and involves some simple
JavaScript. This will result in making the whole scene more functionally interactive. For this example we will use Glitch HelloSel3.html and in the browser.

```html
<a-entity id="a-scene" class="scene" rotation="0 45 0" scale="5 .5 .5" material="#7047A7; quality: 1; display:spin-y shadow="receive: false" onclick="disappear(object)" />
```

Remember we talked about the hands that you see while inside your VR headset? Well one of those hands, the right one in this case, will have a pointer laser coming out of it associated with a Raycaster.

On the `<a-scene>` tag you can see the raycaster defined in a more fine grain way. It basically shows the ids of the objects the raycaster can be intersected and selected with a period in front of them. It also references a class called clickable.

You can see that our `<a-entity>` objects have class clickable and an onclick function call to JavaScript function while passing in a label variable for the object name.

Let’s take the example of the tetrahedron.

```html
<a-entity id="cone" class="clickable" geometry="primitive: cone" rotation="0 45 0" scale="5 .5 .5" material="#7047A7; quality: 1; display:spin-y shadow="receive: false" onclick="disappear(object)"
```

When the raycaster intersects the tetrahedron, while inside the VR headset, and the trigger button is pulled a selection occurs. This calls the disappear function with the object name.
Our JavaScript is added. The `disappear(objectName)` JavaScript function uses the **DOM** `document.getElementById` to find the line of HTML, and then set an attribute called `visible` to the value of false. The attribute `visible` is not shown in the line, but it's implied and the default is true, meaning the object can be seen. By setting it to false we actually make its visibility disappear. The object is still there, but it has super powers, and thus is invisible!
We also added something called sparkle.play();. This plays a disappear sound we loaded globally at the top as a new audio variable called sparkle.

For making all the objects visible again, whether any of them were previously selected or not, we use an onclick function called reappear(); attaching it to our cylinder. We then play the same sparkle sound for our reappearing objects.

You may have noticed a couple of AFRAME.registerComponent in the JavaScript. A-Frame has a way of creating custom components. There are advantages to doing this, mostly performance, and more direct access to the control of THREE.js. You can find out more about this in the A-Frame documentation, where it is explained in detail with examples.

**Controlling Audio**

We are going to add a texture to the <a-box> or cube, and use it to initiate playing music.

```html
<!--[-- Music box texture applied to cube -->
  <img crossorigin="anonymous" id="musicbox" src="assets/img/music.png">


<entity id="cube" class="clickable" geometry="primitive: box" position="-1 0.5 3" rotation="0 45 0" materials="src: #musicbox" shadow="receive: true" audiohandler="/a-entity">

We use the material with src as we did with the landsky.jpg texture in Chapter 1.

Also notice we added something called audiohandler which is an AFRAME.registerComponent which handles selecting, playing, and stopping our music. Our music src to play is defined inside our <audio> tag near the top <body> tag in the HTML. It is set to autoplay and loop. Unfortunately in our case browsers require the user click to enable interaction, then at least once more to initiate audio. So sometimes it may take two clicks to play the music the first time, and then one each thereafter to stop, and start playing again.

**Dramatic Effects (Moving The Camera)**

Now for fun, we are going to add a dramatic effect by including an animation on the camera (or our head id label) which starts our page far out and then moves it inward towards our objects. In Glitch HelloVRFinal3.html and in browser, which is the final example for this chapter.

As shown the animation starts at from: position 0, 10, 50 and moves in towards to: position 0, 1.6, 0 the default starting position in A-Frame.
Dramatic Effects (Toggling The Environment)

Another dramatic effect can be achieved by toggling between the desertland and parkland environments. The code above gives an example of this, and the ChgEnv() JavaScript function is activated by an onclick when the sphere is selected. Toggling the active and visible attributes for the environment as well as the objects we put in them.
Chapter 4 - Inside The Sphere

Up to this point our VR experiences have been a little cartoonish. Just meaning they don’t seem as real as the actual world outside our headsets. We are going to change that in this chapter.

Using 360° Cameras is a way to capture the real world and experience it in VR immersively. We’re going to use images from my 360° Camera to do this in our examples.

We are also going to use the same trick we used in Chapter 1, when we painted the inside of a sphere with the landsky.jpg file. However, instead we are going to use images captured and processed from a 360° Camera. The difference in these images from regular photos is that they are equirectangular images. Imaging peeling the skin off an orange and then pressing it down flat on a surface. If it didn’t tear but stretched instead, it might look very strange, much like an equirectangular image does.

Finally, we are going to allow you to teleport in a sense by flipping quickly through multiple images inside VR. We are even going to use a 360° still video of images, and also fly.

Yes, fly like a superhero! My friend and I attached my 360° Camera to his drone and flew it around outside his house as an experiment. I’ll share that immersive experience with you. So let’s get started, we’ve a lot to cover in this chapter.
Painting The Sphere

In our first piece of code we are starting out simple again. As you see from line 23, we are loading an img asset and giving it the id orb1. That image is a 360° equirectangular image taken with my camera on the rocky beach with a tripod. You can see San Francisco’s Golden Gate Bridge in the distance.

This image is painted on the inside of the sky sphere on line 28. That’s it, simple and effective as an immersive experience inside a VR headset. *Try it out with a VR headset.* A more realistic experience than our previous examples.

Chapter 4 can be found on GitHub here.

Problems With This Photo

Now I’d like to point out a few problems with this photo for learning purposes. First its a still single photo, so there is no moment of the ocean waves. It has no seashore sound, and if you look downward directly toward where your feet would be, you will see part of the tripod where the camera joined the image, and the shadow of the tripod itself.

Also in the distance offshore there is a rock formation which wasn’t joined correctly at one of the edges as the image is wrapped around. Finally, I’m in the image. That’s not a problem, but maybe you want to be alone on the beach to relax, without looking at me in all my photogenic stillness.

You can edit your photos with a photo editor, such as Photoshop or GIMP, but you may find that hard because of the equirectangular aspect of the image. No worries if you are not skilled in photo editing, but I’m going to try anyway. All assets are provided for you in the examples.

I was able to remove myself, and the tripod and its shadow from the image. However, photo editing an equirectangular image with complex natural graphics, in this case sand and rocks is difficult. You’ll want to edit your 360° photos as little as possible, or not at all.

So now let’s add the ocean wave sounds and fix the final joining at our feet where the tripod use to be with a sand textured circular plane. It is not perfect, but much better over all.


Now let’s take a look at the code. The code we added should be recognizable from our earlier chapters. The cameraRig on line 73, with left and right hands on lines 77, and 79. The ocean audio on line 55. With JavaScript code attached to our playOcean button on line 84, with audiohandler.

The difference being our movement-controls don’t seem to let us move around. That’s because we haven’t included aframe-extras like we did in previous examples. Our position is fixed from where the picture was originally taken. Sure we can look around by turning around, or maybe swivel around in an office chair.

What if we could teleport to another VR scene, to many scenes in the same web page.
Teleporting Through Multiple 360° Scenes

This immersive experience in a VR headset should be near realistic, with multiple VR scenes.

The next example code is very large, 485 lines. The example(s) also uses large graphic files, and still images in a streaming video. I have manipulated many of the graphic assets in Photoshop, and in a Video Editor for 360° images myself.

All this will allow us to teleport to locations I have photographed in 360. I have not blurred out any of the faces, as I have asked and acquired the rights to the images, all others have been photographed in public accessible spaces. I will be in many of them, and I give myself the rights to myself!

With all the graphic assets used, this page may take up to 30 seconds or more to load, please be patient.

On GitHub here.


In Glitch 360VRteleport.html and in browser here.

In order to keep this book reasonably succinct, and to not bore you, I’m going to refer to articles I have already written on the internet explaining much of the code provided here with a few small changes to line numbers, and such.

https://michael-mcanally.medium.com/my-360-mind-place-web-app-for-capturing-immersive-vr-moments-free-open-source-code-295e4f9fbdf0

As promised the next example uses a 360° drone video I recorded while a friend is flying over his house. You may experience dizziness, due to the fact that your perspective is flying, and shaky, because of the wind on that day, and the heavier weight on the drone from the attached camera. Also, I didn’t edit out the drone noises, which makes it sound like a bunch of angry bees!

Besides the drone above you and its shadow on the ground, you may notice some choppy delays in the streaming of the video from my server. The reason for this is the video is true 360° footage, and a very large file is coming across the internet. If your download speed, or “number of hops” from the server is many, or even if your internet access speed is constrained from many others using your cable internet at busy times of the day, you will experience stops and starts in the video.

360° video is inherently much larger than regular 2D video. The reason for this is because it includes much more data for views all around you. Imagine a cube face, one side is 2D video, height and width, whatever the resolution. Now imagine that all the 6 sides of the cube, while not an exact and perfect analogy, you see there is much more data to download. Of course there are compression algorithms involved, but suffice it to say, there is a lot of data streaming over the internet. It still all has to be there if you decide to rotate your head around inside the VR headset.
Solving The Large 360° Video File Problem


One solution to the large file download problem is not to use 360° video at all, but a limited stream of 360° still images instead. This is demonstrated in a previous example, and again here with many more still images. Yes, it’s not a perfect solution to the choppy start and stop video, but it’s one compromised solution.

A better solution would be to move the data from the server closer to the VR headset by placing it on say a LAN (local area network) or home server. Then the limitation is usually just Wi-Fi access speeds, but if you have a tethered headset with a graphic card, inside the server itself, serving up the graphic video files as local host, that would be the fastest yet. However, that is not a normal standard setup.

Perhaps in the future there will be ways that a server can only stream the image data that is necessary to the VR headset and respond dynamically with two way communication over the internet when the user begins to rotate their head, and then provide extra data at that time only. However, there still might be some latency issues in response time over greater internet distances.

The final way would be to package everything up into an app and “pre-download” it at install time.
Hosting 360° Video On YouTube

Another way to host and view 360° video is by uploading it (in equirectangular format) to YouTube, then viewing that with a link in the VR headset. The best way to input that link and select it is to already have it typed into a previously created web page with a shorter URL, and bookmark that page in your VR browser.

Here is one I recorded in a 360° video and uploaded to YouTube of me playing a pinball machine at my favorite old time pinball arcade. *I sometimes get a little excited with the game!*

[https://www.youtube.com/watch?v=rs56uxdhnQI](https://www.youtube.com/watch?v=rs56uxdhnQI)

Finally, here is a menu page for additional 360 examples:


Our next chapter 5 will be about Social VR - Meaning multi-player type immersive communication experiences in VR with representative Avatars, and with audio talking, etc.