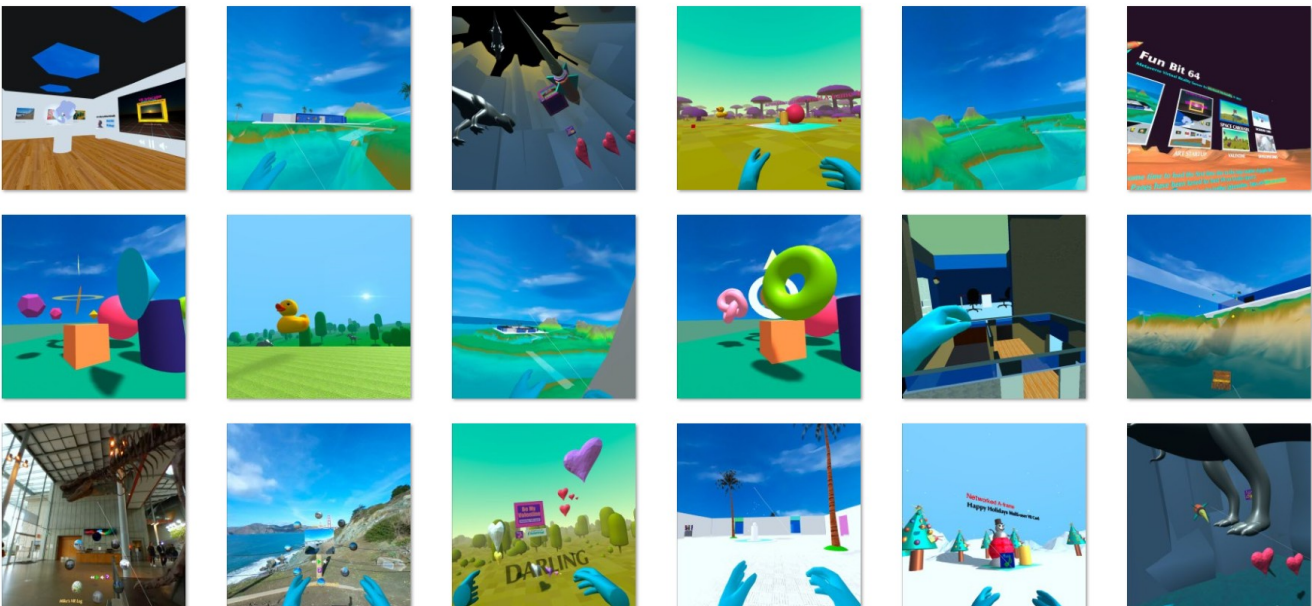


Social VR for the Metaverse with A-Frame WebXR

By [Michael McAnally](#), © Copyright September 14, 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 with A-Frame WebXR.....	1
Book Credits.....	4
Book Corrections.....	4
Introduction.....	5
What is WebVR?.....	5
What is WebXR?.....	5
Why do VR inside a browser?.....	6
What is A-Frame?.....	6
How to begin?.....	7
The Technology Stack Visualized.....	7
The Tick And Rendering Loop.....	8
Note To Developers (Or The Browser Cache Can Be A Pain).....	8
Future Tech.....	9
The Book Approach.....	10
Glitch Link:.....	11
GitHub Link:.....	11
RocketVirtual.com Code Example Links:.....	11
Chapter 1 - Hello World VR!.....	13
Geometric Shapes, Position, Rotation And Color.....	14
Using The A-Frame Inspector.....	16
Changing Colors.....	18
Creating A Custom Sky.....	20
More Geometric Primitives.....	22
Versions, Documentation, Bug Issues And Inspect Console.....	23
Chapter 2 – Animation, Lights And Shadows.....	28
Mixins And Animation.....	28
Lights And The Shadow Camera.....	29
Chapter 3 - VR Environment, Movement And Selection.....	32
Creating A VR Environment.....	32
Movement While Inside VR.....	34
Loading And Displaying 3D glTF Models.....	37
Selection With Raycaster.....	38
Controlling Audio.....	41
Dramatic Effects (Moving The Camera).....	41
Dramatic Effects (Toggling The Environment).....	42
Chapter 4 - Inside The Sphere.....	43
Painting The Sphere.....	44
Problems With This Photo.....	45
Teleporting Through Multiple 360° Scenes.....	47
Solving The Large 360° Video File Problem.....	49
Hosting 360° Video On YouTube.....	50
Chapter 5 - Social VR.....	51
Networked A-Frame (NAF).....	55
Your Own VR Server, And Why?.....	56
If You Want To Setup Your Own Server, What Does That Entail Technically? What Is The Cost?...	56

WebRTC?.....	58
Adapters For Networked-Aframe.....	58
My Disclaimer:.....	61
Some Assumptions.....	61
Purchasing And Setting Up A Domain Name.....	69
Configuring Apache With Your Domain Name And Virtual Hosts.....	70
Setting Up Networked A-frame (NAF).....	75
Testing NAF.....	80
SFTP.....	83
Chapter 6 – Some Social VR Spaces.....	84
Hexoplex Garden.....	88
Avatars.....	88
Art Room Gallery.....	98
Sci-Fi Themed Island.....	99
My VR History.....	99
Other Avatar Systems.....	100
Personalized Avatars.....	101
Video Avatars.....	102
Gifting 3D Printable Assets.....	104
Chapter 7 – Bringing It All Together.....	105
Treasure Island Art Gallery.....	108
3 rd Party Examples.....	109
VR Launch Pages.....	110
Inside VR Screen Shots Of Final Examples.....	111
Addendum (optional).....	121
Installing Drupal 9.....	121

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.

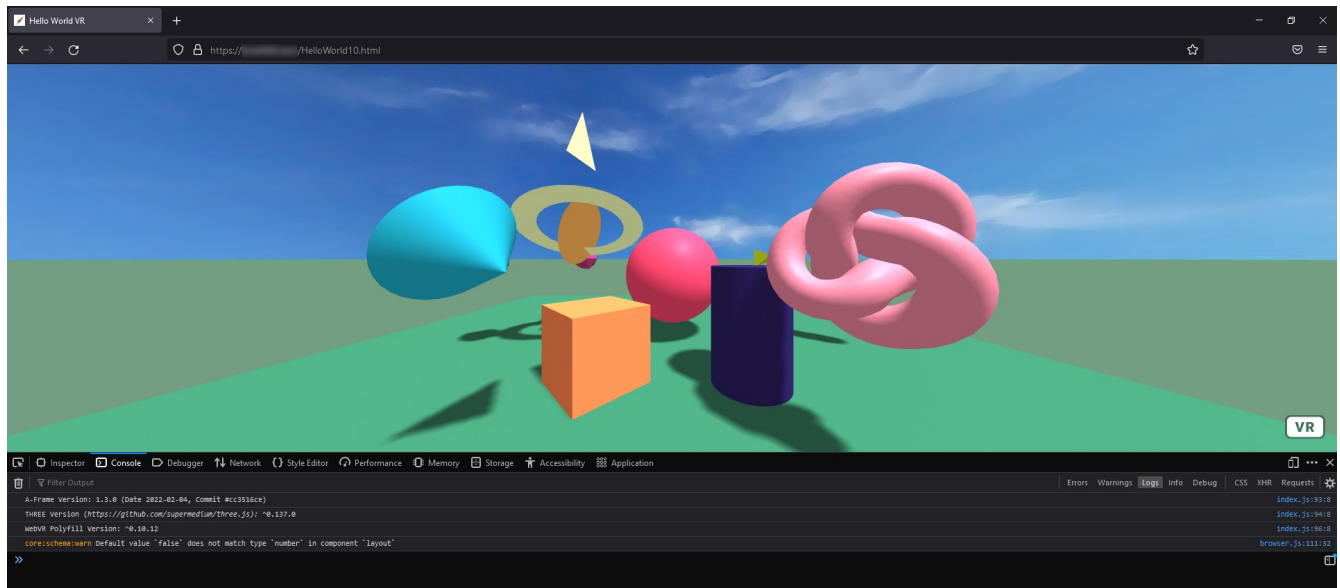
Thank you all very very much!

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



A-Frame HTML open in browser inspect console tab in FireFox.

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?

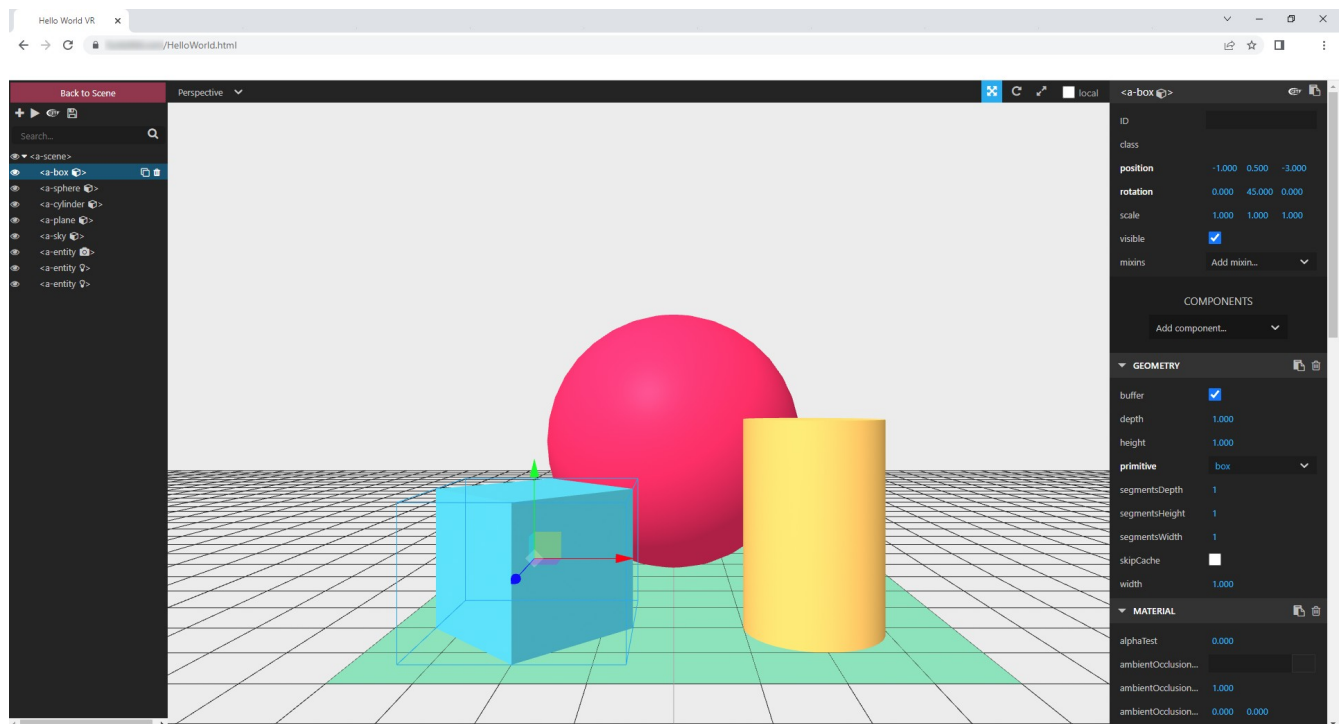
[WebXR](#) is the new [W3C specification for browsers](#). In addition to Virtual Reality, it includes technology specs for [Augmented Reality](#) (AR). Hence the "X" in WebXR can be variably a "V" or an "A" representing VR or AR respectively (or sometimes MR referred to as [Mixed Reality](#)). XR is 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.



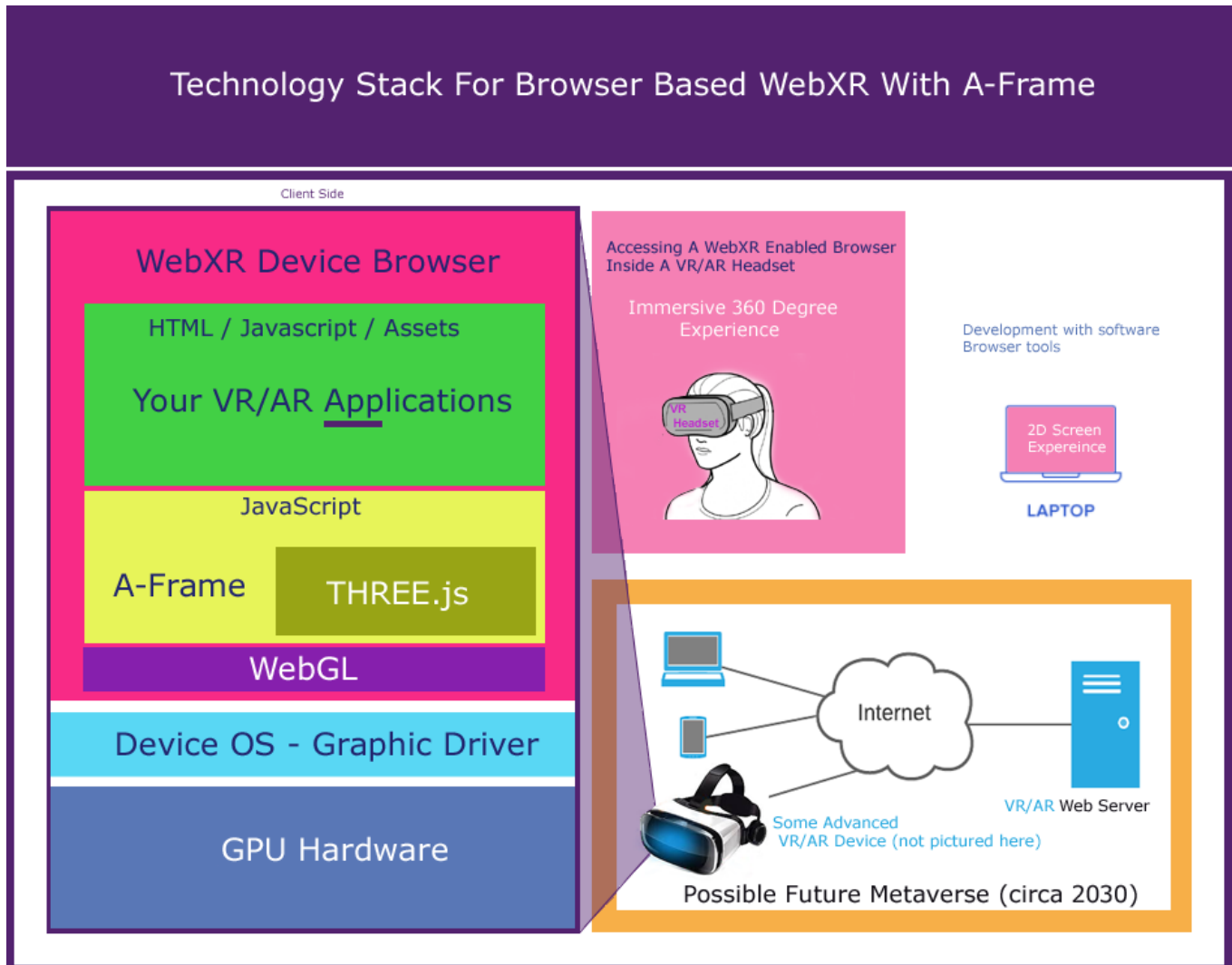
Hello Word example with A-Frame Inspector open.

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

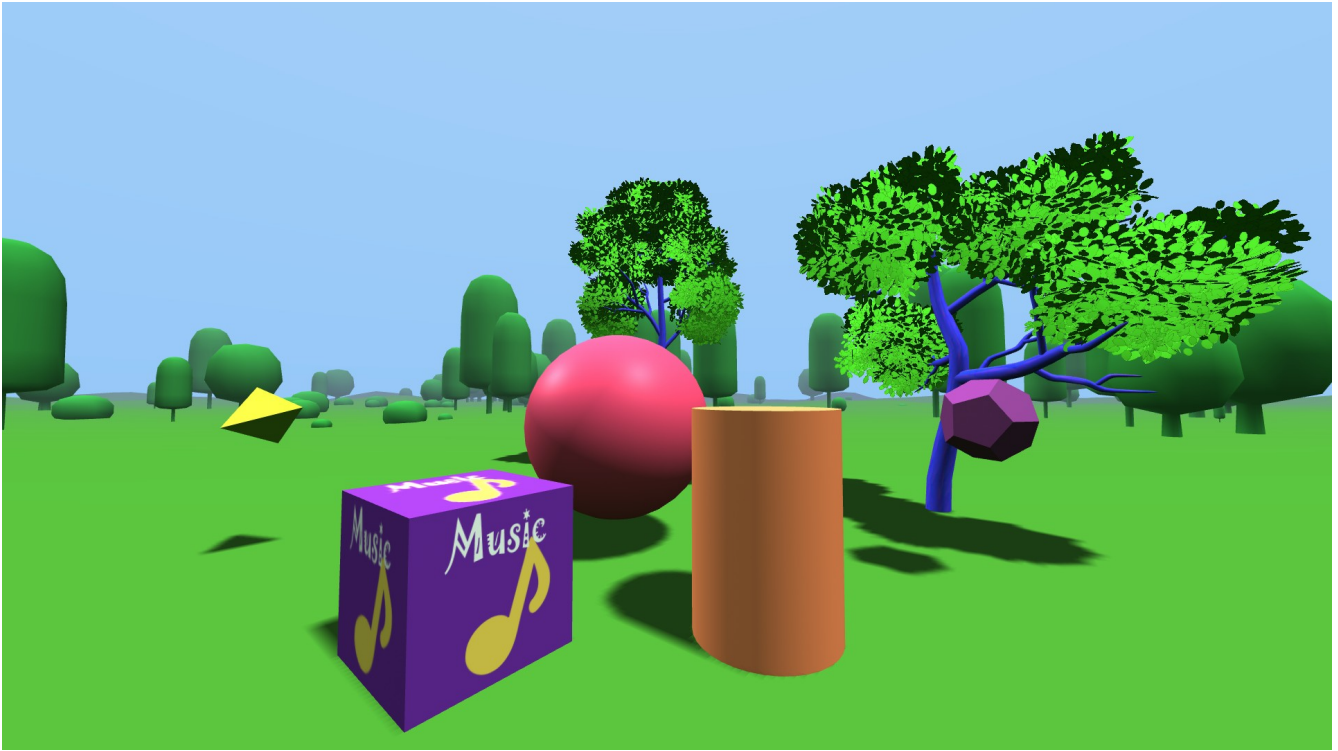
There is such a thing called a tick, or a frame of the scene's render loop. Meaning the scene is drawn before it's pushed into the VR headset on each frame (or tick) of a rendering loop. This usually happens many many times per second. It gives an appearance of smooth animation of objects, while not 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 webservice 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*.

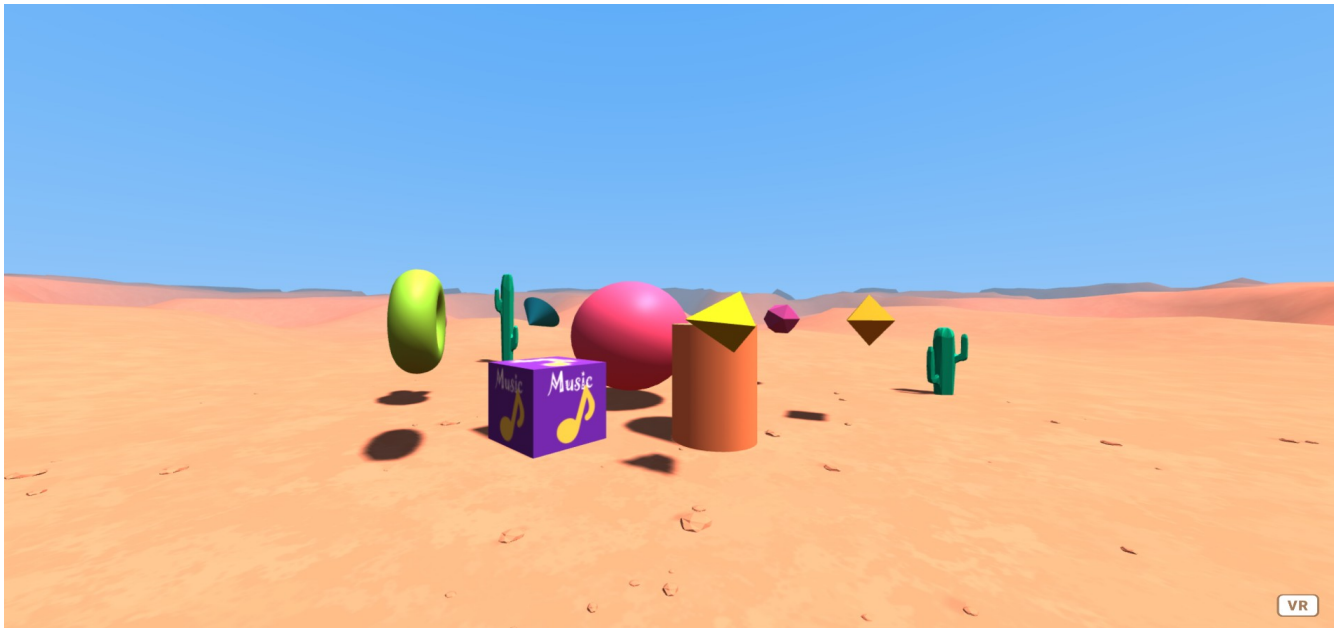
Finally, I might be stepping out on a shaky computer science limb here by saying, A-Frame programming of WebXR with its [entity component system](#), and attributes, might be considered Attribute programming, or Declarative programming, but when you add JavaScript it becomes Imperative programming. So, 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!

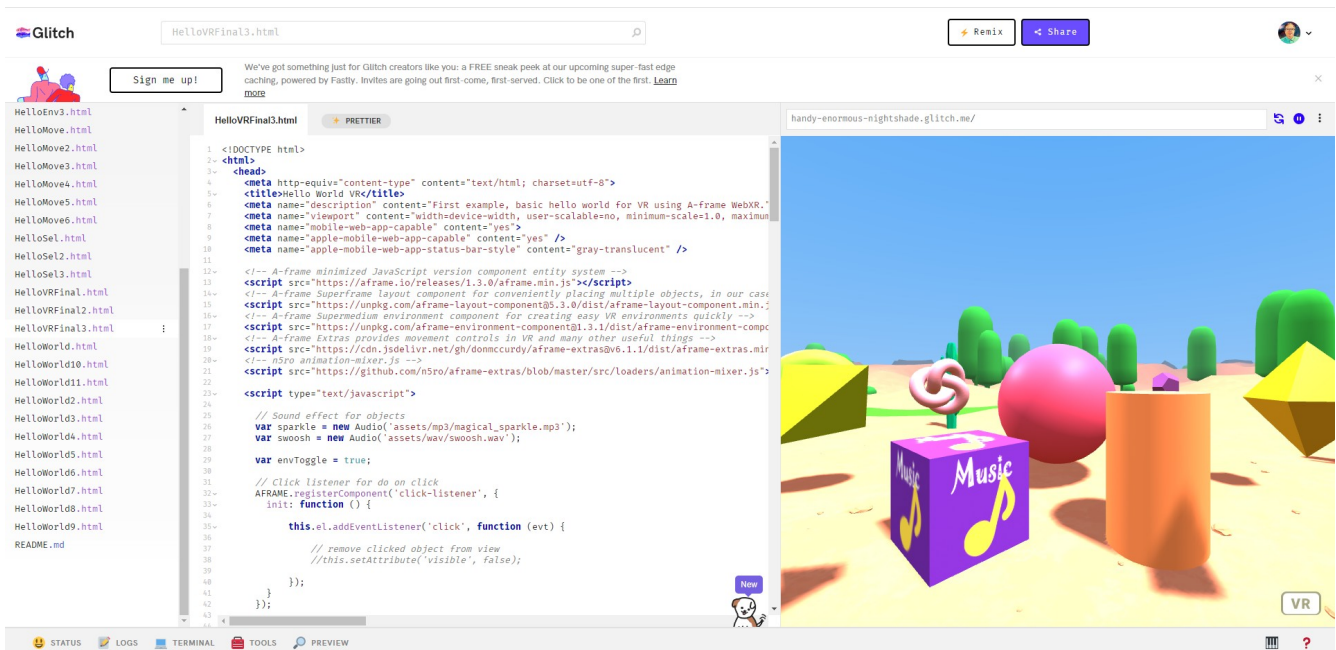


The Book Approach

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:

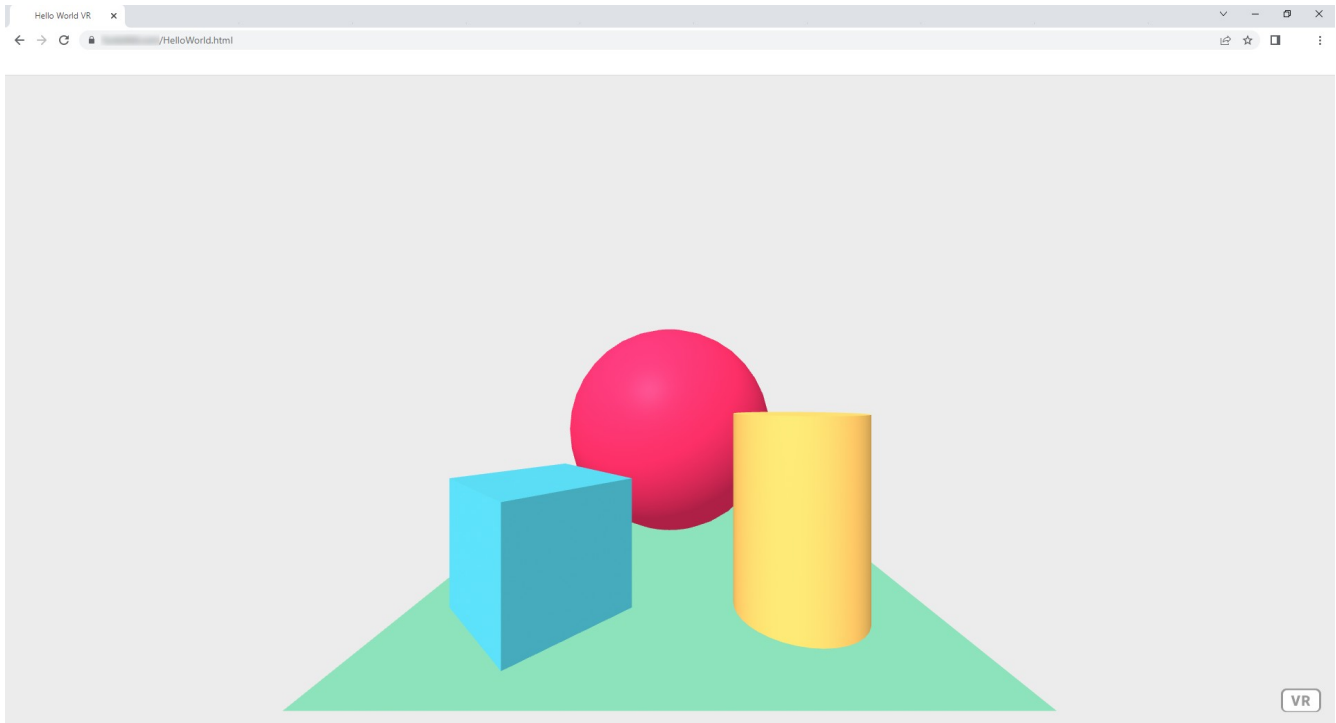
<https://github.com/Mike-McAnally/VRBook>

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/HelloWorld6.html>
- <https://rocketvirtual.com/HelloWorld7.html>
- <https://rocketvirtual.com/HelloWorld8.html>
- <https://rocketvirtual.com/HelloWorld9.html>

[https://rocketvirtual.com/HelloWorld10.html](https://rocketvirtual.com>HelloWorld10.html)
[https://rocketvirtual.com/HelloWorld11.html](https://rocketvirtual.com>HelloWorld11.html)
<https://rocketvirtual.com/HelloEnv.html>
<https://rocketvirtual.com/HelloEnv2.html>
<https://rocketvirtual.com/HelloEnv3.html>
<https://rocketvirtual.com/HelloMove.html>
<https://rocketvirtual.com/HelloMove2.html>
<https://rocketvirtual.com/HelloMove3.html>
<https://rocketvirtual.com/HelloMove4.html>
<https://rocketvirtual.com/HelloMove5.html>
<https://rocketvirtual.com/HelloMove6.html>
<https://rocketvirtual.com/HelloSel.html>
<https://rocketvirtual.com/HelloSel2.html>
<https://rocketvirtual.com/HelloSel3.html>
<https://rocketvirtual.com/HelloVRFinal.html>
<https://rocketvirtual.com/HelloVRFinal2.html>
<https://rocketvirtual.com/HelloVRFinal3.html>

Chapter 1 - Hello World VR!



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

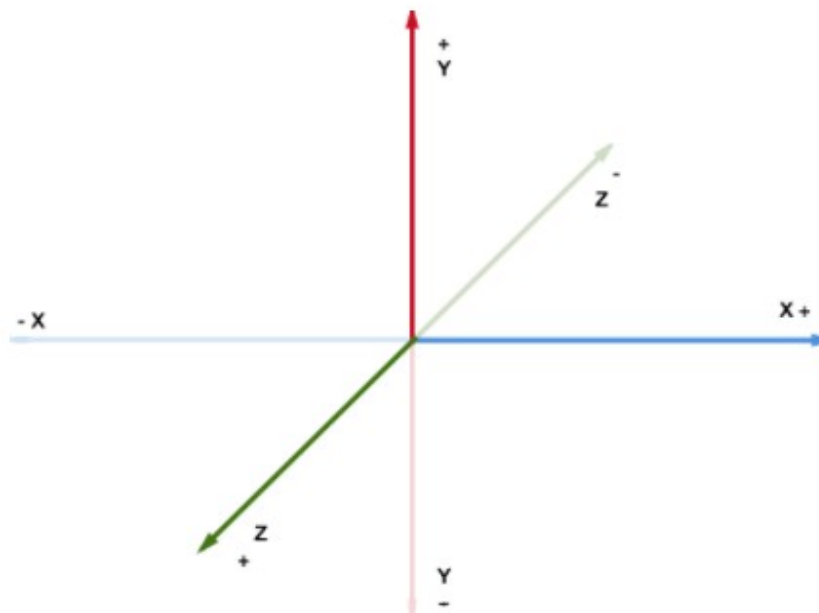
```
1 |<!DOCTYPE html>
2 |<html>
3 |  <head>
4 |    <meta http-equiv="content-type" content="text/html; charset=utf-8">
5 |    <title>Hello World VR</title>
6 |    <meta name="description" content="First example, basic hello world for VR using A-frame WebXR.">
7 |    <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0, shrink-to-fit=no">
8 |    <meta name="mobile-web-app-capable" content="yes">
9 |    <meta name="apple-mobile-web-app-capable" content="yes" />
10 |   <meta name="apple-mobile-web-app-status-bar-style" content="gray-translucent" />
11 |
12 |   <!-- A-frame minimized JavaScript version component entity system -->
13 |   <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
14 |
15 | </head>
16 | <body>
17 |   <!-- Used to create a scene, like on the stage of a play, but in 360 ° -->
18 |   <a-scene>
19 |
20 |     <!-- 3D primitive objects inside our scene, positioned x, y, z and rotation, and color -->
21 |     <!-- from our default created camera at view point 0, 1.6, 0 -which is 1.6 meters about the ground -->
22 |     <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
23 |     <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
24 |     <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
25 |
26 |     <!-- The plane upon which the objects are virtually positioned -->
27 |     <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
28 |
29 |     <!-- We are inside a gigantic hollow sphere, looking at its inner surface painted a light color of gray -->
30 |     <a-sky color="#ECECEC"></a-sky>
31 |   </a-scene>
32 | </body>
33 | </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.

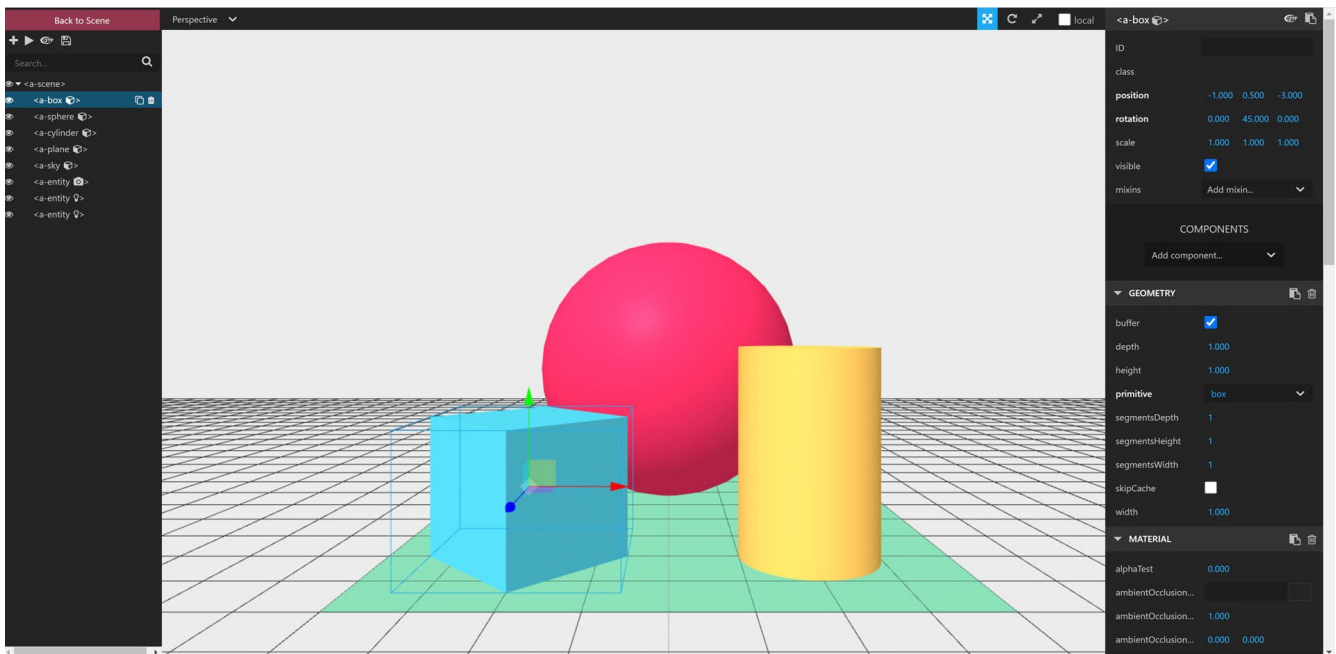
Because our `<a-box>` has `position=-1 0.5 -3` means it appears -1 to the left, raised 0.5 up and -3 back into the screen, if that makes sense. The same goes for the other geometry shapes in our scene which all have their own positions defined.

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.

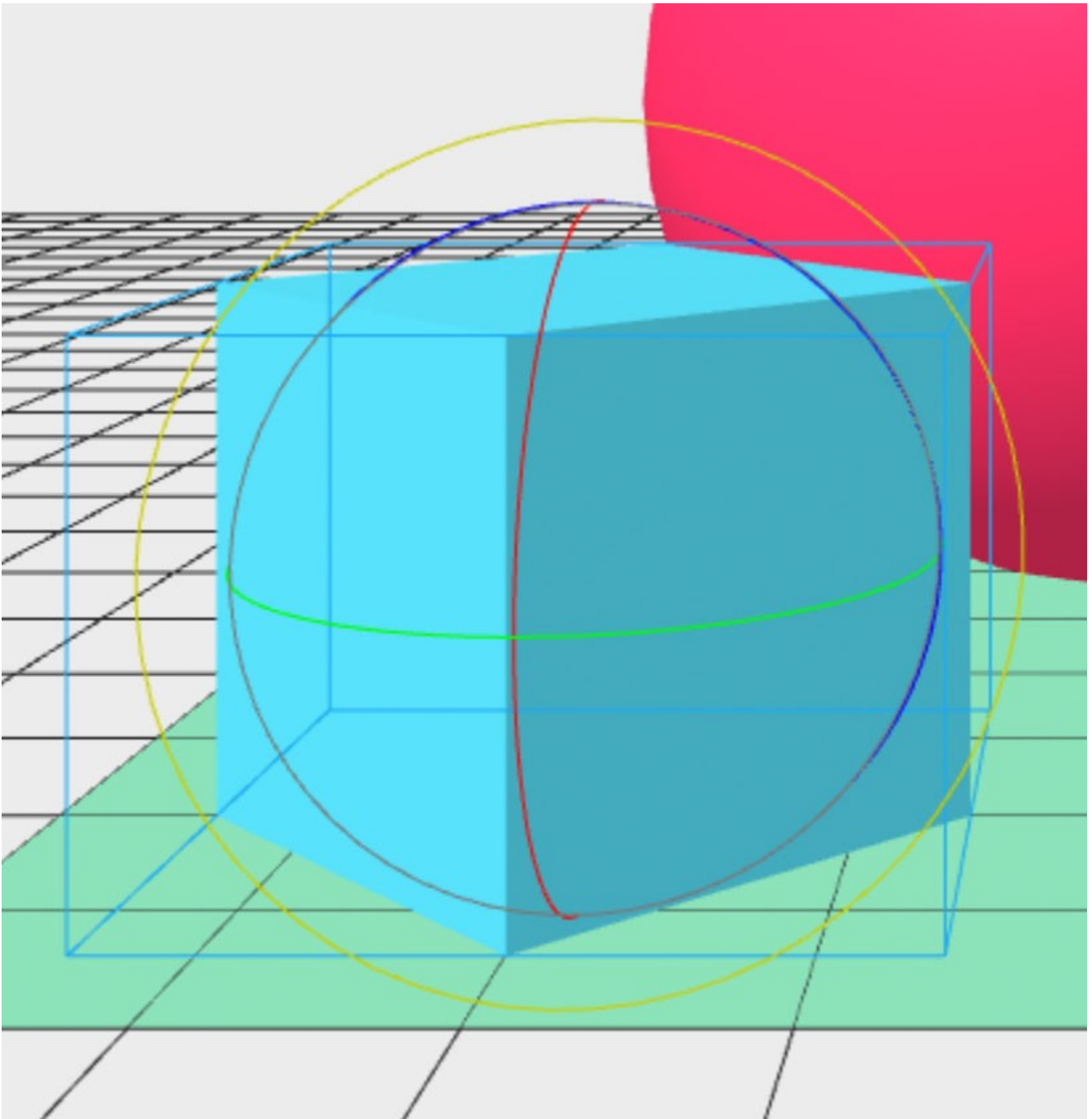
To enter the inspector, press the Ctrl Alt and i keys simultaneously (<Ctrl> + <Alt> + i).

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.



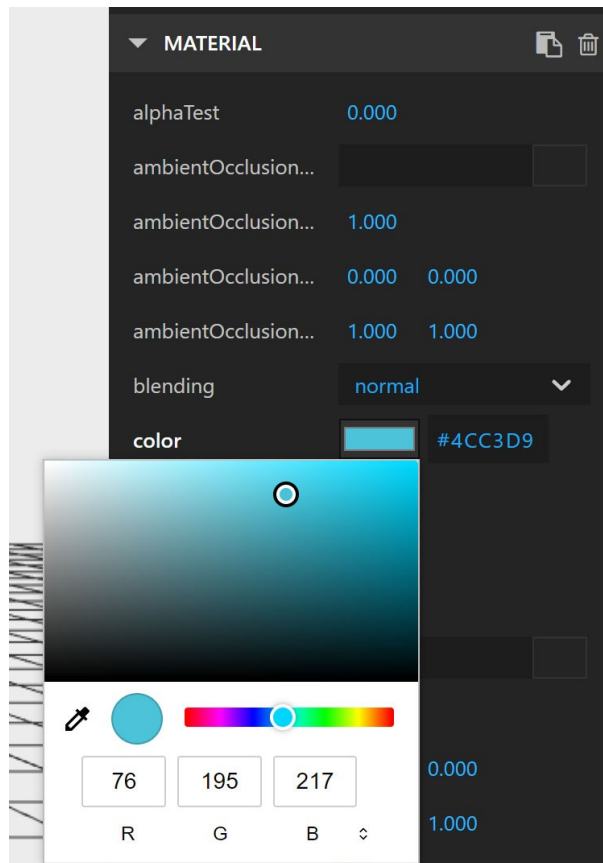
Notice the controls on the top, just slightly left of the right panel. Select the rotation circle control, like this.





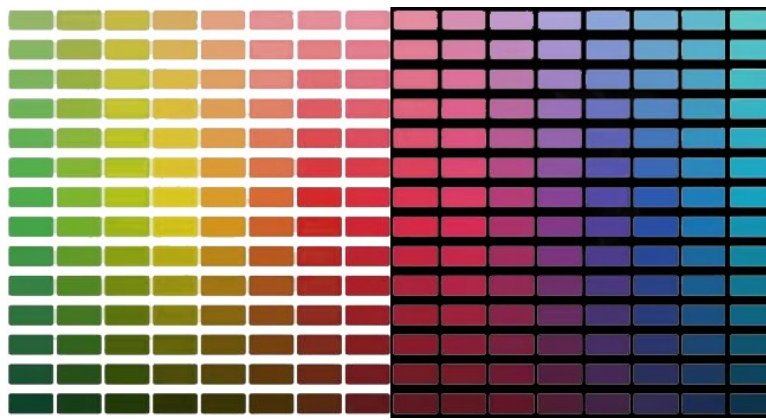
You will now notice a change to the <a-box>.

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.

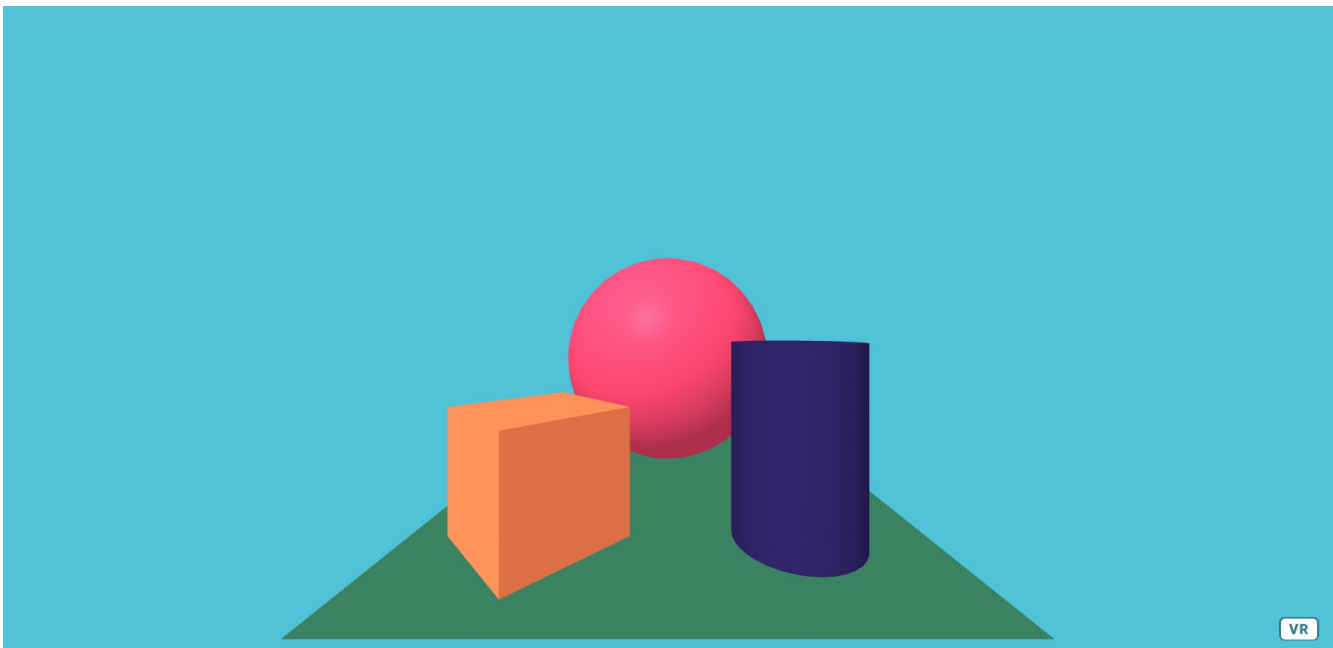


A multitude of color to choose from

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.

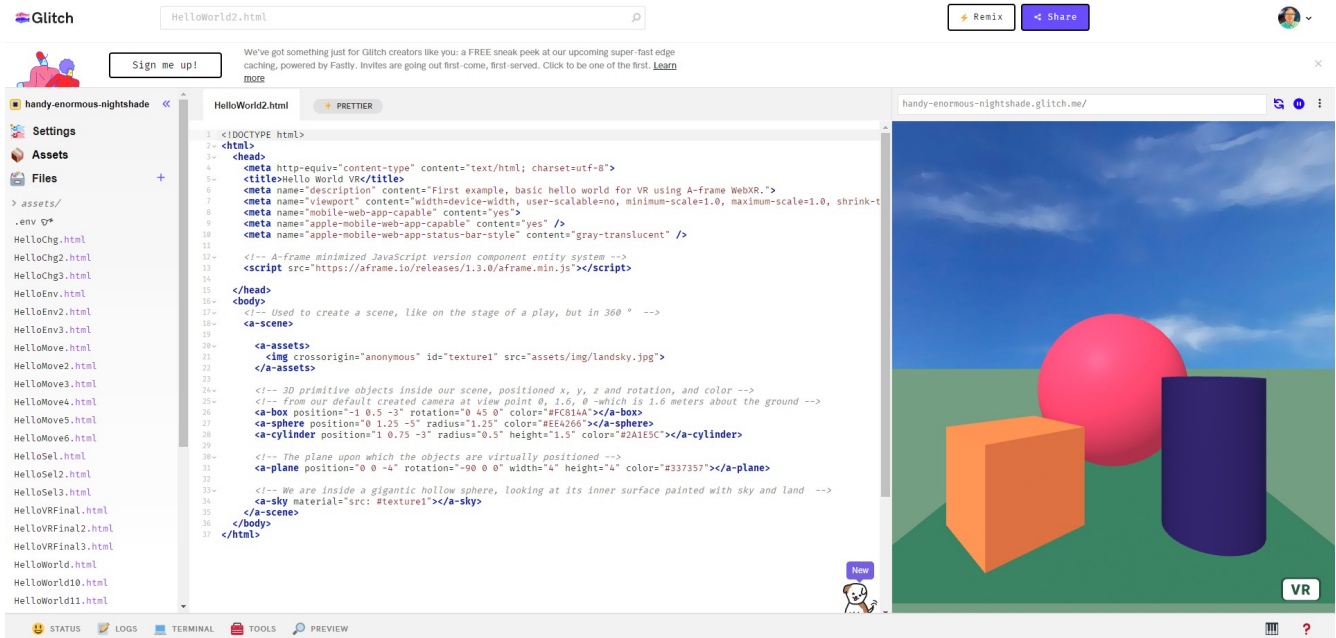
Color Generator link
<https://coolors.co>



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.



```
12 <!-- A-frame minimized JavaScript version component entity system -->
13 <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
14
15 </head>
16 <body>
17 <!-- Used to create a scene, like on the stage of a play, but in 360 ° -->
18 <a-scene>
19
20   <a-assets>
21     
22   </a-assets>
23
24   <!-- 3D primitive objects inside our scene, positioned x, y, z and rotation, and color -->
25   <!-- from our default created camera at view point 0, 1.6, 0 -which is 1.6 meters about the ground -->
26   <a-box position="-1 0.5 -3" rotation="0 45 0" color="#FC814A"></a-box>
27   <a-sphere position="0 1.25 -5" radius="1.25" color="#EE4266"></a-sphere>
28   <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#2A1E5C"></a-cylinder>
29
30   <!-- The plane upon which the objects are virtually positioned -->
31   <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#337357"></a-plane>
32
33   <!-- We are inside a gigantic hollow sphere, looking at its inner surface painted with sky and land -->
34   <a-sky material="src: #texture1"></a-sky>
35 </a-scene>
36 </body>
37 </html>
```

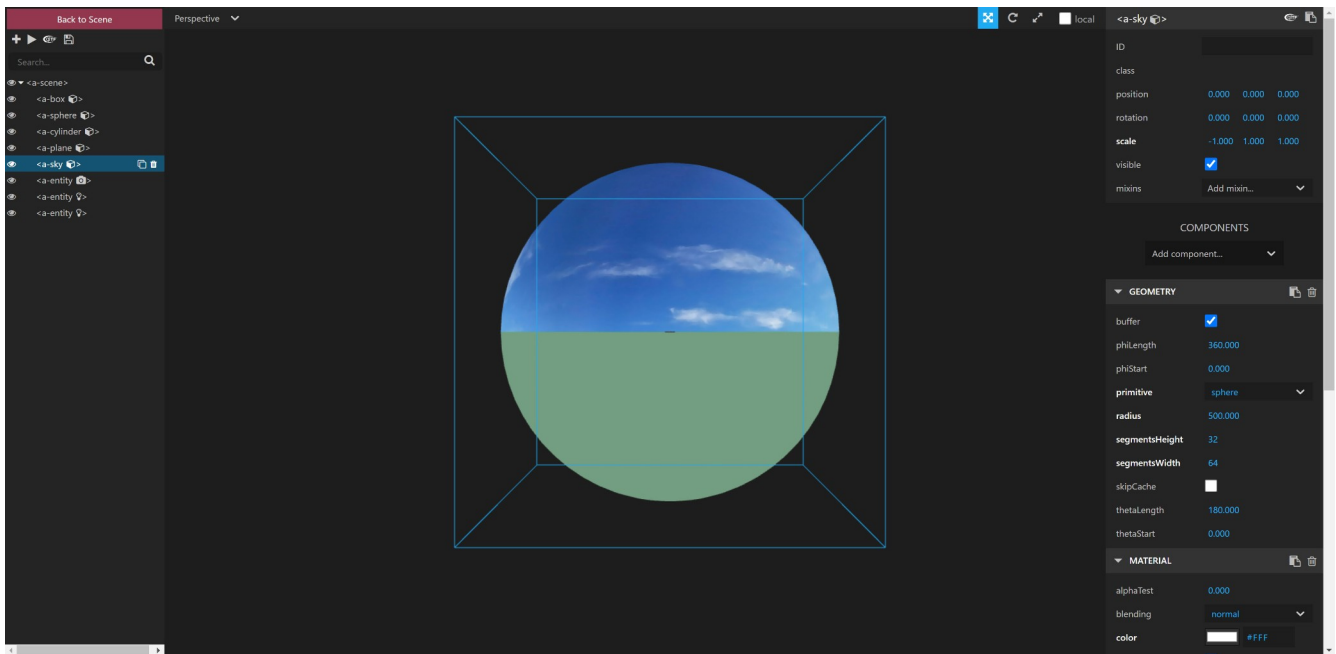
Take a look at the new source code [HelloWorld2.html](#) and [browser rendition](#).

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.



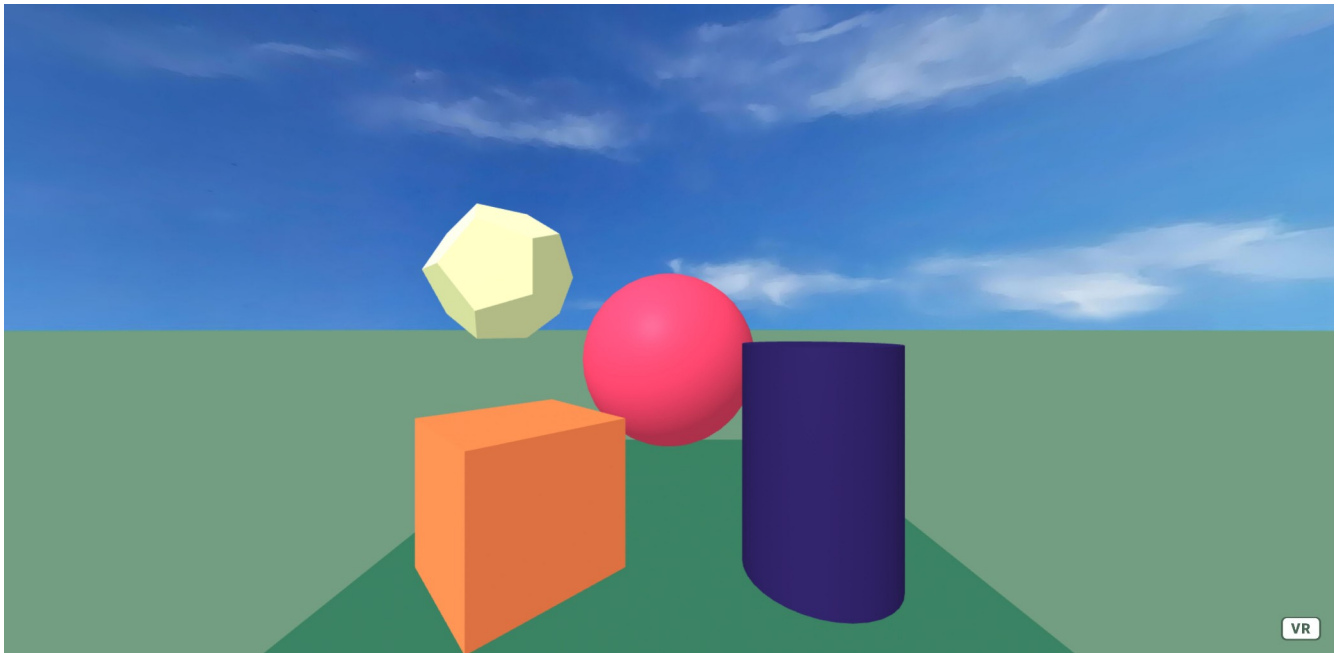
You can see this if you enter the A-Frame inspector again (`<Ctrl> + <Alt> + i`) then select `<a-sky>` in the left panel. Now scroll back away from the screen, keep scrolling a lot, until a sphere will appear with the `landsky.jpg` texture mapped onto it. *You should imagine yourself as inside this painted 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.

```
<a-entity geometry="primitive: dodecahedron; radius: 0.75" position="-1.8 2.2 -4.6" rotation="0 45 0" material="color: #F3FFB6"></a-entity>
<a-entity geometry="primitive: box" position="-1 0.5 -3" rotation="0 45 0" material="color: #FC814A" ></a-entity>
<a-entity geometry="primitive: sphere" position="0 1.25 -5" radius="1.25" material="color: #EE4266" ></a-entity>
<a-entity geometry="primitive: cylinder; height: 1.5; radius: 0.5" position="1 0.75 -3" material="color: #2A1ESC" ></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.



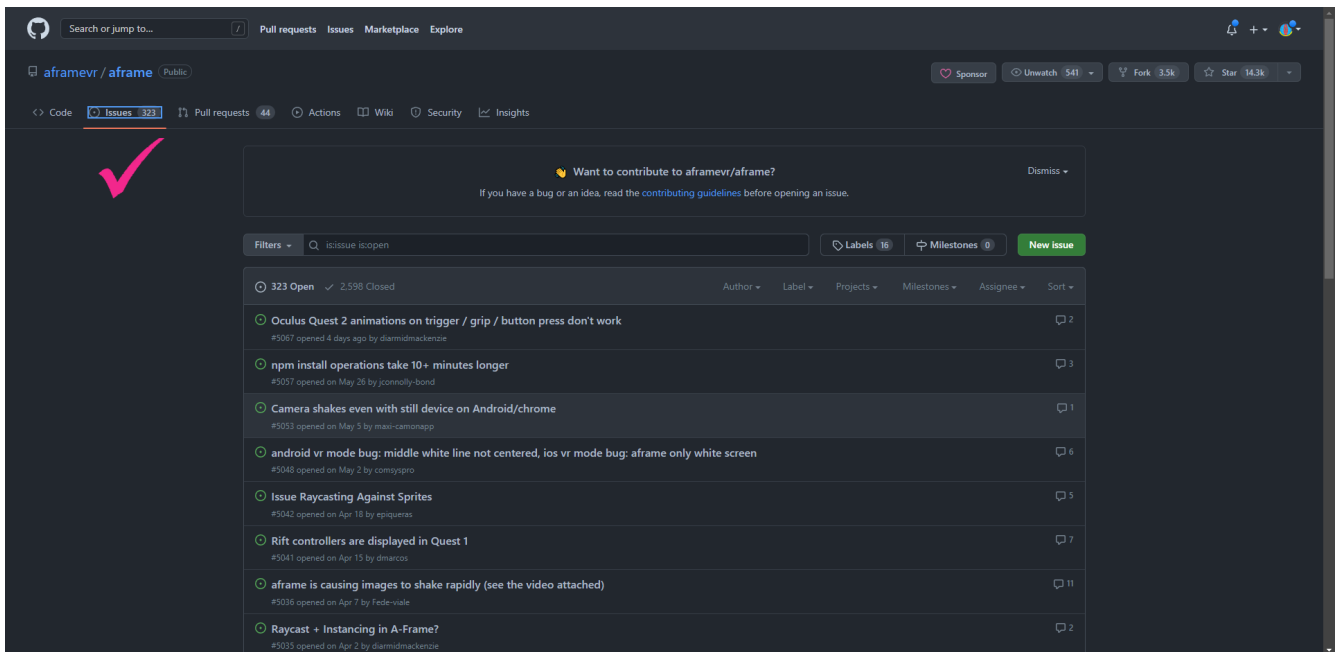
[Our modified scene should look something like this.](#) [In Glitch this is the HelloWorld4.html.](#)

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.

```
<!-- 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.



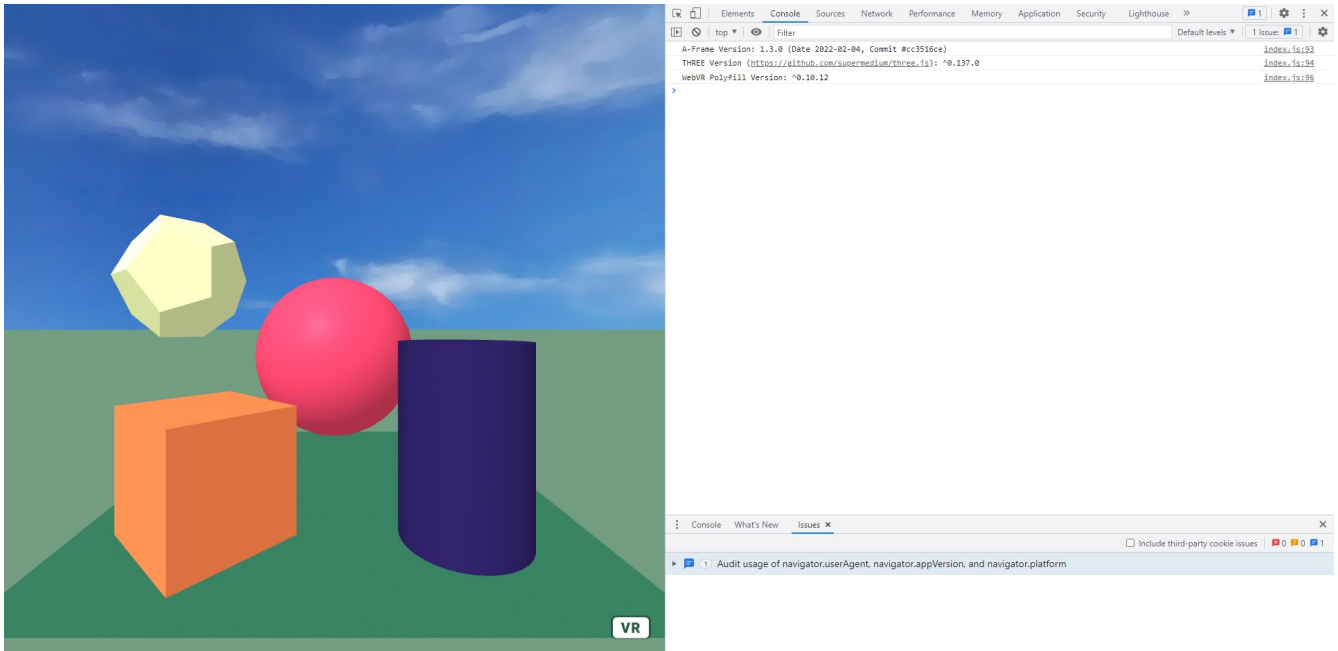
Issues with latest A-Frame master and releases can be found under the issues tab on GitHub.

[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.](#)



Console tab in Chrome inspect showing version of A-Frame with any possible warnings, errors and info.

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.

```
<!-- A-frame minimized JavaScript version component entity system -->
<script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
<script src="https://unpkg.com/aframe-layout-component@4.3.1/dist/aframe-layout-component.min.js"></script>
</head>
<body>
  <!-- Used to create a scene, like on the stage of a play, but in 360 ° -->
  <a-scene>
    <a-assets>
      
    </a-assets>

    <!-- 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 about the ground -->

    <!-- A-frame layout component creates a circle around a position with a radius on the x-z plane (notice no positions are defined on the objects within the layout </entity>) -->
    <a-entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" >

      <!-- Scale was set to half, .5 for each entity and opacity was also set to half, giving each object a transparent see through look -->
      <a-entity id="cone" geometry="primitive: cone" rotation="0 45 0" scale=".5 .5 .5" material="color: #1DA7C3; opacity: .5"></a-entity>
      <a-entity id="dodecahedron" geometry="primitive: dodecahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #7D347C; opacity: .5"></a-entity>
      <a-entity id="icosahedron" geometry="primitive: icosahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A7346D; opacity: .5"></a-entity>
      <a-entity id="octahedron" geometry="primitive: octahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #D19418; opacity: .5"></a-entity>
      <a-entity id="tetrahedron" geometry="primitive: tetrahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A4B615; opacity: .5"></a-entity>
      <a-entity id="torus" geometry="primitive: torus; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #82B32E; opacity: .5"></a-entity>
      <a-entity id="torusknot" geometry="primitive: torusknot; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #E38097; opacity: .5"></a-entity>

    <!-- This is the close block for layout -->
    </a-entity>

    <a-entity id="cube" geometry="primitive: box" position="-1 0.5 -3" rotation="0 45 0" material="color: #FC814A" ></a-entity>
    <a-entity id="sphere" geometry="primitive: sphere" position="0 1.25 -5" radius="1.25" material="color: #EE4266" ></a-entity>
    <a-entity id="cylinder" geometry="primitive: cylinder; height: 1.5; radius: 0.5" position="1 0.75 -3" material="color: #2A1E5C" ></a-entity>

    <!-- The plane upon which the objects are virtually positioned -->
    <a-plane id="plane" position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#337357"></a-plane>

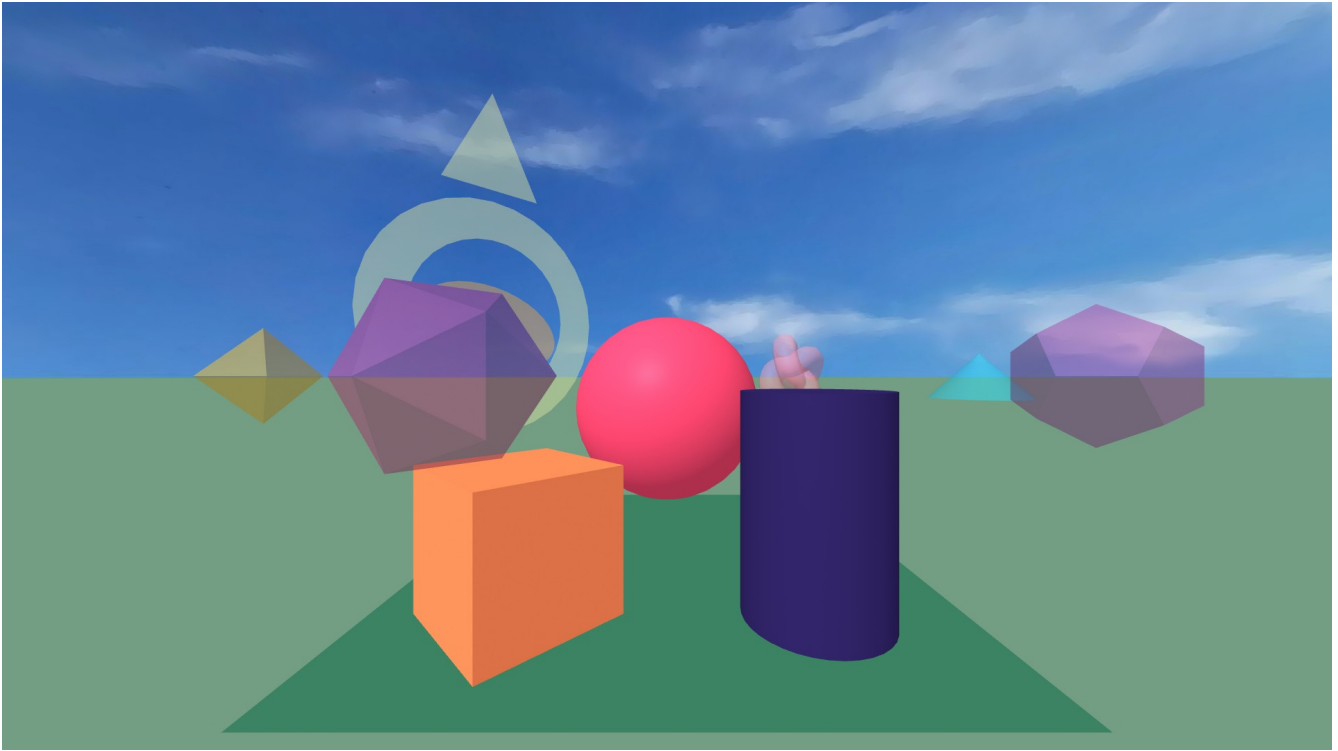
    <!-- Other planar (flat) 2D objects -->
    <a-entity id="triangle" geometry="primitive: triangle" position="-1.8 4.01656 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: .5"></a-entity>
    <a-entity id="ring" geometry="primitive: ring" position="-1.8 2.2 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: .5"></a-entity>
    <a-entity id="circle" geometry="primitive: circle; radius: 0.75" position="-2 2.2 -5.08638" rotation="-43 45 -20" material="color: #CD9844; opacity: .5"></a-entity>

    <!-- We are inside a gigantic hollow sphere, looking at its inner surface painted with sky and land -->
    <a-sky id="sky" material="src: #texture1"></a-sky>
  </a-scene>
```

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.](#)

Chapter 2 – Animation, Lights And Shadows



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](#).

```

<a-scene>
  <a-assets>
    <!-- this is a mixin which provides for a looping spin on the y-axis -->
    <a-mixin id="spin-y" animation="property: rotation; to: 0 360 0; loop: true; dur: 10000; easing: linear"></a-mixin>
    <a-mixin id="spin-x" animation="property: rotation; to: 360 0 0; loop: true; dur: 10000; easing: linear"></a-mixin>
    <a-mixin id="spin-z" animation="property: rotation; to: 0 0 360; loop: true; dur: 10000; easing: linear"></a-mixin>

    
  </a-assets>

  <!-- 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 about the ground -->

  <!-- A-frame layout component creates a circle around a position with a radius on the x-z plane (notice no positions are defined on the objects within the layout <entity>) -->
  <a-entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" >

    <!-- Scale was set to half, .5 for each entity and opacity was also set to half, giving each object a transparent see through look -->
    <a-entity id="cone" geometry="primitive: cone" rotation="0 45 0" scale=".5 .5 .5" material="color: #1DA7C3; opacity: .5" mixin="spin-x"></a-entity>
    <a-entity id="dodecahedron" geometry="primitive: dodecahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #7D347C; opacity: .5" mixin="spin-y"></a-entity>
    <a-entity id="icosahedron" geometry="primitive: icosahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A7346D; opacity: .5" mixin="spin-y"></a-entity>
    <a-entity id="octahedron" geometry="primitive: octahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #D19418; opacity: .5" mixin="spin-y"></a-entity>
    <a-entity id="tetrahedron" geometry="primitive: tetrahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A4B615; opacity: .5" mixin="spin-z"></a-entity>
    <a-entity id="torus" geometry="primitive: torus; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #82B32E; opacity: .5" mixin="spin-y"></a-entity>
    <a-entity id="torusknot" geometry="primitive: torusknot; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #E38097; opacity: .5" mixin="spin-y"></a-entity>

  <!-- This is the close block for layout -->
  </a-entity>

  <a-entity id="cube" geometry="primitive: box" position="-1 0.5 -3" rotation="0 45 0" material="color: #FC814A" ></a-entity>
  <a-entity id="sphere" geometry="primitive: sphere" position="0 1.25 -5" radius="1.25" material="color: #EE4266" ></a-entity>
  <a-entity id="cylinder" geometry="primitive: cylinder; height: 1.5; radius: 0.5" position="1 0.75 -3" material="color: #2A1E5C" ></a-entity>

  <!-- The plane upon which the objects are virtually positioned -->
  <a-plane id="plane" position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#337357"></a-plane>

  <!-- Other planar (flat) 2D objects -->
  <a-entity id="triangle" geometry="primitive: triangle" position="-1.8 4.01656 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: .5" mixin="spin-y"></a-entity>
  <a-entity id="ring" geometry="primitive: ring" position="-1.8 2.2 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: .5" mixin="spin-x"></a-entity>
  <a-entity id="circle" geometry="primitive: circle; radius: 0.75" position="-2 2.2 -5.08638" rotation="-43 45 -20" material="color: #CD9044; opacity: .5" mixin="spin-z"></a-entity>

```

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](#).

```

<!-- The plane upon which the objects are virtually positioned -->
<a-plane id="plane" position="0 0 -4" rotation="-90 0 0" width="12" height="12" color="#337357" shadow="receive: true"></a-plane>

<!-- Other planar (flat) 2D objects -->
<a-entity id="triangle" geometry="primitive: triangle" position="-1.8 4.01656 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: 1; side: double" mixin="spin-y" shadow="receive: false"></a-entity>
<a-entity id="ring" geometry="primitive: ring" position="-1.8 2.2 -4.6" rotation="0 45 0" material="color: #F3FFB6; opacity: 1; side: double" mixin="spin-x" shadow="receive: false"></a-entity>
<a-entity id="circle" geometry="primitive: circle; radius: 0.75" position="-2 2.2 -5.08638" rotation="-43 45 -20" material="color: #CD9044; opacity: 1; side: double" mixin="spin-y" shadow="receive: false"></a-entity>

<!-- We are inside a gigantic hollow sphere, looking at its inner surface painted with sky and land -->
<a-sky id="sky" material="src: #texture1"></a-sky>

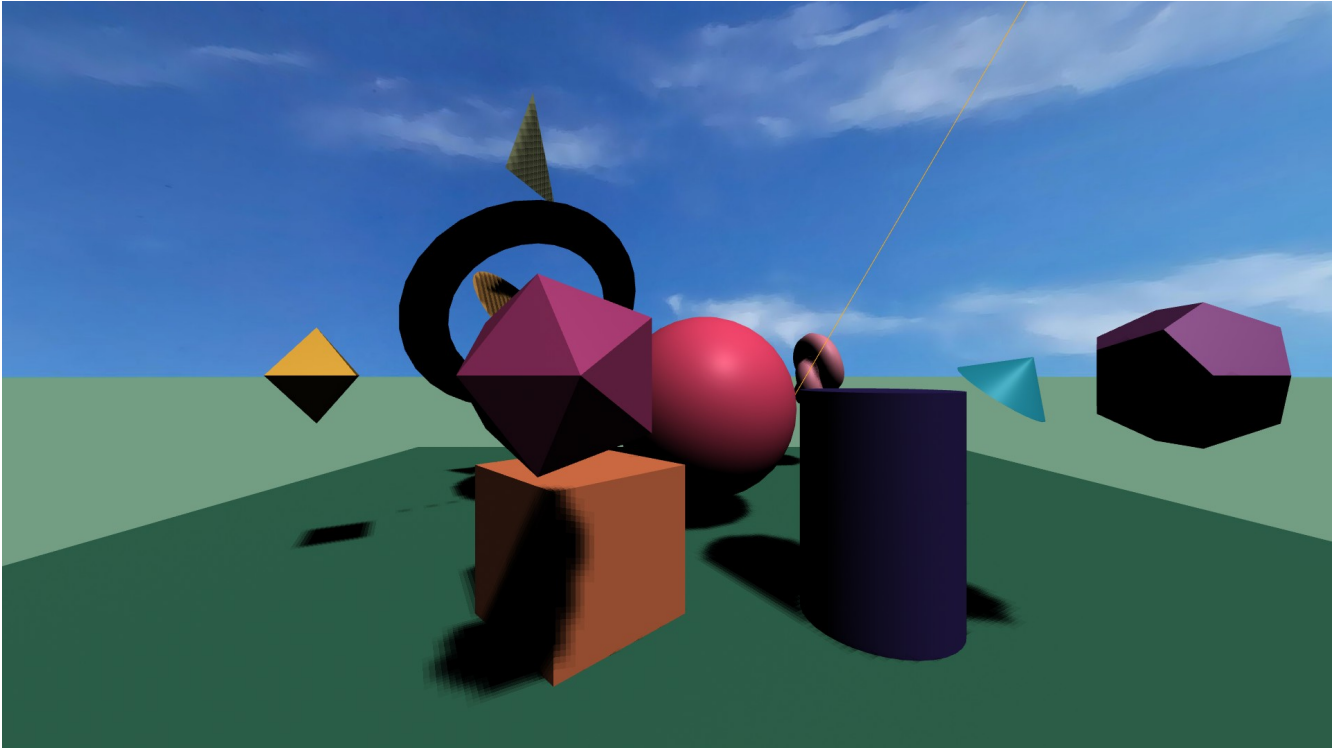
<!-- A light in the scene with shadow camera casting shadows -->
<a-entity light="castShadow: true; shadowCameraLeft: -10; shadowCameraBottom: -10; shadowCameraRight: 20; shadowCameraTop: 10; shadowCameraVisible: false; shadowCameraFar: 30" position="5.0091 13.0395 8.5074"></a-entity>

<!-- An ambient light to brighten scene -->
<a-entity light="intensity: 0.7; type: ambient" visible=""></a-entity>
</a-scene>

```

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.](#)

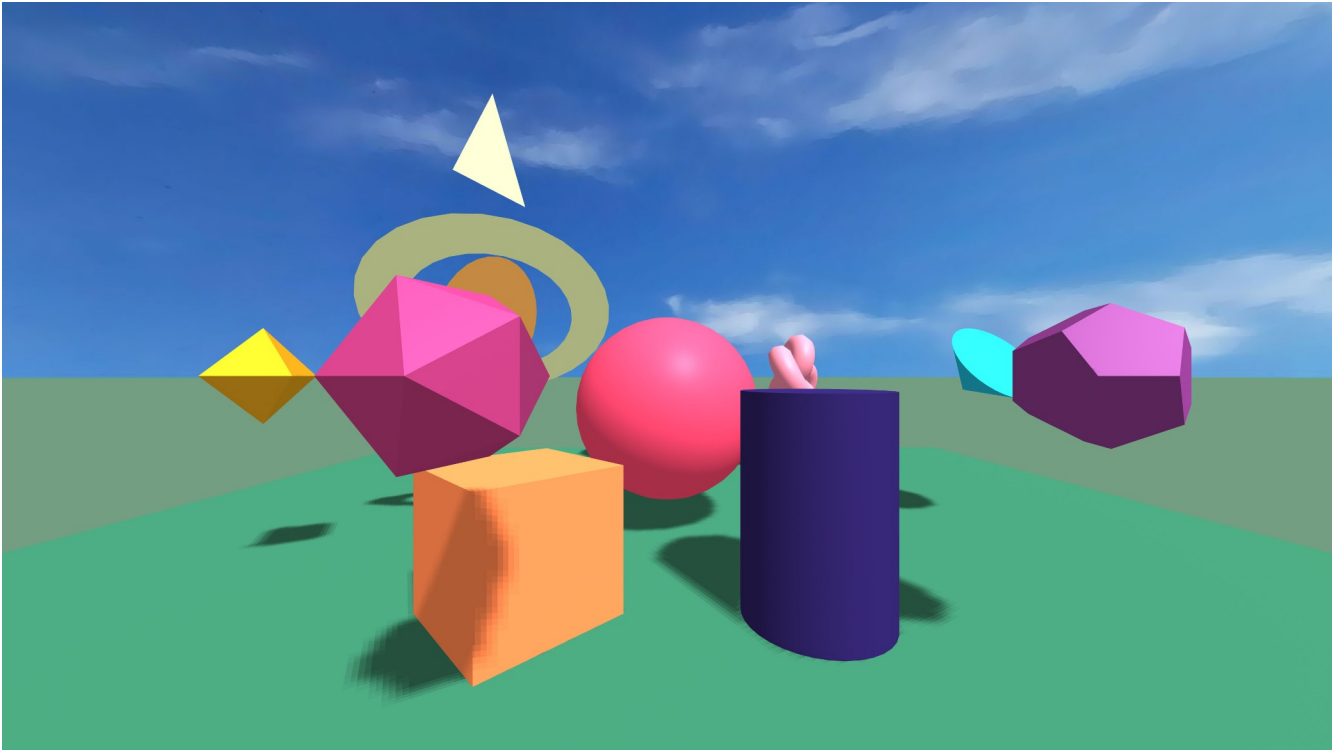


With our light and shadow camera added our scene object shadows have darkened a little bit. To fix that we will add an ambient light to brighten things up, setting its intensity to 0.7.

```
<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>
```

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.

```
<!-- A-frame layout component creates a circle around a position with a radius on the x-z plane
<a-entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" mixin="spin-y">
```

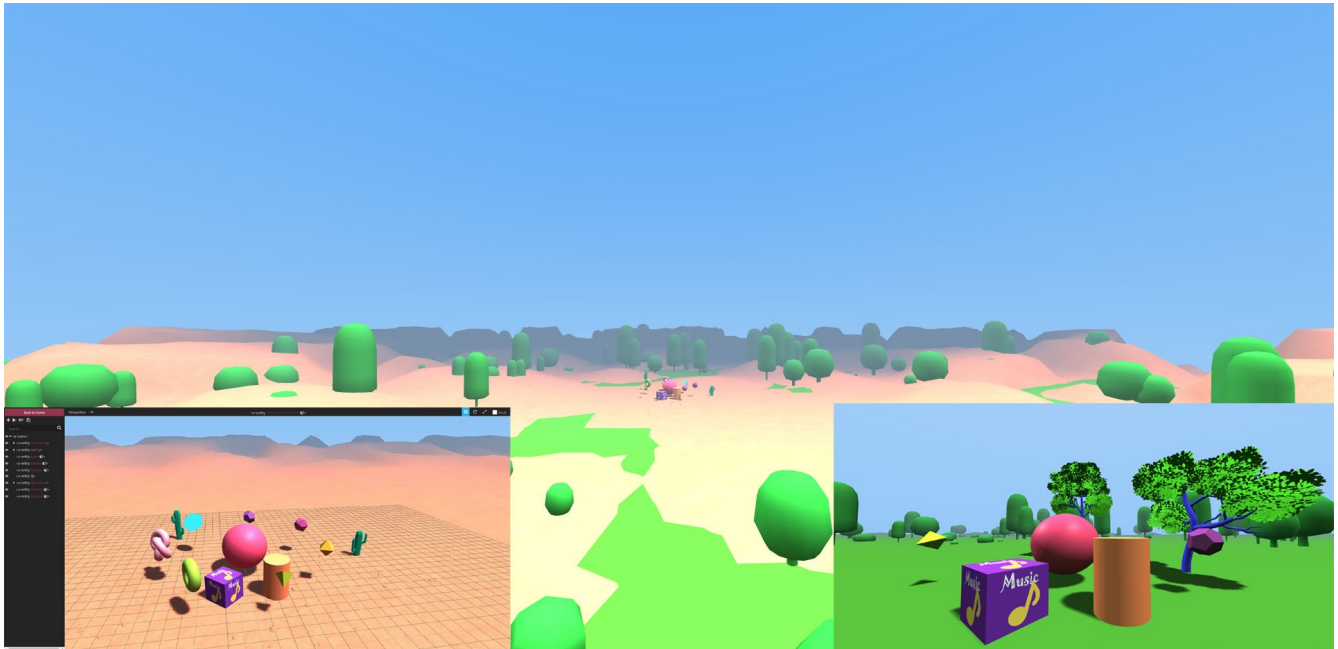
Now for fun, and a more dramatic effect add the spin-y mixin to the layout circle. It causes the whole layout of the objects in the circle to spin around. *That is powerful animation!*

```
<!-- A-frame layout component creates a circle around a position with a radius on the x-z plane (notice no positions are defined on the objects within the layout </entity>) -->
<a-entity layout="type: circle; radius: 3.5; plane: xz;" position="0 1.6 -5" mixin="spin-y" animation_2="property: position; from: 0 1.6 -5 ; to: 0 1 -5; dir: alternate; loop: true">
```

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

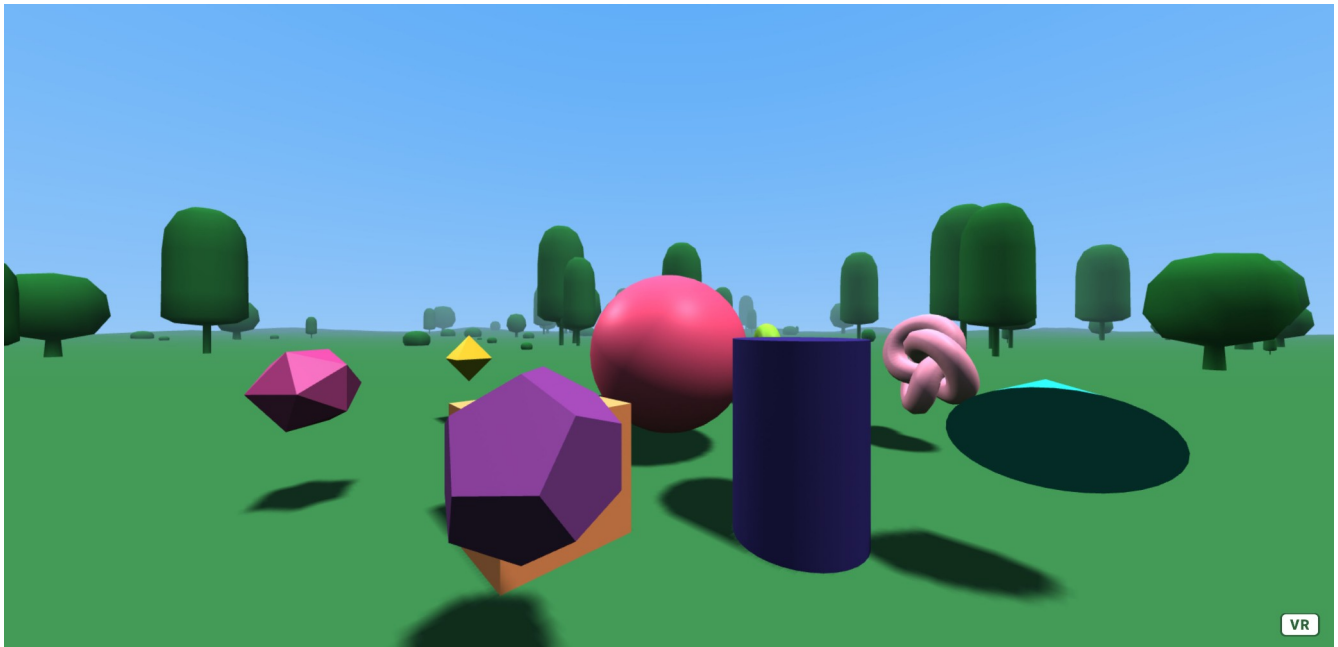
Our original landsky environment was simple and effective. However, persons inside of VR are known to explore more extensively, much as they do in the real world, and usually require a richer, more immersive environment. To accomplish this with the least amount of effort, we will introduce and use the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

```
<!-- A-frame supermedium environment component for creating easy VR environments quickly -->  
<script src="https://unpkg.com/aframe-environment-component@1.3.1/dist/aframe-environment-component.min.js"></script>
```

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

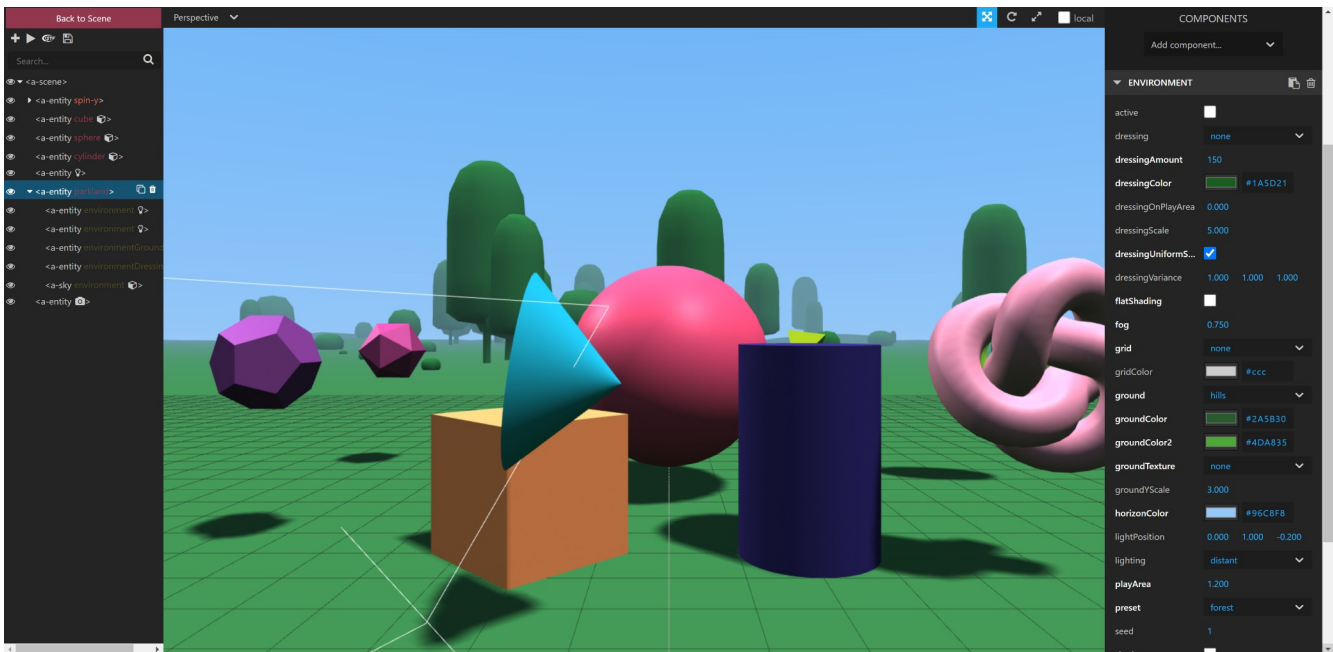
```
<!-- An ambient light to brighten scene  
<a-entity light="intensity: 0.7; type: ambient" visible=""></a-entity -->  
  
<!-- Our Environment -->  
<a-entity id="parkland" environment="preset: forest; playArea: 1.2; dressingAmount: 150; dressingUniformScale: true; dressingColor: #1A5D21; ground: hills; groundTexture: none; groundColor: #2A5B30; groundColor2: #4D8B35; grid: none; skyType: gradient; skyColor: #4BA3F8; horizonColor: #96CAF8; fog: .75; flatShading: false" shadow="recieve: true" ></a-entity>  
</a-scene>
```

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.*

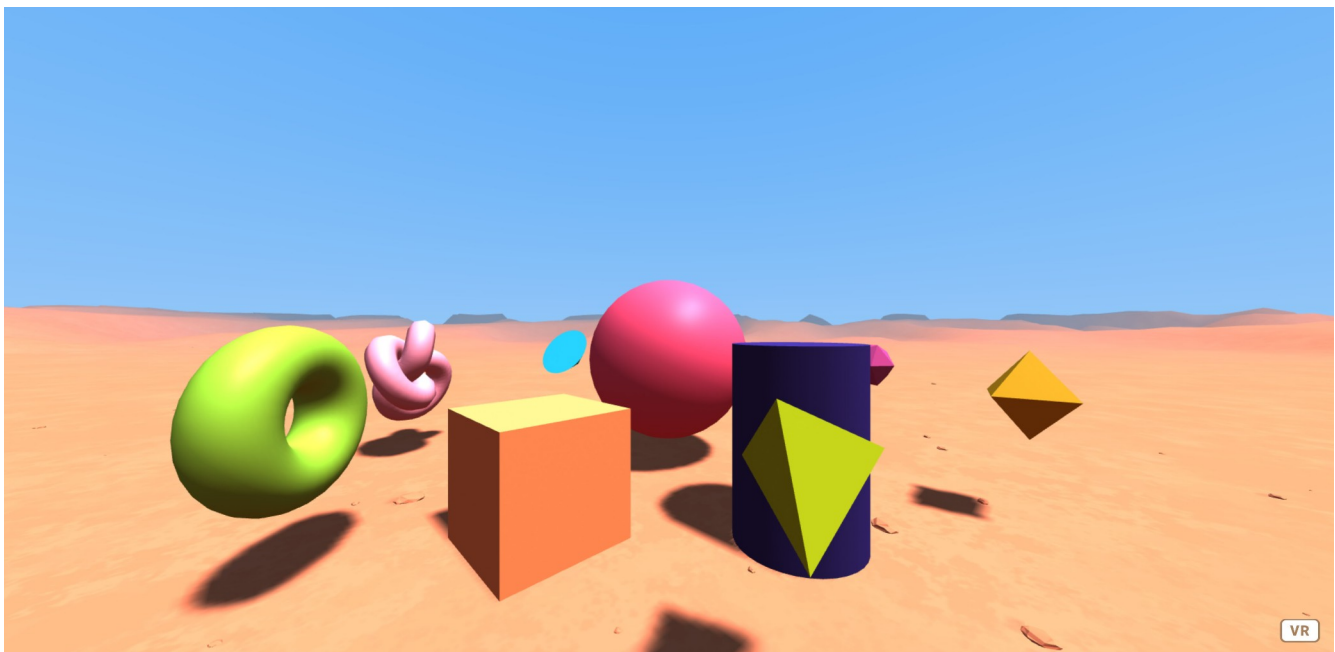


Any changes you make will need to be copied back to the source for them to become permanent. You can copy those attributes back by selecting one of the copy symbols in the right panel, then pasting the text into the source.

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.

```
<!-- Our Environment
<a-entity id="parkland" environment="preset: forest; playArea: 1.2; dressingAmount: 150; dressingUniformScale: true; dressingColor: #1A5D21; ground: hills; groundTexture: none; groundColor: #2A5B30;
groundColor2: #4D8835; grid: none; skyType: gradient; skyColor: #4BA3F8; horizonColor: #96C8F8; fog: .75; flatShading: false" shadow="receive: true" ></a-entity> -->

<!-- Desert - Arizona like environment -->
<a-entity id="desertland" environment="preset: yavapai; playArea: 1.2; dressingAmount: 150; skyType: gradient; skyColor: #4BA3F8; horizonColor: #96C8F8; fog: .75; flatShading: false" shadow="
receive: true" ></a-entity>
```



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!

```
<!-- A-Frame Extras provides movement controls in VR and many other useful things -->
<script src="https://cdn.jsdelivr.net/gh/donmccurdy/aframe-extras@v6.1.1/dist/aframe-extras.min.js"></script>
</head>
<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>
    <!-- this is a mixin which provides for a looping spin on the y-axis -->
    <a-mixin id="spin-y" animation="property: rotation; to: 0 360 0; loop: true; dur: 15000; easing: linear; dir: alternate"></a-mixin>
    <a-mixin id="spin-x" animation="property: rotation; to: 360 0 0; loop: true; dur: 15000; easing: linear"></a-mixin>
    <a-mixin id="spin-z" animation="property: rotation; to: 0 0 360; loop: true; dur: 15000; easing: linear"></a-mixin>
  </a-assets>
  <!-- Basic movement in VR with lowpoly hands and raycaster laser pointer for selection -->
  <a-entity id="cameraRig" movement-controls >
    <!-- camera -->
    <a-entity id="head" camera="active: true" position="0 1.6 0" look-controls="pointerLockEnabled: true" ></a-entity>
    <!-- Left Controller Hand -->
    <a-entity id="leftHand" hand-controls="hand: left; handModelStyle: lowPoly; color: #15ACCF" visible="true"></a-entity>
    <!-- Right Controller Hand -->
    <a-entity id="rightHand" hand-controls="hand: right; handModelStyle: lowPoly; color: #15ACCF" laser-controls raycaster="showLine: true; far: 10; interval: 0; objects: .clickable, a-link;"
      line="color: #7cfc00; opacity: 0.5" visible="true"></a-entity>
  </a-entity>
</a-scene>
</body>
</html>
```

[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](#).

```
<!-- 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>
```

```
<!-- Desert -Arizona like environment -->
<a-entity id="desertland" environment="preset:yavapai; playArea: 1.2; dressingAmount: 150; skyType: gradient; skyColor: #4BA3F8; horizonColor: #96C8F8; fog: .75; flatShading: false" shadow="receive: true" ></a-entity>

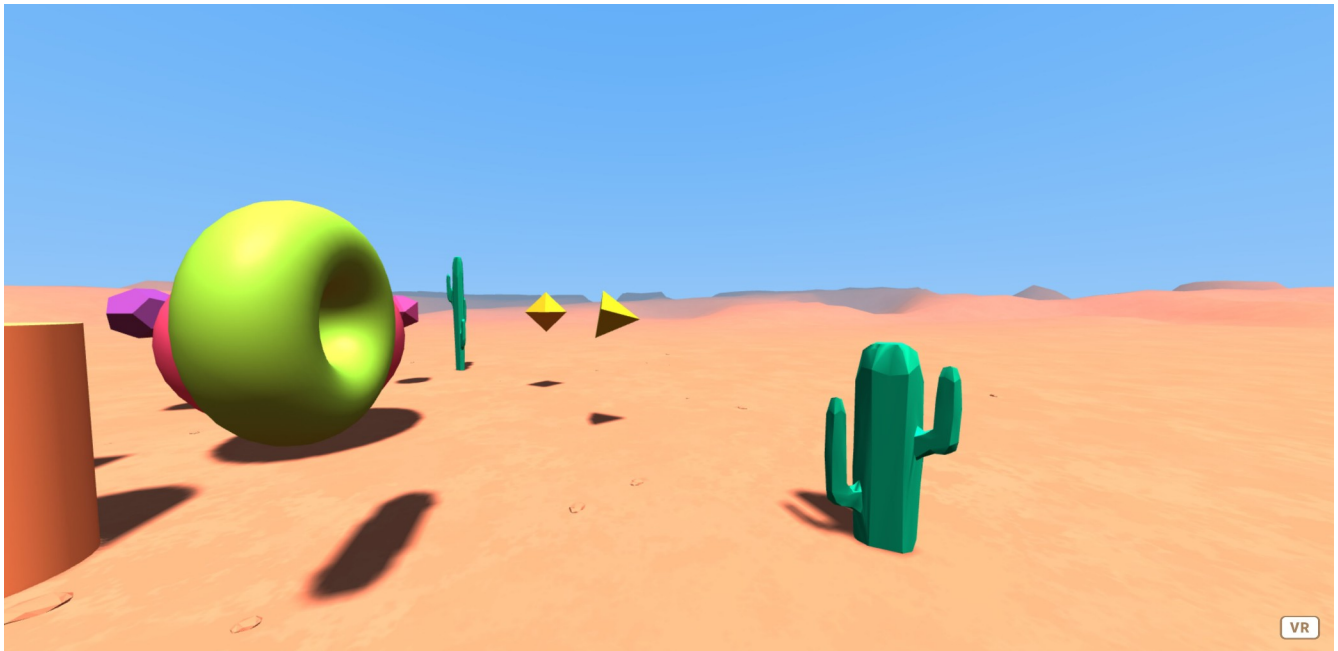
<!-- Place a couple of simple cactus models to enhance the scene -->
<a-entity id="Cactus1" gltf-model="#cactusShort" position="3.43339 -0.29994 -9.49947" scale="2 6 1" shadow="receive: false"></a-entity>
<a-entity id="Cactus2" gltf-model="#cactusTall" position="5.8929 -0.12904 -5.78972" scale="2 2 1" shadow="receive: false"></a-entity>
```

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.

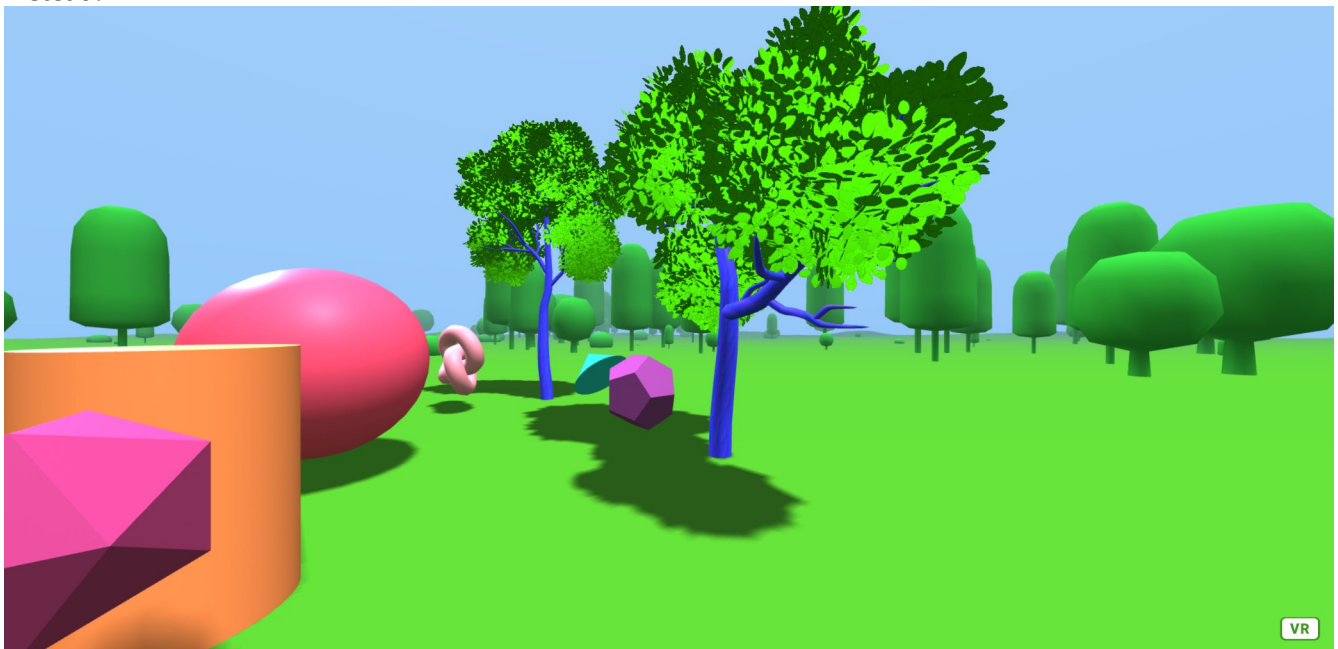
```
<!-- Tree models -3D Gltf models -->
<a-asset-item id="Tree1" src="assets/gltf/NormalTree_1.glb"></a-asset-item>
<a-asset-item id="Tree2" src="assets/gltf/NormalTree_2.glb"></a-asset-item>

</a-assets>
```

```
<!-- Place a couple of simple tree models to enhance the scene -->
<a-entity id="ParkTree1" gltf-model="#Tree1" position="0.95027 -0.29994 -9.81891" scale=".01 .01 .01" shadow="receive: false"></a-entity>
<a-entity id="ParkTree2" gltf-model="#Tree2" position="4.3576 -0.12904 -5.78972" rotation="0 -73.934 0" scale=".01 .01 .01" shadow="receive: false"></a-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](#).

```
<!-- Right Controller Hand -->
<a-entity id="rightHand" hand-controls="hand: right; handModelStyle: lowPoly; color: #15ACCF" laser-controls raycaster="showLine: true; far: 10; interval: 0; objects: .clickable, a-link;"
line="color: #7cfc00; opacity: 0.5" visible="true"></a-entity>
</a-entity>
```

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](#).

```
<body>
<!-- Used for playing our musical tune -->
<audio id="playAudio" autoplay loop
  <!-- If you replace music, make sure it's open source (no copyright), or you have digital rights to it. Credit creator. -->
  <source src="assets/wav/music.wav" type="audio/mpeg">
</audio>

<!-- Used to create a scene, like on the stage of a play, but in 360 -->
<a-scene shadow="type: pcfsoft" renderers="antialias: true; highRefreshRate: true;" shadow="autoUpdate: false" raycaster="objects: .raycastable, .clickable, .cube, .cylinder, .cone, .dodecahedron,
.icosaahedron, .octahedron, .tetrahedron, .torus, .torusKnot, a-link">

<a-assets>
```

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.

```
<!-- Scale was set to half, .5 for each entity and opacity was also set to half, giving each object a transparent see through look -->
<a-entity id="cone" class="clickable" geometry="primitive: cone" rotation="0 45 0" scale=".5 .5 .5" material="color: #1DA7C3; opacity: 1" mixin="spin-x" shadow="receive: false" onclick="disappear('cone');"></a-entity>
<a-entity id="dodecahedron" class="clickable" geometry="primitive: dodecahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #7D347C; opacity: 1" mixin="spin-y" shadow="receive: false" onclick="disappear('dodecahedron');"></a-entity>
<a-entity id="icosahedron" class="clickable" geometry="primitive: icosahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A7346D; opacity: 1" mixin="spin-y" shadow="receive: false" onclick="disappear('icosahedron');"></a-entity>
<a-entity id="octahedron" class="clickable" geometry="primitive: octahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #D19418; opacity: 1" mixin="spin-y" shadow="receive: false" onclick="disappear('octahedron');"></a-entity>
<a-entity id="tetrahedron" class="clickable" geometry="primitive: tetrahedron; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #A48615; opacity: 1" mixin="spin-z" shadow="receive: false" onclick="disappear('tetrahedron');"></a-entity>
<a-entity id="torus" class="clickable" geometry="primitive: torus; radius: 0.75" rotation="0 45 0" scale=".5 .5 .5" material="color: #82832E; opacity: 1" mixin="spin-y" shadow="receive: false" onclick="disappear('torus');"></a-entity>
<a-entity id="torusKnot" class="clickable" geometry="primitive: torusKnot; radius: 0.75; radiusTubular: 0.165" rotation="0 45 0" scale=".5 .5 .5" material="color: #E38097; opacity: 1" mixin="spin-y" shadow="receive: false" onclick="disappear('torusKnot');"></a-entity>
```

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.

```

<script type="text/javascript">
  // Sound effect for objects
  var sparkle = new Audio('assets/mp3/magical_sparkle.mp3');

  // Click listener for do on click
  AFRAME.registerComponent('click-listener', {
    init: function () {
      this.el.addEventListener('click', function (evt) {
        // remove clicked object from view
        //this.setAttribute('visible', false);
      });
    }
  });

  // Handles activating and toggle on-off audio control for browser
  AFRAME.registerComponent('audiohandler', {
    init: function() {
      let playing = false;
      let audio = document.querySelector("#playAudio");
      this.el.addEventListener('click', () => {
        if(!playing) {
          audio.play();
        } else {
          audio.pause();
          audio.currentTime = 0;
        }
        playing = !playing;
      });
    }
  });

  function playSound() {
    //alert("TEST Sound 1 playing functional!!!");
    sparkle.play();
  }

  // Make our selected object invisible by name and play a sound
  function disappear(objectName) {
    document.getElementById(objectName).setAttribute('visible', false);
    sparkle.play();
  }

  // Make all our spinning objects visible again (even if they are still visible), when cylinder is selected, and play a sound
  function reappear() {
    document.getElementById('cone').setAttribute('visible', true);
    document.getElementById('dodecahedron').setAttribute('visible', true);
    document.getElementById('icosahedron').setAttribute('visible', true);
    document.getElementById('octahedron').setAttribute('visible', true);
    document.getElementById('tetrahedron').setAttribute('visible', true);
    document.getElementById('torus').setAttribute('visible', true);
    document.getElementById('torusKnot').setAttribute('visible', true);
    sparkle.play();
  }
}
</script>

```

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.

```
<!-- Music box texture applied to cube -->

```

```
<a-entity id="cube" class="clickable" geometry="primitive: box" position="-1 0.5 -3" rotation="0 45 0" material="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.

```
<!-- camera -also added an animation to fly in for dramatic visual effect at start only -->
<a-entity id="head" camera="active: true" position="0 1.6 0" look-controls="pointerLockEnabled: false" animation="property: position; dur: 15000; from: 0 10 50; to: 0 1.6 0; easing: easeInOutSine; loop: false; direction: normal"></a-entity>
```

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.

```

// Toggle our environment from desertland to parkland, when sphere is selected
function ChgEnv() {

  if (envToggle !== false) {

    document.getElementById('desertland').setAttribute('active', false);
    document.getElementById('desertland').setAttribute('visible', false);
    document.getElementById('parkland').setAttribute('active', true);
    document.getElementById('parkland').setAttribute('visible', true);
    document.getElementById('Cactus1').setAttribute('visible', false);
    document.getElementById('Cactus2').setAttribute('visible', false);
    document.getElementById('ParkTree1').setAttribute('visible', true);
    document.getElementById('ParkTree2').setAttribute('visible', true);

  } else {

    document.getElementById('desertland').setAttribute('active', true);
    document.getElementById('desertland').setAttribute('visible', true);
    document.getElementById('parkland').setAttribute('active', false);
    document.getElementById('parkland').setAttribute('visible', false);
    document.getElementById('Cactus1').setAttribute('visible', true);
    document.getElementById('Cactus2').setAttribute('visible', true);
    document.getElementById('ParkTree1').setAttribute('visible', false);
    document.getElementById('ParkTree2').setAttribute('visible', false);

  }

  swoosh.play();
  envToggle = !envToggle;
}

</script>

```

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



360° camera equirectangular image of me on a San Francisco beach (notice the shadow of the tripods legs are bent)

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

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="X-UA-Compatible" content="IE=edge">
5     <meta http-equiv="content-type" content="text/html; charset=utf-8">
6     <meta name="viewport" content="width=device-width, shrink-to-fit=no, user-scalable=no, maximum-scale=1, minimum-scale=1">
7     <title>360° VR Camera Image</title>
8     <meta name="description" content="This is a 360° VR camera equirectangular image painted on the inside of a sphere."></meta>
9     <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0, shrink-to-fit=no">
10    <meta name="mobile-web-app-capable" content="yes">
11    <meta name="apple-mobile-web-app-capable" content="yes" />
12    <meta name="apple-mobile-web-app-status-bar-style" content="gray-translucent" />
13
14    <!-- A-frame minimized JavaScript version component entity system -->
15    <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
16
17  </head>
18  <body>
19    <a-scene background="color: #FAFAFA" >
20
21      <a-assets>
22
23        
24
25      </a-assets>
26
27      <!-- Set sky to Orb1 -->
28      <a-sky id="orbSky" material="src: #orb1" rotation="0 -90 0" ></a-sky>
29
30    </a-scene>
31  </body>
32 </html>
```

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.](#)

At Rocket Virtual <https://rocketvirtual.com/360VRsphere.html>. In [Glitch 360VRspere.html](#) and [in the browser here](#).

Problems With This Photo



Now I'd like to point out a few problems with this photo for learning purposes. First it's 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 a 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.

At Rocket Virtual <https://rocketvirtual.com/360VRsphere2.html>. In [Glitch 360VRsphere2.html](#) and [in browser here](#).

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.

At Rocket Virtual <https://rocketvirtual.com/360VRsphere3.html>. In [Glitch 360VRsphere3.html](#) and in [browser here](#).

```
15 <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
16
17 <script type="text/javascript">
18
19 // Click listener for do on click
20 AFRAME.registerComponent('click-listener', {
21   init: function () {
22     this.el.addEventListener('click', function (evt) {
23       // remove clicked object from view
24       //this.setAttribute('visible', false);
25     });
26   });
27
28
29
30
31
32 // Handling activating and toggle on-off audio control for browser
33 AFRAME.registerComponent('audiohandler', {
34   init: function() {
35     let playing = false;
36     let audio = document.querySelector("#playAudio");
37     this.el.addEventListener('click', () => {
38
39       if(!playing) {
40         audio.play();
41       } else {
42         audio.pause();
43         audio.currentTime = 0;
44       }
45       playing = !playing;
46     });
47   });
48
49
50 </script>
51
52 </head>
53 <body>
54   <audio id="playAudio" autoplay loop
55     <source src="assets/mp3/oceansound.mp3" type="audio/mpeg">
56   </audio>
57   <a-scene background="color: #FAFAFA" raycaster="objects: .clickable">
58
59     <a-assets>
60
61     
62     
63     
64
65   </a-assets>
66
67   <!-- Set sky to orb1 -->
68   <a-sky id="orb1sky" material="src: #orb1" rotation="0 -90 0" ></a-sky>
69
70   <a-entity id="mouseCursor" cursor="rayorigin: mouse"></a-entity>
71
72   <!-- basic movement in VR with lowpoly hands and raycaster laser pointer for selection -->
73   <a-entity id="cameraRig" movement-controls >
74     <!-- camera - also added an animation to fly in for dramatic visual effect at start only -->
75     <a-entity id="head" camera="active: true" position="0 1.6 0" look-controls="pointerlockenabled: false"></a-entity>
76     <!-- Left Controller Hand -->
77     <a-entity id="leftHand" hand-controls="hand: left; handModelStyle: lowPoly; color: #15ACCF" visible="true"></a-entity>
78     <!-- Right Controller Hand -->
79     <a-entity id="rightHand" hand-controls="hand: right; handModelStyle: lowPoly; color: #15ACCF" laser-controls="showLine: true; far: 30; interval: 0; objects: .clickable" line="color: #7fcf00; opacity: 0.5" visible="true"></a-entity>
80
81   </a-entity>
82
83   <a-entity id="circle" geometry="primitive: circle; radius: 0.75" position="0 -50 0" rotation="-90 0 0" material="src: #sand" shadow="receive: false" scale="33 33 33" opacity=".5"></a-entity>
84   <a-image class="clickable" id="playOcean" width="0.4" height="0.4" src="play" position="-0.015 .5 -4.49178" rotation="0 0 0" audiohandler></a-image>
85
86 </a-scene>
```

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](#).

On Rocket Virtual at <https://rocketvirtual.com/360VRteleport.html>.

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-295e4f9bdf0>



Screenshots from flying drone with 360° camera.

On Rocket Virtual at <https://rocketvirtual.com/360VRteleport2.html>.

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

On Rocket Virtual at <https://rocketvirtual.com/360VRteleport3.html>.

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>

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

<https://rocketvirtual.com/MP/index.html>

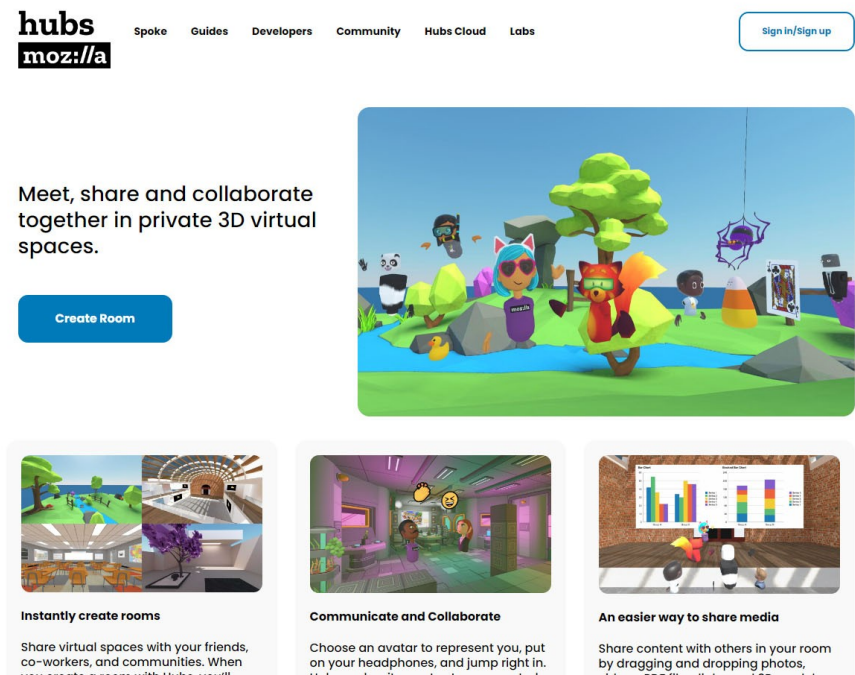
Our next chapter 5 will be about Social VR - Meaning multi-player type experiences in VR with representative Avatars, and with audio talking, etc.

Chapter 5 - Social VR

Up to this point we have been alone inside our VR headset. Meaning no one else is present, except maybe in still photo form. Social VR is about having others experience, usually the same VR experience, as you do simultaneously, like having a real person near you in real reality. Meaning multi-player type experiences in VR with representative Avatars, and with audio talking, etc.

Before we get into how to do this ourselves with actual code, let's first examine some of the current VR Social Platforms that exist at the time of this writing.

In no particular order, [one of them is Mozilla Hubs](#).

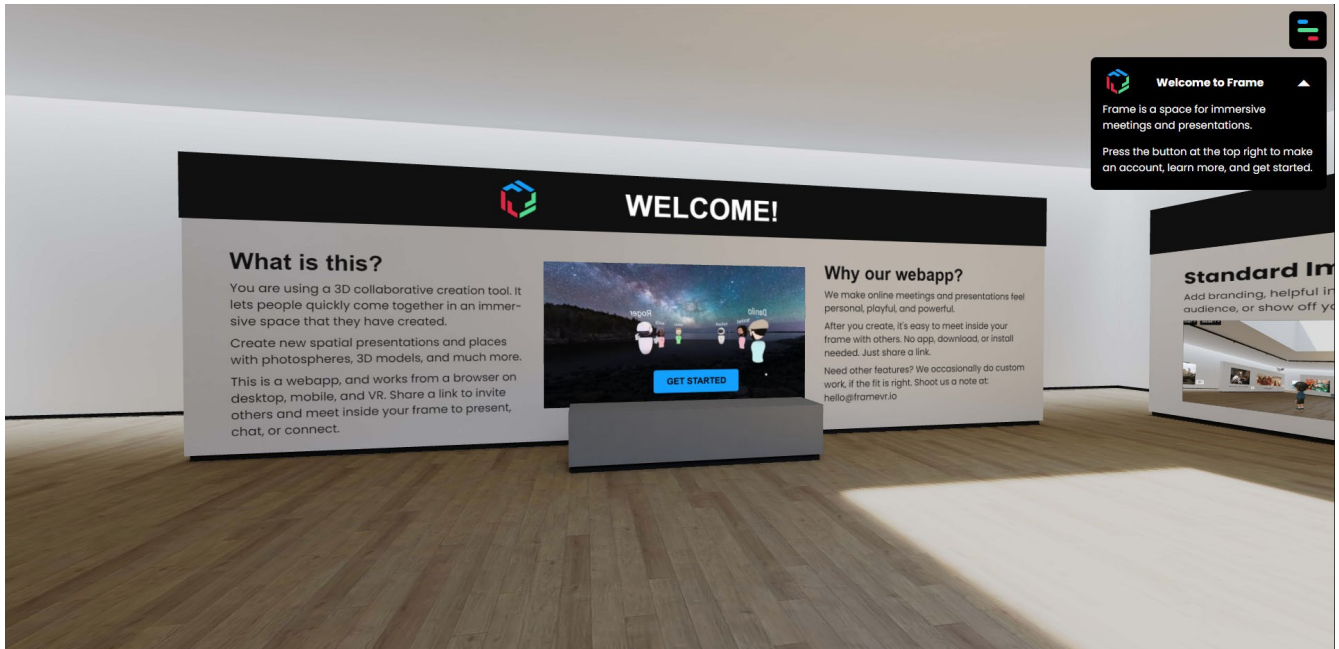


The screenshot shows the Mozilla Hubs website. At the top left is the logo with 'hubs' in white on a black background and 'moz://a' below it. To the right of the logo are navigation links: 'Spoke', 'Guides', 'Developers', 'Community', 'Hubs Cloud', and 'Labs'. Further right is a 'Sign In/Sign up' button. Below the navigation is a large central image of a vibrant 3D virtual world with various avatars and objects. To the left of this image is the text 'Meet, share and collaborate together in private 3D virtual spaces.' and a blue 'Create Room' button. Below the main image are three smaller feature cards: 1. 'Instantly create rooms' with a sub-image of a virtual meeting room and text 'Share virtual spaces with your friends, co-workers, and communities. When...'. 2. 'Communicate and Collaborate' with a sub-image of avatars in a room and text 'Choose an avatar to represent you, put on your headphones, and jump right in...'. 3. 'An easier way to share media' with a sub-image of a presentation screen and text 'Share content with others in your room by dragging and dropping photos...'. The bottom of the page is mostly blank.



[Mozilla Hubs](#) is by the same people who bring you the FireFox browser.

Another one is [Frame VR](#).



Another one is [Vrland.io](#).



Facebook Meta has created one called [Horizon Worlds](#).

Meta Quest

PRODUCTS EXPERIENCES APPS & GAMES SUPPORT

OVERVIEW LEARN COMMUNITY JOIN

COMMUNITY SPOTLIGHT

Keep the party vibes going. Check out these fun worlds from our creators.

The Stu

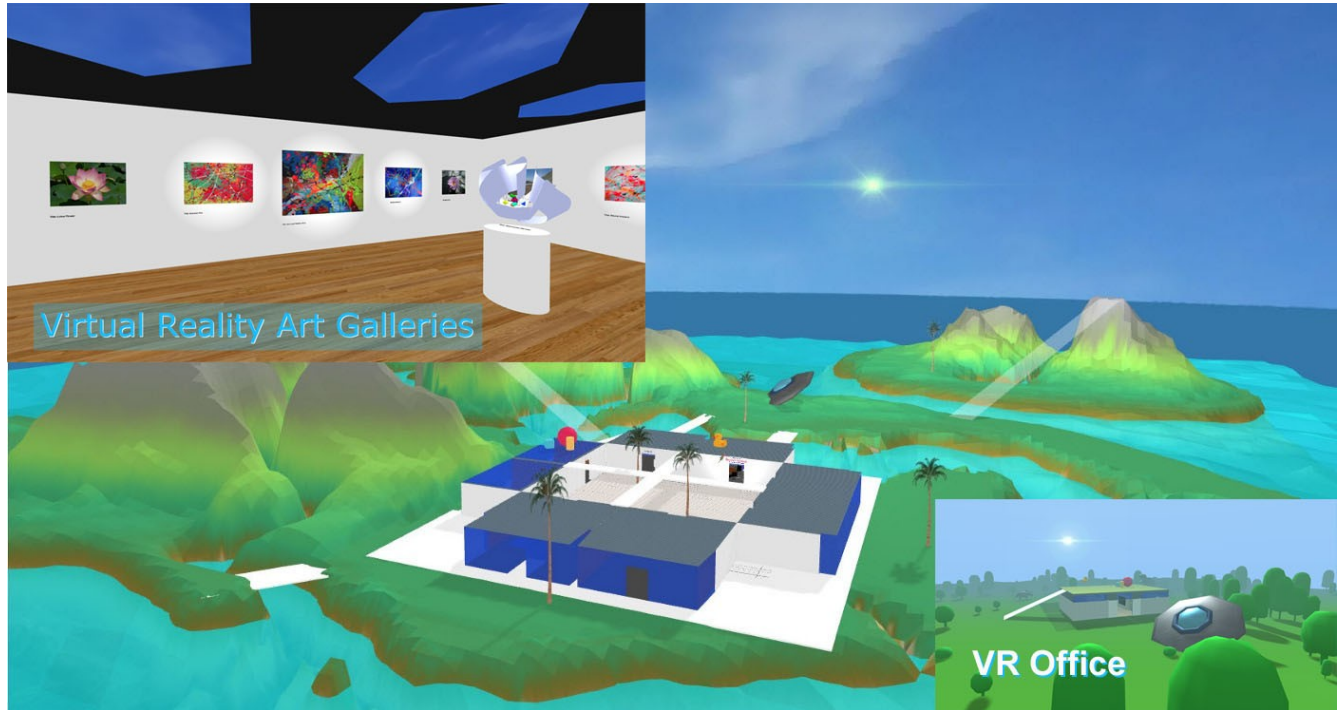
Soapstone Comedy Club

Arena Clash

I'm not going to recommend any specific Social VR Platform over the other because this book is partly about creating a shared VR social space of our own, with code. *Still, there are lots of options out there already, and I believe there can only be more to choose from in the future.*

If you choose to build upon those platforms, that's fine. However, remember one thing, even if they have free hosting options initially, you still have to play by their rules, and with their tools for building out your social VR space. Maybe one other important thing, once you build out on any one specific social VR platform, there is currently no way to transfer your VR content assets to another, except by starting over on that new platform.

Networked A-Frame (NAF)



My Social VR Spaces, Art Gallery, Private Island, VR Office. All self-hosted on my own VR server.

It's not that well known or tried, but there is an A-Frame component called Networked-Aframe (or NAF for short) that works with A-Frame to give multi-user capabilities. It can [be found on GitHub here](#).

I have used earlier versions of it to build out a few VR multi-user spaces (as shown in the image above) and have tested it with up to 5 people simultaneously in VR from separate locations across the internet at the same time. One of them was in England, while I was in California, San Francisco. It works with both VR Headsets and browsers too. So you can experience VR with friends or customers who don't have VR headsets yet.

I will be completely honest with you, if you're not technically inclined, it's not easy or trivial to set up at all. It usually requires your own hosted VR server configuration, either on the internet, or sometimes it can be hosted from your home, on a cable network. But again, be warned, it's tricky to set up.

Your Own VR Server, And Why?

Why? *I'm tempted to say, just because.* Why not host on one of the previously mentioned Social VR Platforms? The up side is of course that you will have access to a larger VR traffic flow through these platforms, than possibly your own. However, some self hosted VR social spaces make sense, such as privately hosted art galleries, your own custom VR office spaces, maybe a custom built VR game. It's really up to your imagination, when you control your own server.

Maybe that's the most important reason, it is for me. I can learn all about how things work behind the scenes, under the hood so to speak. I can tune my own tech, debug it, and admit or ban whomever I want, all with my own VR server storage space for my assets. I can make my own rules, within the legal ones, and those of the hosting service.

If You Want To Setup Your Own Server, What Does That Entail Technically? What Is The Cost?

Before we select a hosting service for our VR server (which is really a web server serving A-Frame enabled web pages, along with NAF) we need to understand that we will need to use something called [Node.js](#).

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the [V8 engine](#) (and executes JavaScript code) outside a web browser, which was designed to build scalable network applications. A scalable architecture is an architecture that can scale up to meet increased work loads, in this case on the server. Also using Node.js on the back-end (server side) in combination with front-end, or browser client side code, provides for a full stack coding solution with the JavaScript language.

So because we need Node.js for [Networked-Aframe](#), we need more than just a normal web hosting service. [We need one that allows us to run Node.js](#). Also A-Frame requires https to access its pages in a browser, so we will also need an [SSL certificate](#) authorized [by a Certificate Authority](#).

Finally, we need a [dedicated IP address](#) assigned to the server, and usually a [domain name](#) for our [DNS](#) entry.

Now that we fully understand our base requirements, we can make an informed decision on choosing a hosting service for our VR server. I personally like [VPS \(Virtual Private Server\)](#) services as opposed to other [cloud hosting services](#) like say AWS.

The reason I like a VPS is because I like more fine grain control over my server setup, and to load and configure all my own services. Besides, I don't need to be greatly scalable for all my visitors (less than say 25, not in the hundreds), for my purposes the VPS will scale out just fine with minor configuration.

[Others will differ in their opinions \(some very valid\)](#), but ultimately that choice is up to you. *You will not have to maintain your own hardware, but now you have the added [responsibilities of a Server Admin](#), as well as the flexibility, and believe me you will learn a lot of valuable skills!*

Now to answer the second question, how much does it cost? Be aware that I am not recommending or endorsing these choices, these are just choices I made, please make your own.

How much it costs really all depends on what hosting options you choose, and their configurations. I went with a low monthly paid solution. [A VPS with a Linux OS loaded, and 2 CPU cores](#). For SSL certificate I used the free [Let's Encrypt](#). For Domain Name, I bought one from [namecheap.com](#).

One other thing you might also want to consider is a hosting facility that is near you physically in the world, unless they are very large, and have geographically dispersed data centers.

The cost would be a fixed \$12 per month hosting for 2 CPUs on a Linux VPS, that's actually cheaper than my Netflix plan subscription. I also have a Windows server, and that's \$20 per month, but much harder to configure with Node.js. [See my article here for a server Windows setup with Node.js](#).

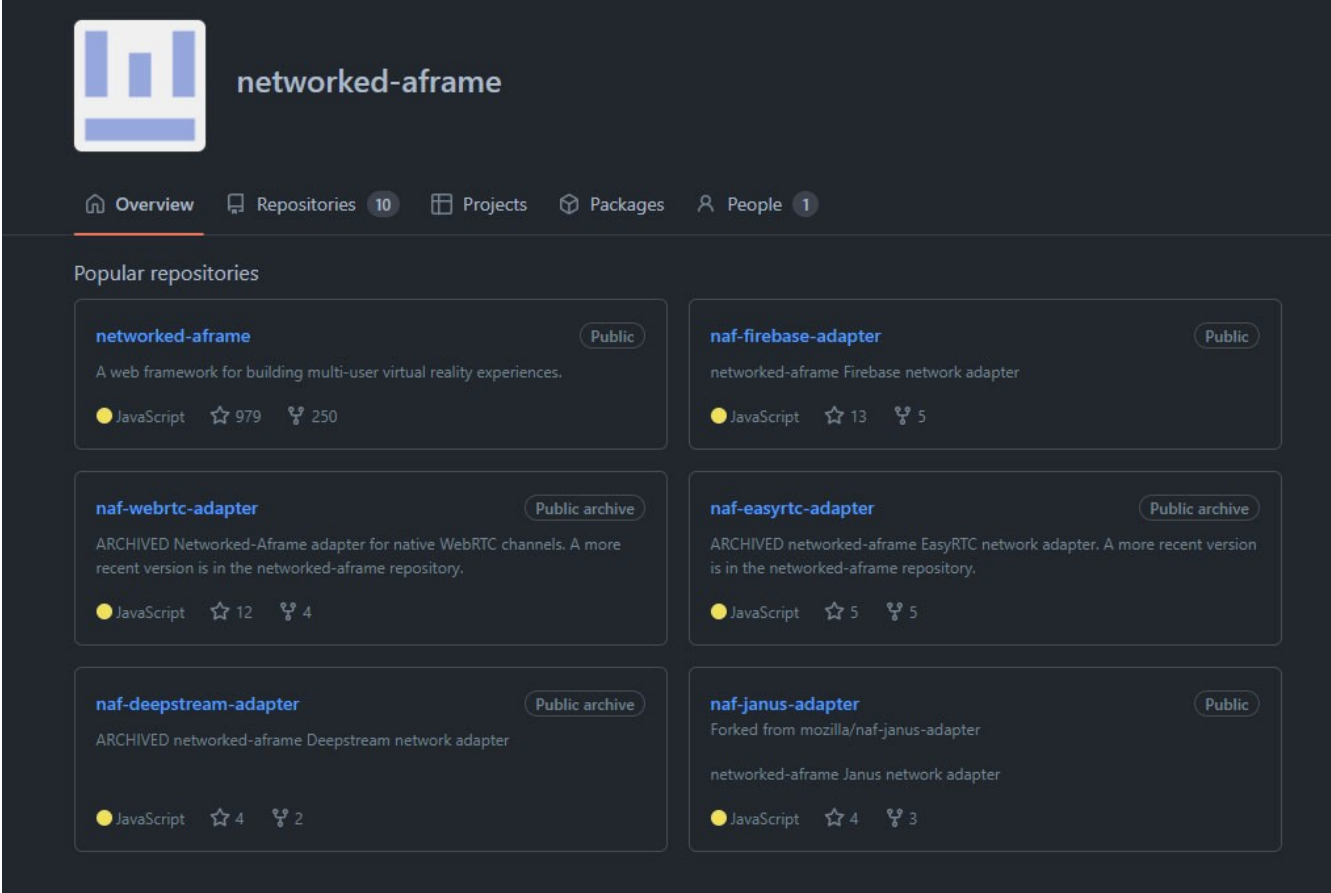
Can you host an alternative cloud service for less than \$12 per month? I don't know. All prices will usually increase from the time of this writing.

WebRTC?

So I need to mention a few more things before we go further. [WebRTC](#) is a real-time communication open standard that supports video, voice, and generic data to be sent between browsers used to build voice, and video communication applications. *Basically this is so powerful that it allows browsers to talk directly to each other over the internet, with or without a server in between them.*

There are two network topologies we can choose from to implement WebRTC on networked-aframe (NAF). [They are Mesh Network or a Star Network. You can find out specific details here](#), but to make it simple, Mesh is from browser-to-browser with no server in between, and Star has a server-between-browser. That may be over simplifying it, but there are advantages and disadvantages to both approaches. We are going to use Mesh to keep things simple in our setup.

Adapters For Networked-Aframe

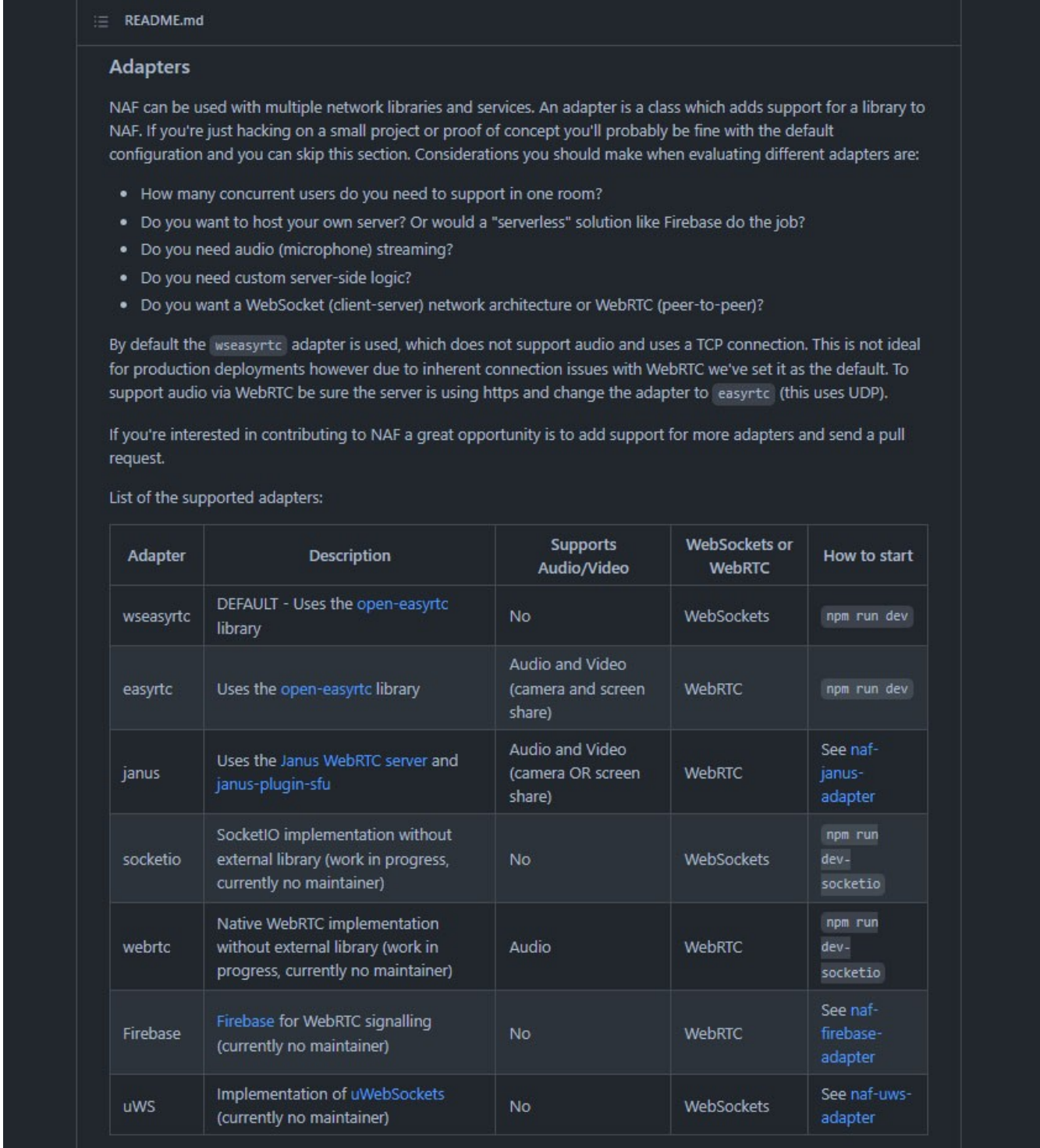


The screenshot shows the GitHub repository page for 'networked-aframe'. The page features a navigation bar with 'Overview', 'Repositories 10', 'Projects', 'Packages', and 'People 1'. Below the navigation bar, there is a section titled 'Popular repositories' which lists several adapters:

- networked-aframe** (Public): A web framework for building multi-user virtual reality experiences. JavaScript, 979 stars, 250 forks.
- naf-firebase-adapter** (Public): networked-aframe Firebase network adapter. JavaScript, 13 stars, 5 forks.
- naf-webrtc-adapter** (Public archive): ARCHIVED Networked-Aframe adapter for native WebRTC channels. A more recent version is in the networked-aframe repository. JavaScript, 12 stars, 4 forks.
- naf-easyrtc-adapter** (Public archive): ARCHIVED networked-aframe EasyRTC network adapter. A more recent version is in the networked-aframe repository. JavaScript, 5 stars, 5 forks.
- naf-deepstream-adapter** (Public archive): ARCHIVED networked-aframe Deepstream network adapter. JavaScript, 4 stars, 2 forks.
- naf-janus-adapter** (Public): Forked from mozilla/naf-janus-adapter. networked-aframe Janus network adapter. JavaScript, 4 stars, 3 forks.

The reason we mentioned network topologies is because NAF allows you to choose which one you are going to use by selecting an adapter. These adapters set up the communication channels over WebRTC for NAF.

In NAF these adapters have these capabilities:



Adapters

NAF can be used with multiple network libraries and services. An adapter is a class which adds support for a library to NAF. If you're just hacking on a small project or proof of concept you'll probably be fine with the default configuration and you can skip this section. Considerations you should make when evaluating different adapters are:

- How many concurrent users do you need to support in one room?
- Do you want to host your own server? Or would a "serverless" solution like Firebase do the job?
- Do you need audio (microphone) streaming?
- Do you need custom server-side logic?
- Do you want a WebSocket (client-server) network architecture or WebRTC (peer-to-peer)?

By default the `wseasyrtc` adapter is used, which does not support audio and uses a TCP connection. This is not ideal for production deployments however due to inherent connection issues with WebRTC we've set it as the default. To support audio via WebRTC be sure the server is using https and change the adapter to `easyrtc` (this uses UDP).

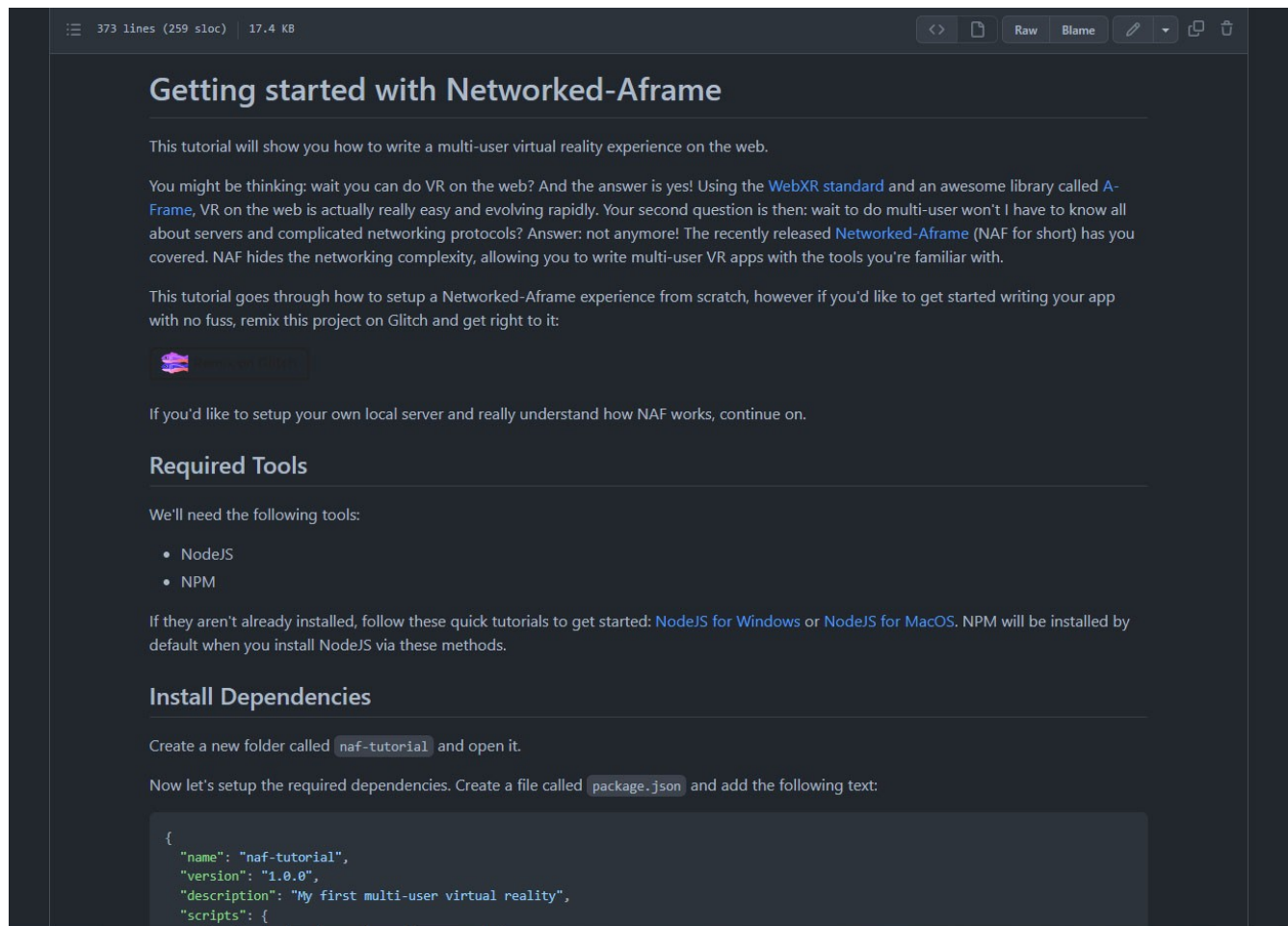
If you're interested in contributing to NAF a great opportunity is to add support for more adapters and send a pull request.

List of the supported adapters:

Adapter	Description	Supports Audio/Video	WebSockets or WebRTC	How to start
wseasyrtc	DEFAULT - Uses the open-easyrtc library	No	WebSockets	<code>npm run dev</code>
easyrtc	Uses the open-easyrtc library	Audio and Video (camera and screen share)	WebRTC	<code>npm run dev</code>
janus	Uses the Janus WebRTC server and janus-plugin-sfu	Audio and Video (camera OR screen share)	WebRTC	See naf-janus-adapter
socketio	SocketIO implementation without external library (work in progress, currently no maintainer)	No	WebSockets	<code>npm run dev-socketio</code>
webrtc	Native WebRTC implementation without external library (work in progress, currently no maintainer)	Audio	WebRTC	<code>npm run dev-socketio</code>
Firebase	Firebase for WebRTC signalling (currently no maintainer)	No	WebRTC	See naf-firebase-adapter
uWS	Implementation of uWebSockets (currently no maintainer)	No	WebSockets	See naf-uws-adapter

We are using the easyrtc adapter which will allow us audio capabilities for our examples, video is possible as well, but we won't be using that here at first. There is a default version of easyrtc adapter configured in the networked-iframe repository which we will use.

NOTE: At this point you might be a little confused. **Why do we need a server at all if we are going to be talking directly between browsers?** The answer to that is, we need a server initially, to at least help find (or discover), and set up the WebRTC communication directly between our browsers, on our mesh network. *Think of it as sort of like an intermediary friend introducing you to each other for the first time. Once you have each other's communication details, you can carry on a conversation with each other directly.*



The screenshot shows a code editor window with a dark theme. At the top, it displays '373 lines (259 sloc) | 17.4 KB'. The main content is a tutorial titled 'Getting started with Networked-Aframe'. The text explains that the tutorial will show how to write a multi-user VR experience on the web, mentioning the WebXR standard and the A-Frame library. It also notes that Networked-Aframe (NAF) simplifies the networking complexity. A small image of a VR headset is shown. Below this, there is a section titled 'Required Tools' which lists NodeJS and NPM. It provides links to tutorials for installing NodeJS on Windows and MacOS. The next section is 'Install Dependencies', which instructs the user to create a folder named 'naf-tutorial' and a file named 'package.json' with the following content:

```
{
  "name": "naf-tutorial",
  "version": "1.0.0",
  "description": "My first multi-user virtual reality",
  "scripts": {
```

Okay, so now we have a basic understanding of how we will use Node.js on our server with NAF, and WebRTC, to communicate in VR with each other's browser. Sounds simple when you say it like that, but now we have to set up our server. Best to glance over and [begin reading the NAF documentation first](#).

My Disclaimer:

I (the book author) of course can not be responsible if your specific server setup doesn't work out for you, for whatever reason. You must take full responsibility for all your decisions, and actions going forward. I have done my best to explain things. If you feel it is too complex a task for your current skill level to achieve successfully, then don't do it. That's completely fine, we are all learning together, best to skip ahead to the next chapter.

Some Assumptions

Some assumptions about my server setup. I have an [Ubuntu 20.04 server version](#) of Linux setup on a VPS as mentioned before. So for my purposes, I am going to use a [LAMP stack](#). This is because I have other things I want to add to this server such as a [MariaDB SQL database](#), [PHP](#), and I'm using an [Apache server](#) to serve up my web pages. You don't need the SQL, PHP and you could use [NGINX](#) instead to server your web pages, or even [IIS on a Windows Server \(more complex to setup with Node.js\)](#), but I'm using Apache on Ubuntu.

We are ready to start. These commands (in this color) are typed at the Ubuntu terminal.

```
sudo apt update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install apache2 mariadb-server-10.3 mariadb-client php libapache2-mod-php php-  
cli php-fpm php-json php-common php-mysql php-zip php-gd php-intl php-mbstring php-curl  
php-xml php-pear php-tidy php-soap php-bcmath php-xmldrpc
```

Install an Ubuntu desktop.

```
sudo apt install tasksel
```

```
sudo tasksel install ubuntu-desktop
```

```
reboot
```

Now we are going to configure a firewall on the server. At a terminal type:

```
sudo ufw app list
```

```
sudo ufw allow 'Apache Full'
```

```
sudo ufw allow 22/tcp
```

```
sudo ufw allow 80/tcp
```

```
sudo ufw allow 443/tcp
```

```
sudo ufw allow 3025/tcp
```

```
sudo ufw allow 3025/udp
```

You may encounter problems with OpenSSH on Ubuntu 22.04 server, if that's the case see this article

<https://linuxhint.com/enable-use-ssh-ubuntu/>

```
sudo ufw allow OpenSSH
```

```
sudo ufw allow from any to any port 3389 proto tcp
```

```
sudo ufw status
```

```
sudo ufw --force enable
```

```
reboot
```

If you need to setup a Remote Desktop into your Ubuntu, then you would issue the following commands and [configure a RDP client to access your Ubuntu desktop](#).

```
sudo apt install xrdp
```

```
sudo systemctl enable --now xrdp
```

Now open a web browser on the server machine in the remote desktop and type 127.0.0.1 in the address bar. You will get the Apache default web page, showing the web server is serving web pages locally.



ubuntu

Apache2 Ubuntu Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Ubuntu systems. It is based on the equivalent page on Debian, from which the Ubuntu Apache packaging is derived. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Ubuntu's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Ubuntu tools. The configuration system is **fully documented in [/usr/share/doc/apache2/README.Debian.gz](#)**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Ubuntu systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|   |-- ports.conf
|-- mods-enabled
|   |-- *.Load
|   |-- *.conf
|-- conf-enabled
|   |-- *.conf
|-- sites-enabled
|   |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`.

Good, the Apache web server is installed, and now working.

sudo mysql_secure_installation

sudo mysql

```
ubuntu@ubuntu: ~
... Success!
Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.
Reload privilege tables now? [Y/n] Y
... Success!
Cleaning up...
All done! If you've completed all of the above steps, your MariaDB
installation should now be secure.
Thanks for using MariaDB!
ubuntu@ubuntu:~$ sudo mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 57
Server version: 10.3.31-MariaDB-0ubuntu0.20.04.1 Ubuntu 20.04
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]> █
```

Inside MariaDB MySQL compatible database

Now you will be at a MariaDB (MySQL) prompt: Congratulations, you now have a database server installed. Setup the root password and write it down so you remember it later. Remove anonymous users, and disallow remote login. Remove the test database. Reload privilege tables. We are not going to issue any database commands at this time.

Type

exit

If you are still in MariaDB. To get out, and back to the terminal. Otherwise no need to type exit.

sudo chown -R www-data:www-data /var/www/

sudo chmod -R 777 /var/www/

cd /var/www/html

ls -l

Now you have ownership of the /html directory where files from the web server are served from to the outside internet.

Okay, let's install one of my favorite graphical text editors. That would be Sublime. Type at the terminal:

sudo snap install sublime-text --classic

reboot

Using the Sublime graphical text editor in the desktop (if it is easier for you, you can also use nano at the command line to text editor throughout this document), create the following file (**text is in this color**):

<?php

phpinfo();

?>

Save that filename as info.php into the /var/www/html directory of the web server.

Now open your browser again and type 127.0.0.1/info.php



PHP Version 7.4.3

System	Linux linuxscrew-host 5.8.0-43-generic #49~20.04.1-Ubuntu SMP Fri Feb 5 09:57:56 UTC 2021 x86_64
Build Date	Oct 6 2020 15:47:56
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-ffi.ini, /etc/php/7.4/apache2/conf.d/20-fileinfo.ini, /etc/php/7.4/apache2/conf.d/20-ftp.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-json.ini, /etc/php/7.4/apache2/conf.d/20-phar.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvsem.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902.NTS
PHP Extension Build	API20190902.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0. Copyright (c) Zend Technologies
with Zend OPcache v7.4.3, Copyright (c), by Zend Technologies



This will give you a PHP Info screen confirming you have a working PHP language.

Type:

```
sudo chmod -R 755 /var/www/
```

```
sudo service apache2 restart
```

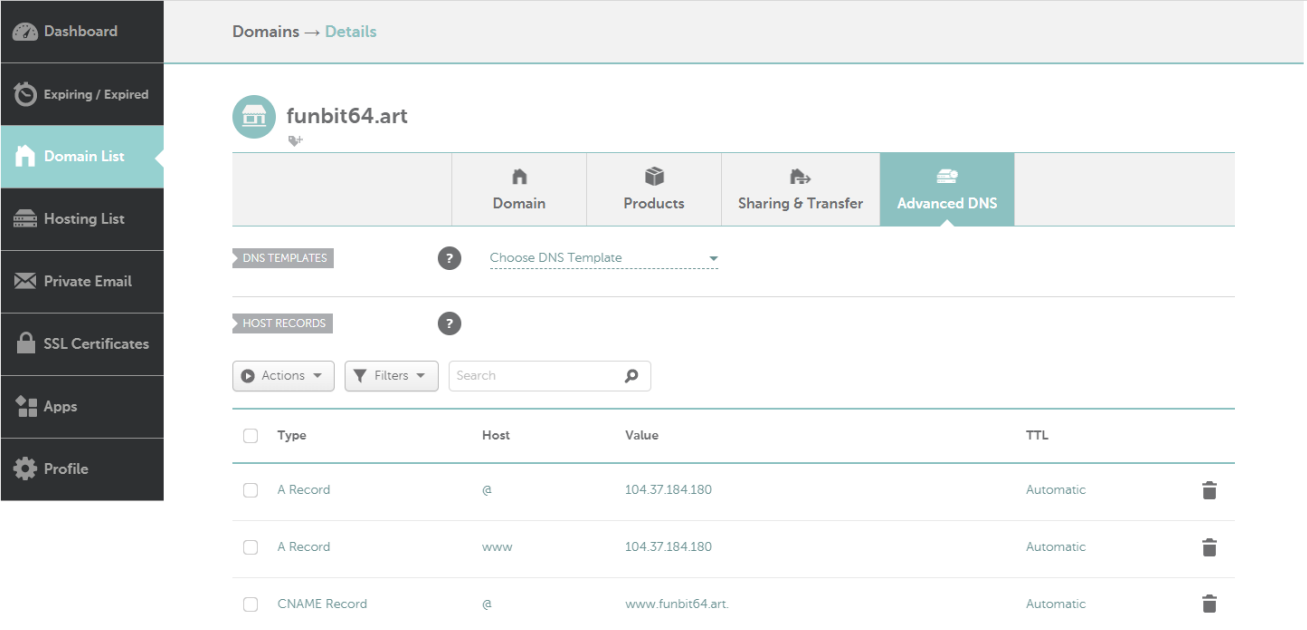
```
sudo systemctl enable apache2
```

We now should have all the rudiments of our LAMP stack installed. Ubuntu (**L**inux), Apache (**W**eb Server), **M**ariaDB (MySQL compatible database server), and **P**HP (programming language).

Purchasing And Setting Up A Domain Name

Let's buy a cheap domain name. I bought one on namecheap.com *I am not affiliated with namecheap, nor do I receive any benefit by mentioning them. You should buy one from wherever you want.*

To make this easy, I will give a screenshot of my managed domain records for the domain I purchased.



The screenshot shows a domain management interface for the domain **funbit64.art**. The left sidebar contains navigation options: Dashboard, Expiring / Expired, Domain List (highlighted), Hosting List, Private Email, SSL Certificates, Apps, and Profile. The main content area is titled "Domains → Details" and features a top navigation bar with tabs for Domain, Products, Sharing & Transfer, and Advanced DNS (selected). Below the tabs, there are sections for "DNS TEMPLATES" (with a "Choose DNS Template" dropdown) and "HOST RECORDS" (with a search bar and filters). The "HOST RECORDS" section displays a table with the following data:

Type	Host	Value	TTL	
<input type="checkbox"/> A Record	@	104.37.184.180	Automatic	
<input type="checkbox"/> A Record	www	104.37.184.180	Automatic	
<input type="checkbox"/> CNAME Record	@	www.funbit64.art.	Automatic	

Notice, I am using the Advanced DNS tab above, and the Add New Record button. One of the things that **may be hard to see in the image above is a . (period) after www.funbit64.art**. On the CNAME record line. **Please make note of it, leaving it out will cause a problem.**

After purchasing, and creating your new domain, search for the manage button. This will allow you to add records to the domain. You will need two A Records, and a CNAME Record:

Creating the records does not mean they will work immediately. DNS propagation delays exist.

Configuring Apache With Your Domain Name And Virtual Hosts

Virtual Hosts should be set up on Apache. Type this at the terminal:

```
sudo chmod -R 777 /etc/apache2/sites-available/
```

Open and edit the following file:

```
/etc/apache2/sites-available/000-default.conf
```

in a sublime editor. Your file should look similar to this, maybe with more comments, etc. Change everywhere there is “your_domain” to the actual domain name you purchased and set up earlier. Making sure there is a ServerName, and a ServerAlias statement.

```
<VirtualHost *:80>
```

```
ServerName your_domain
```

```
ServerAlias www.your_domain
```

```
ServerAdmin webmaster@localhost
```

```
DocumentRoot /var/www/html
```

```
ErrorLog ${APACHE_LOG_DIR}/error.log
```

```
CustomLog ${APACHE_LOG_DIR}/access.log combined
```

```
</VirtualHost>
```

Now save the file.

Now we are going to create another virtual host for our Node.js Networked-Aframe process. Create this file in sublime editor.

Replacing example.com with your domain name

```
<VirtualHost *:80>

    ServerName example.com

    ServerAlias www.example.com

    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/html

    ErrorLog ${APACHE_LOG_DIR}/error.log

    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

and then save it to

```
/etc/apache2/sites-available/
```

under the file name **NAF.com.conf**

Now, at the terminal type

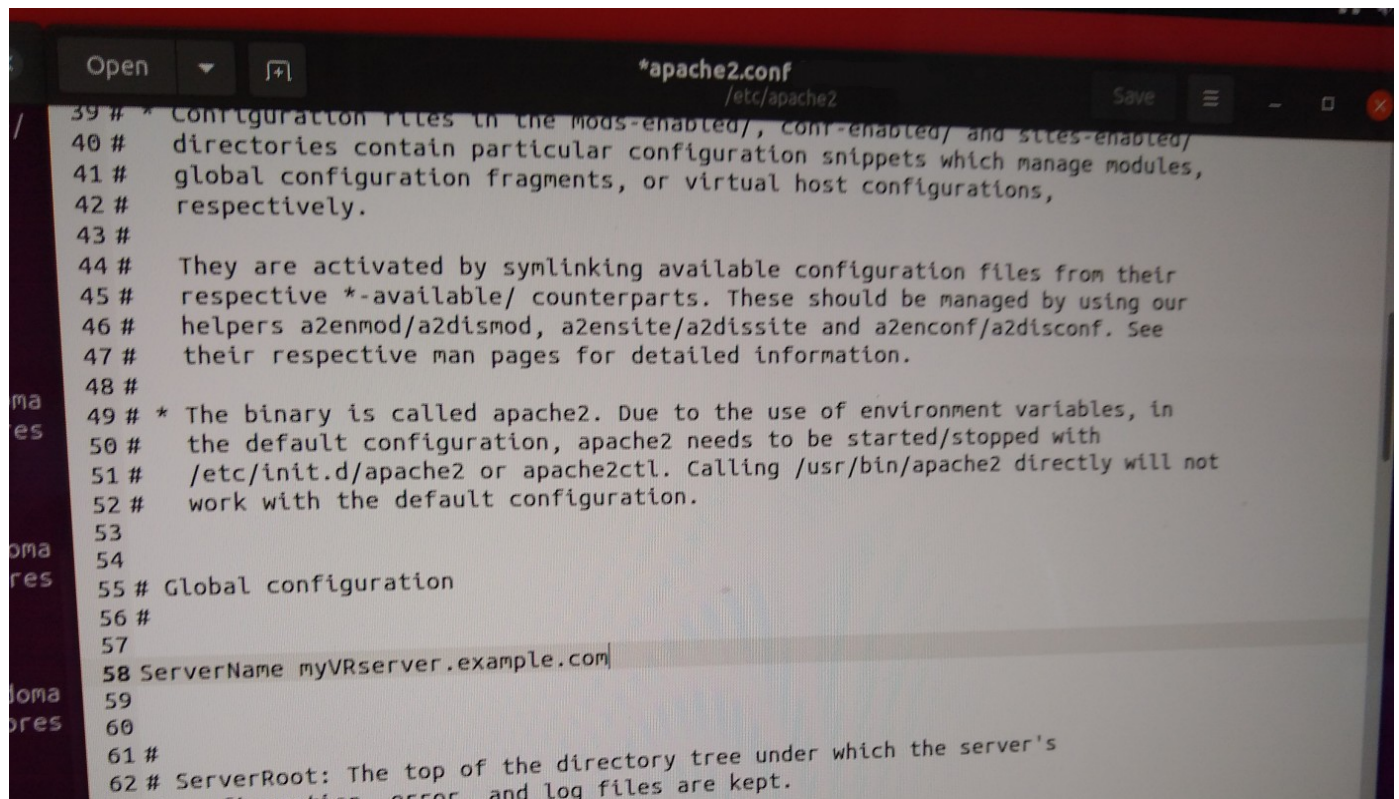
```
sudo chmod -R 777 /etc/apache2/
```

Now we are going to edit the apache2.conf file. It can be found in /etc/apache2/

Open it with sublime text editor and scroll down to find the area labeled Global configuration. Just below, on an uncommented line type:

ServerName myVRserver.example.com

Replacing example.com (see image below) with your domain name, as before. Now save the modified file and close the Text Editor.



We will test for configuration errors next. Should say OK.

sudo apache2ctl configtest

Now we will clean up and restart Apache.

```
sudo chmod -R 755 /etc/apache2/  
sudo a2enmod proxy proxy_http rewrite headers expires  
sudo a2ensite NAF.com.conf  
sudo systemctl restart apache2
```

Installing Node.js as a proxy from Apache with a SSL Certificate

Now we are going to install Node.js with an Apache proxy. Let's just make sure everything is updated first.

```
sudo apt update && sudo apt install nodejs npm  
nodejs -v
```

I had some incompatible version numbers. Instructions in this article and commands below fixed it.

<https://tecadmin.net/how-to-install-nvm-on-ubuntu-22-04/>

```
sudo apt install curl  
curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash  
source ~/.profile  
nvm install-latest-npm  
nvm install node  
nvm ls
```

That will have installed Node, npm, and given you the version number.

Now we are going to get a free SSL Certificate with Let's Encrypt Certbot. Prior to executing Certbot, I should let you know you're going to be asked a number of questions.

However, the most important answers to the questions you will be asked are:

Select both domains listed.

Yes, **2. Redirect**, you want to redirect traffic to the website to HTTPS

Now, at the terminal type:

```
sudo snap install core; sudo snap refresh core
```

```
sudo apt-get install certbot
```

```
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

Ignore if last command above causes File exists error. continue/proceed.

```
sudo apt-get update -y
```

```
sudo apt-get install -y python3-certbot-apache
```

```
sudo certbot --apache
```

That should have installed your SSL Certificate for Apache. You can test your domain on the internet with https:// It will show a lock on the address bar for the Apache default page.

What you have now is a LAMP web server, with SSL enabled, and Node.js installed. Next we are going to load and configure NAF (Networked A-frame) for the “multi-user VR capabilities” functionality next. Good job, so far!

Setting Up Networked A-frame (NAF)

Now we are going to set up Networked-Aframe. This is the magic software which will allow us to “VR talk” across the internet, and do more.

At the terminal, get the networked-aframe from GitHub

```
sudo git clone https://github.com/networked-aframe/networked-aframe.git
```

After NAF is installed, type

```
cd networked-aframe
```

```
sudo npm install
```

```
sudo npm audit fix
```

```
sudo apt-get install -y webpack
```

Under /networked-aframe/server/ find and open the file easyrtc-server.js in sublime editor.

We are going to modify this file to read the SSL certificate configured with the Certbot earlier and reply encrypted to the reverse proxy on port 3025. **Add and modify the file as noted in the color changes below. Also change the YourDomainName.com with your domain name instead.** Now save the file. Make sure in copying and pasting the file, you don’t accidentally include the line numbers from this document.

Https will be used instead of http. This is necessary for the a-frame VR pages to be fully functional.

```

// Load required modules

const https = require("https"); // https server core module

const path = require("path");

const express = require("express"); // web framework external module

const socketIo = require("socket.io"); // web socket external module

const easyrtc = require("open-easyrtc"); // EasyRTC external module

const fs = require('fs'); // file system mode

const options = {

  cert: fs.readFileSync('/etc/letsencrypt/live/YourDomainName.com/cert.pem'),

  key: fs.readFileSync('/etc/letsencrypt/live/YourDomainName.com/privkey.pem')

};

// Set process name

process.title = "networked-aframe-server";

// Get port or default to 3025

const port = process.env.PORT || 3025;

// Setup and configure Express http server.

const app = express();

app.use(express.static(path.resolve(__dirname, "..", "examples")));

// Serve the example and build the bundle in development.

if (process.env.NODE_ENV === "development") {

  const webpackMiddleware = require("webpack-dev-middleware");

```

```

const webpack = require("webpack");

const config = require("../webpack.config");

app.use(

webpackMiddleware(webpack(config), {

publicPath: "/"

})

);

}

// Start Express https server with SSL certificate options

const webServer = https.createServer(options,app);

// Start Socket.io so it attaches itself to Express server

const socketServer = socketIo.listen(webServer, {"log level": 1});

const myIceServers = [

{"urls":"stun:stun1.l.google.com:19302"},

{"urls":"stun:stun2.l.google.com:19302"},

// {

// "urls":"turn:[ADDRESS]:[PORT]",

// "username":"[USERNAME]",

// "credential":"[CREDENTIAL]"

// },

// {

```

```

// "urls":"turn:[ADDRESS]:[PORT][?transport=tcp]",

// "username":"[USERNAME]",

// "credential":"[CREDENTIAL]"

// }

};

easyrtc.setOption("appIceServers", myIceServers);

easyrtc.setOption("logLevel", "debug");

easyrtc.setOption("demosEnable", false);

// Overriding the default easyrtcAuth listener, only so we can directly access its callback

easyrtc.events.on("easyrtcAuth", (socket, easyrtcid, msg, socketCallback, callback) => {

  easyrtc.events.defaultListeners.easyrtcAuth(socket, easyrtcid, msg, socketCallback, (err, connectionObj) => {

    if (err || !msg.msgData || !msg.msgData.credential || !connectionObj) {

      callback(err, connectionObj);

      return;

    }

    connectionObj.setField("credential", msg.msgData.credential, {"isShared":false});

    console.log("[ "+easyrtcid+" ] Credential saved!", connectionObj.getFieldValueSync("credential"));

    callback(err, connectionObj);

  });

});

// To test, lets print the credential to the console for every room join!

```

```

easyrtc.events.on("roomJoin", (connectionObj, roomName, roomParameter, callback) => {

  console.log("[ "+connectionObj.getEasyrtcid()+" ] Credential retrieved!",
connectionObj.getFieldValueSync("credential"));

  easyrtc.events.defaultListeners.roomJoin(connectionObj, roomName, roomParameter, callback);

});

// Start EasyRTC server

easyrtc.listen(app, socketServer, null, (err, rtcRef) => {

  console.log("Initiated");

  rtcRef.events.on("roomCreate", (appObj, creatorConnectionObj, roomName, roomOptions, callback) => {

    console.log("roomCreate fired! Trying to create: " + roomName);

    appObj.events.defaultListeners.roomCreate(appObj, creatorConnectionObj, roomName, roomOptions, callback);

  });

});

// Listen on port

webServer.listen(port, () => {

  console.log("listening on http://localhost:" + port);

});

```

Then type the following, and logout of your session. If you have a different username than ubuntu, change it to that username.

```
sudo chown ubuntu -R /etc/letsencrypt
```


Testing NAF

Now we should be ready to test NAF (Networked-Aframe). At the terminal while in the /networked-aframe directory, type:

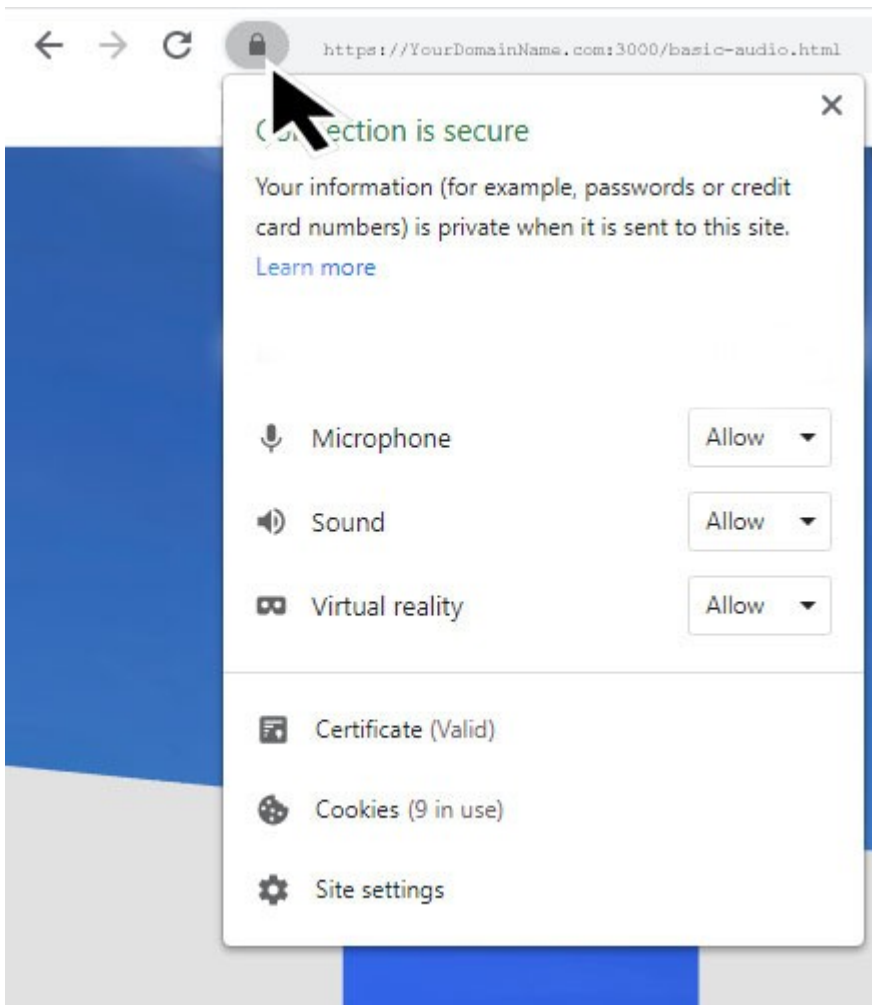
```
cd networked-aframe
```

```
npm start
```

The node NAF process will run and initialize (assuming no errors).

Now, open a browser from a desktop computer, type into the address bar, substituting with your actual domain name (open two tabs with same URL to test):

```
https://YourDomainName.com:3025/basic-audio.html
```



Clicking lock on address bar shows status of allowed browser features in Chrome

Depending on which browser you use to view the web page, the “allow controls” will appear a little different. At this time the only supported browsers are Firefox, Chrome, Edge and Oculus.

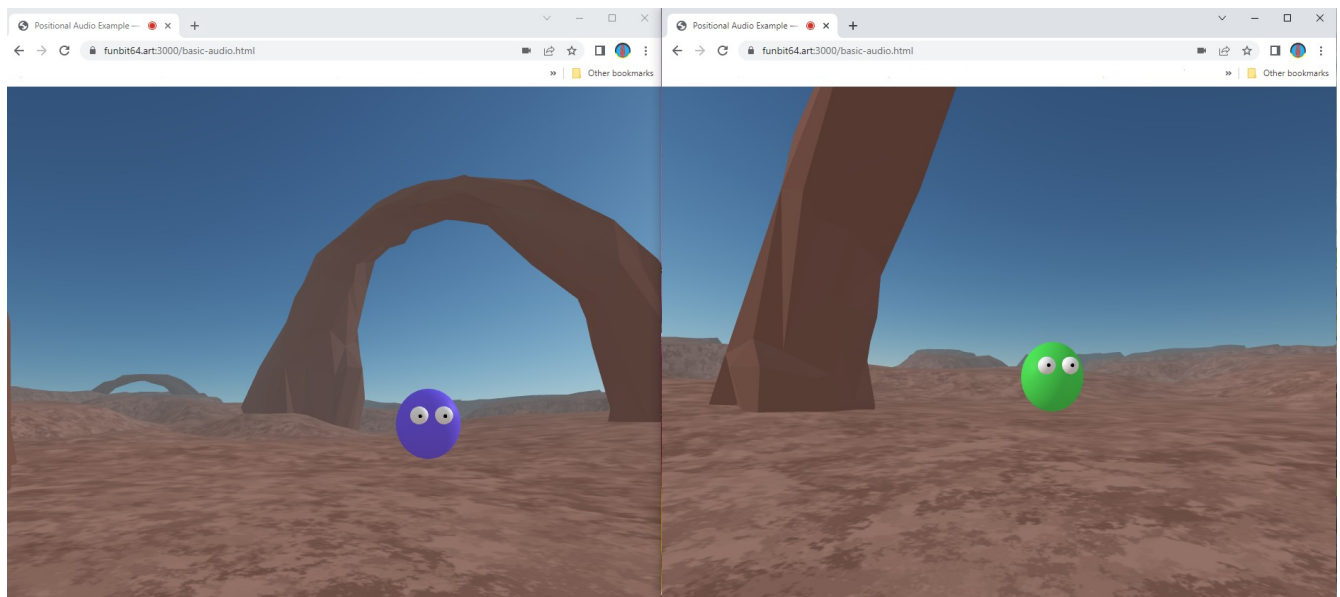
Click on the little lock next to the URL in the address bar in the upper top left of your browser screen to see the microphone and sound settings in the Chrome browser. Allow the microphone and sound. If you have troubles with the Chrome sound enable, some people do, you might want to try using the Firefox or Edge browsers instead.

In Chrome, to enable sound:

1. Click on the three dots upper right corner of Chrome browser
2. Select “Settings”
3. Select “Privacy and security” on the left.
4. Find “Site Settings” in the middle of the screen.
5. Scroll down to find “Additional Content Settings”.
6. Under sound expand and select and click “Sites can play sound”.
7. Make sure Microphone capability can be enabled as well as Virtual Reality.

Microphone/speakers (recommend earphones to eliminate possible feedback) must be available on your computer to hear any speaking audio.

Open another tab with the + and enter the same again. Using the WASD keys on the keyboard, and holding down the left mouse button and dragging (on PC) or on Apple Mac, use command button and mouse/pad (be sure to test with Chrome, or Firefox browser for this).



This will show something like this. Meaning it is working. Say something, and if you have audio enable in the browser, you should hear an echo through the mic. As you move around the corresponding position of the other sphere head will move accordingly.

SFTP

Now that we have a working version of networked-aframe installed and configured on our server, we need a way to get files to the server. Files that we have most likely created on our desktop. One way to do this is secure file transfer or SFTP.

This article explains how to do that in detail.

<https://linuxize.com/post/how-to-setup-ftp-server-with-vsftpd-on-ubuntu-20-04/>

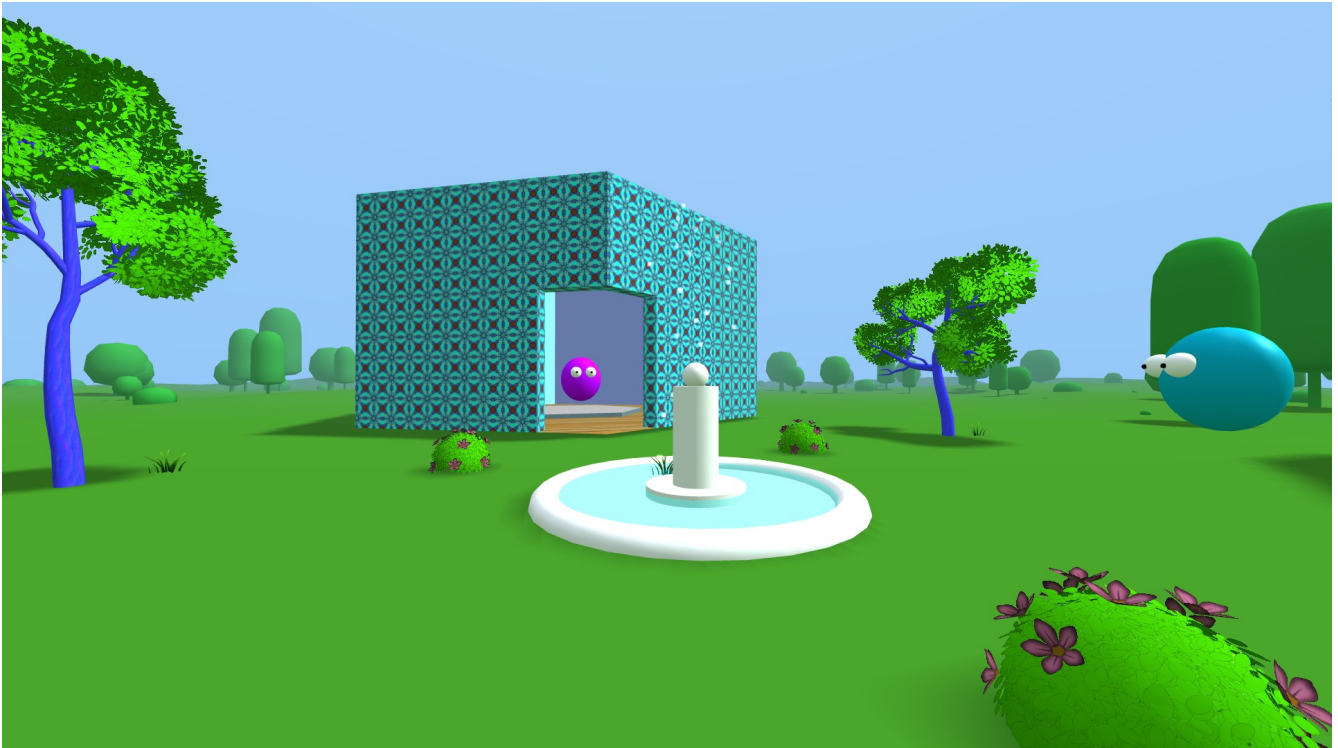
sudo apt update

sudo apt upgrade

sudo apt install vsftpd

sudo systemctl status vsftpd

Chapter 6 – Some Social VR Spaces



We are going to start out this chapter by modifying the `basic-audio.html` file located in the `networked-aframe/examples/` folder. This gives us positional audio in VR, it also creates spherical avatars for us, and positions them randomly within an area of radius. It creates a room which we have changed the name of to `basicNAF` (on line 47 below):

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="content-type" content="text/html; charset=utf-8">
5 <title>Modified Positional Audio Example – Networked-Aframe</title>
6 <meta name="description" content="Basic multi-user Social VR example with positional audio.">
7 <meta name="viewport" content="width=device-width, user-scalable=no, minimum-scale=1.0, maximum-scale=1.0, shrink-to-fit">
8 <meta name="mobile-web-app-capable" content="yes">
9 <meta name="apple-mobile-web-app-capable" content="yes" />
10 <meta name="apple-mobile-web-app-status-bar-style" content="gray-translucent" />
11
12 <script src="https://aframe.io/releases/1.3.0/aframe.min.js"></script>
13 <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.4.0/socket.io.slim.js"></script>
14 <script src="/easyrtc/easyrtc.js"></script>
15 <script src="/dist/networked-aframe.js"></script>
16 <script>
17 // see issue https://github.com/networked-aframe/networked-aframe/issues/267
18 NAF.schemas.getComponentsOriginal = NAF.schemas.getComponents;
19 NAF.schemas.getComponents = (template) => {
20   if (!NAF.schemas.hasTemplate('#avatar-template')) {
21     NAF.schemas.add({
22       template: '#avatar-template',
23       components: [
24         'position',
25         'rotation',
26         {
27           selector: '.head',
28           component: 'material',
29           property: 'color'
30         }
31       ]
32     });
33   }
34   const components = NAF.schemas.getComponentsOriginal(template);
35   return components;
36 };
37 </script>
38 <script src="https://unpkg.com/aframe-randomizer-components@3.0.1/dist/aframe-randomizer-components.min.js"></script>
39 <script src="https://unpkg.com/aframe-environment-component@1.3.1/dist/aframe-environment-component.min.js"></script>
40 <script src="/js/spawn-in-circle.component.js"></script>
41 <link rel="stylesheet" type="text/css" href="/css/style.css" />
42 </head>
43 <body>
44 <a-scene
45   networked-scene="
46   room: basicNAF;
47   debug: true;
48   adapter: easyrtc;
49   audio: true;
50 "
51 >

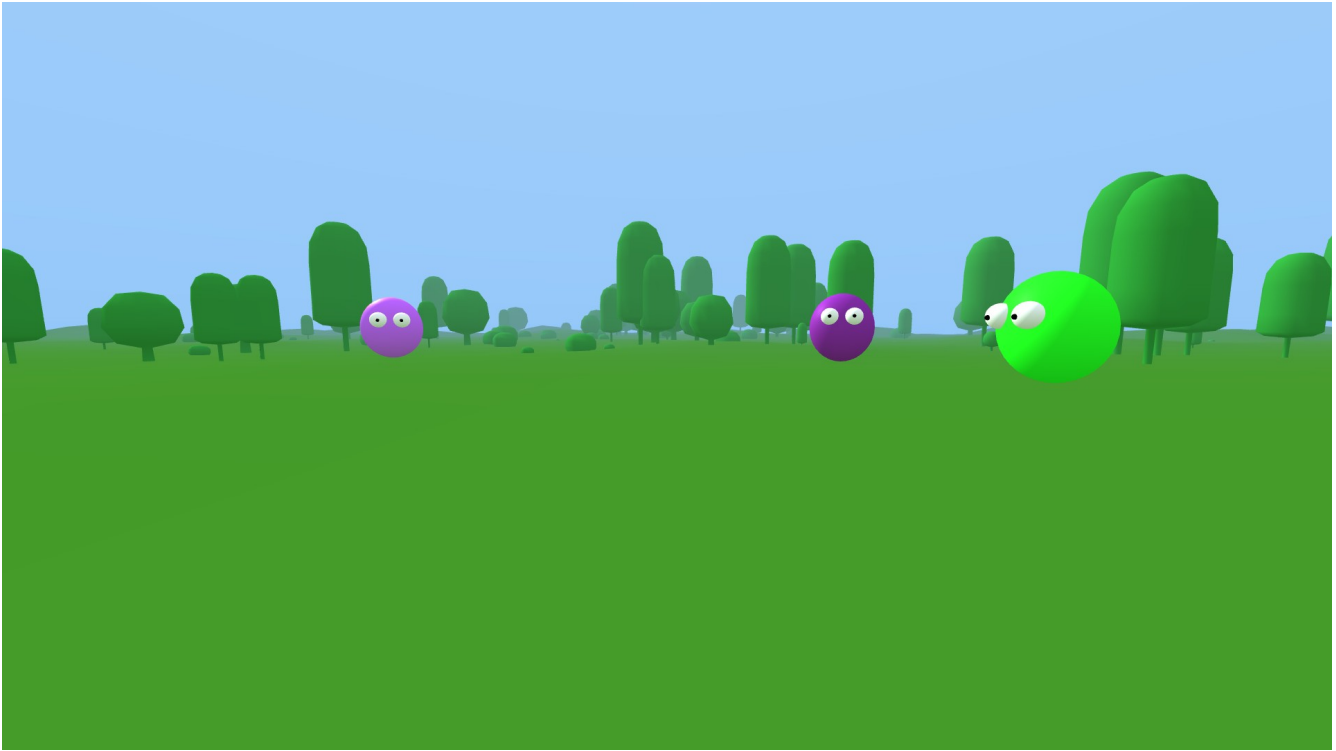
```

Also notice that we are using the easyrtc adapter, and audio is set to true. JavaScript lines 13-15 are needed and necessary to make networked-aframe (*NAF abbreviated*) work. We will discuss lines 17-36 later on.

We added some meta tags at the top which we were using in previous examples.

You can [see the code on GitHub here](#).

[A working example can be found here](#). [Be sure to open multiple tabs in the browser to view more than one spherical avatar in the room](#).



You should see something like this image above. The amazing thing about this is that you have a way to communicate verbally with other persons visiting this same web page over the internet inside of VR! *This is rudimentary Social VR.*

Modifying our example further, we are going to make it look much more interesting by adding a water fountain (made out of basic geometric shapes), some colorful buildings, and a few trees. Nothing we really haven't shown how to do in previous chapters, but now our understanding of things should begin to nicely come all together.

```
<!-- Fix for particle system with aframe 1.3.0 found here https://geeksrepos.com/IdeaSpaceVR/aframe-particle-system-component/issues/66 -->
<script src="js/aframe-particle-system-component.min.js"></script>
```

For the water in the fountain we introduce another component called the aframe-particle-system-component. This component stopped working in A-Frame 1.3.0 and other versions for a while, but we now have a fix. I will be including that in a /js folder directly on the server to stabilize the code fix.

```
<!-- water -->
<a-entity id="water" position="-1.03368 1.629 0.46317" particle-system="color:#000, #FFF;duration:NaN;velocitySpread:2 3 2;velocityValue:0 9 0;accelerationSpread:1 0 1;particleCount:15;maxAge:1.2;texture: https://funbit64.art:3000/assets/img/dot2.png" visible="true"></a-entity>
```

What it gives us is a nice particle animation and the impression of a water fountain with particle droplets spouting out of it. It's not perfect, but if you play soft audio sounds of a water fountain the illusion becomes more believable.

The more quality sensory input you can add to your virtual simulations, both visual, auditory, tactile (say though [haptic feedback](#)) and [in the future olfactory](#), and things like fireplace heat (from a safely

placed external heater), cold, wind blowing (variable speed fan), etc., basically makes the simulation more believable.

[You can find the modified basicNAF3.html on GitHub here.](#) [See it in a browser here.](#) *Click on the spherical ball on top of the tall cylinder inside the fountain to activate the water audio.*

Hexoplex Garden

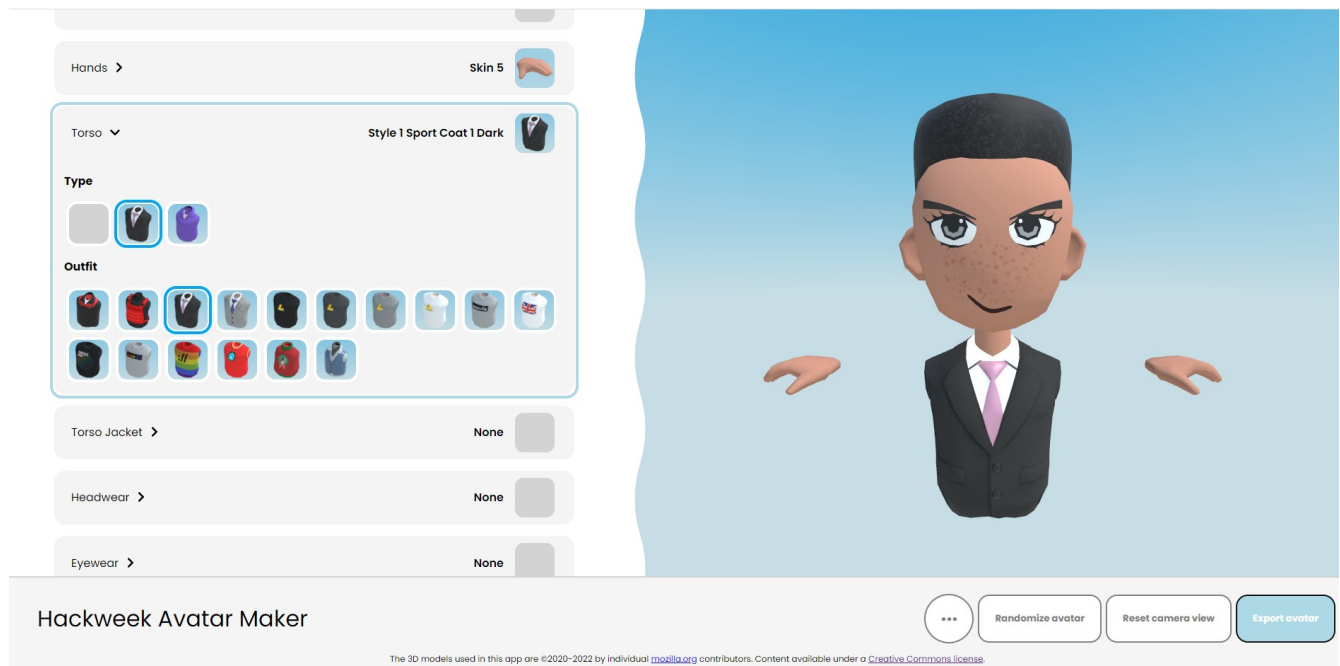
You can use this scene to put a few custom 3D models in the buildings for viewing.

<https://funbit64.art:3025/HexoplexNAF.html>

[SocialVR/HexoplexNAF.html at main · Mike-McAnally/SocialVR \(github.com\)](#)

We are going to change the spherical avatars to custom avatar heads in the social VR scene. *Keep going!*

Avatars

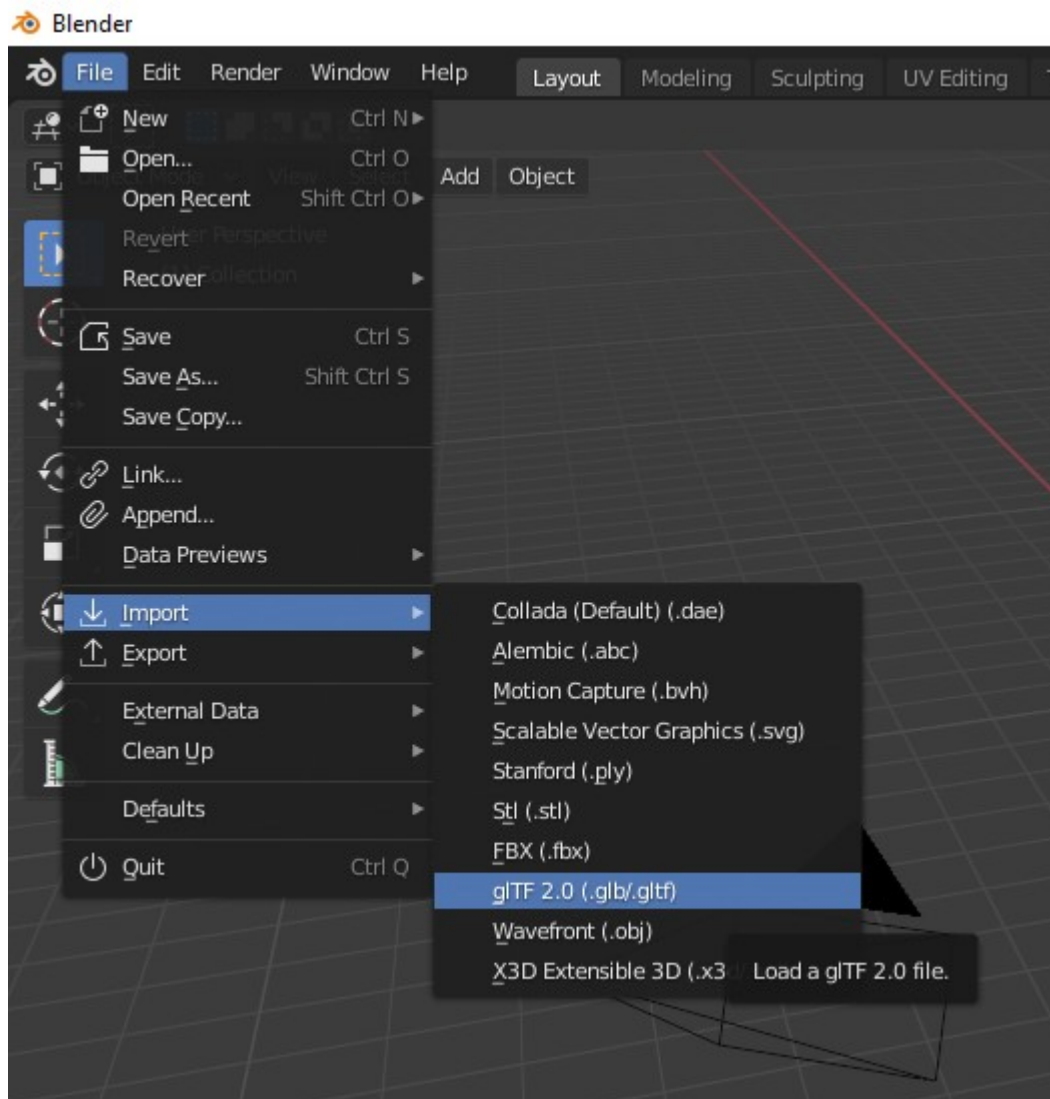


Having customizable avatars, rather than generic spheres, or robot heads just seems like a given for any Social VR platform.

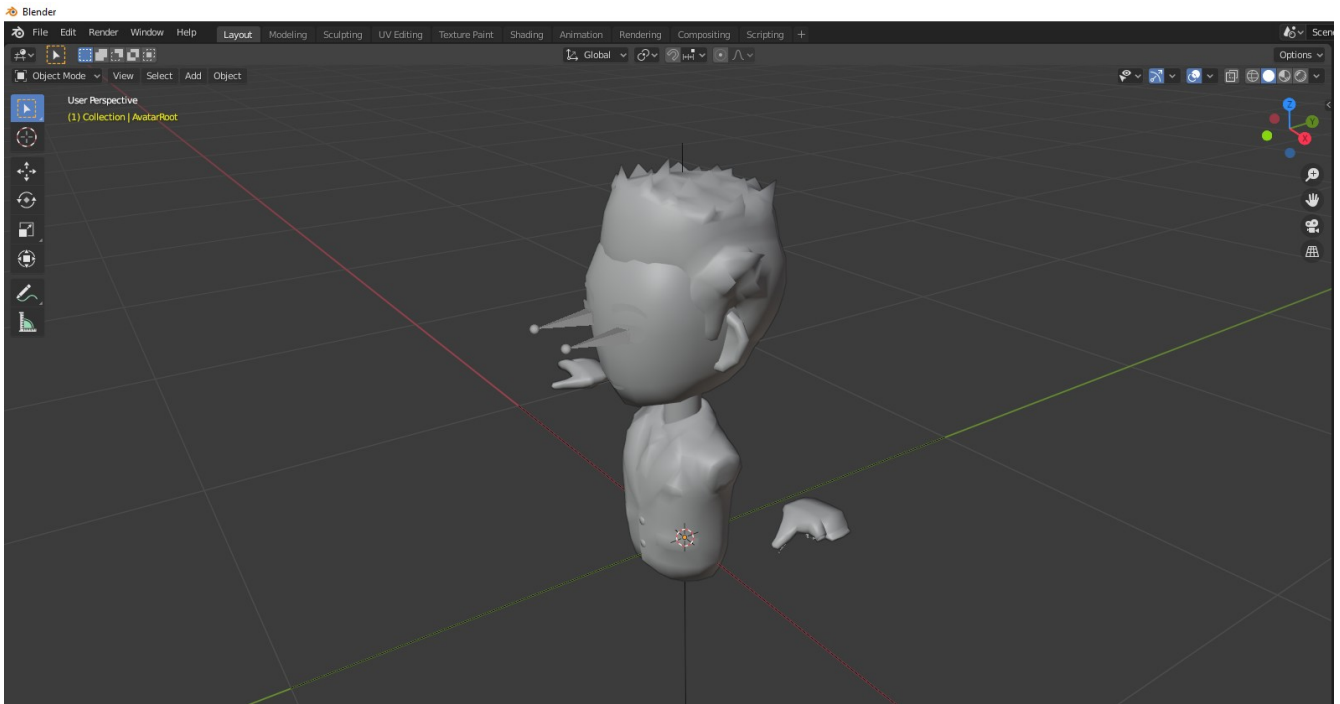
<https://mozilla.github.io/hackweek-avatar-maker/>

Now we are going to attach some avatars made with the avatar generator link above to a few new social VR scenes. So have some fun making a few new avatars, and then export them.

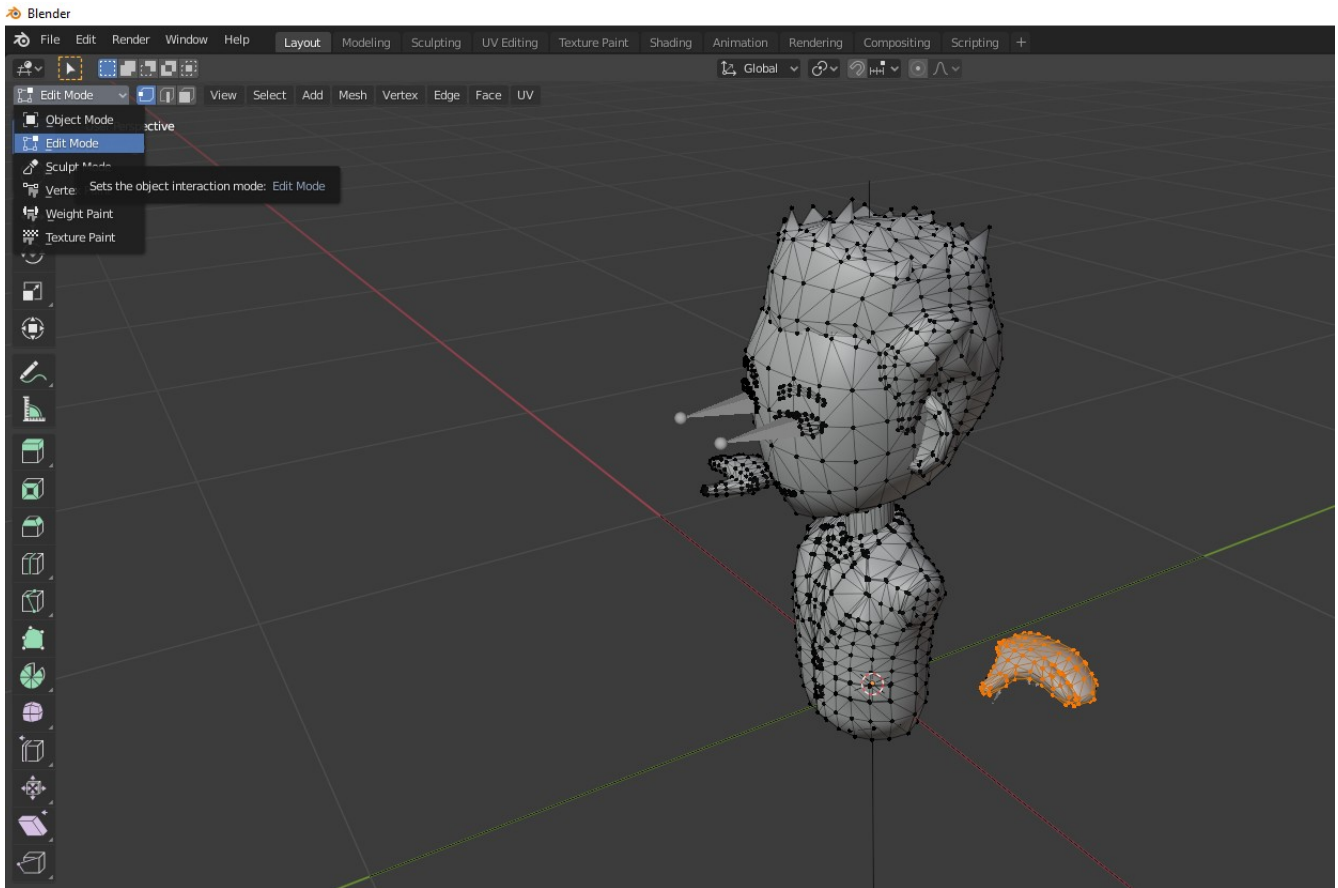
You'll see that they create a custom_avatar.glb file. We are going to edit that files inside of Blender. [So first install blender on your computer if you don't have it already.](#) Then open it, select away the splash logo, then delete the cube with the delete key, now import our custom_avatar.glb file.



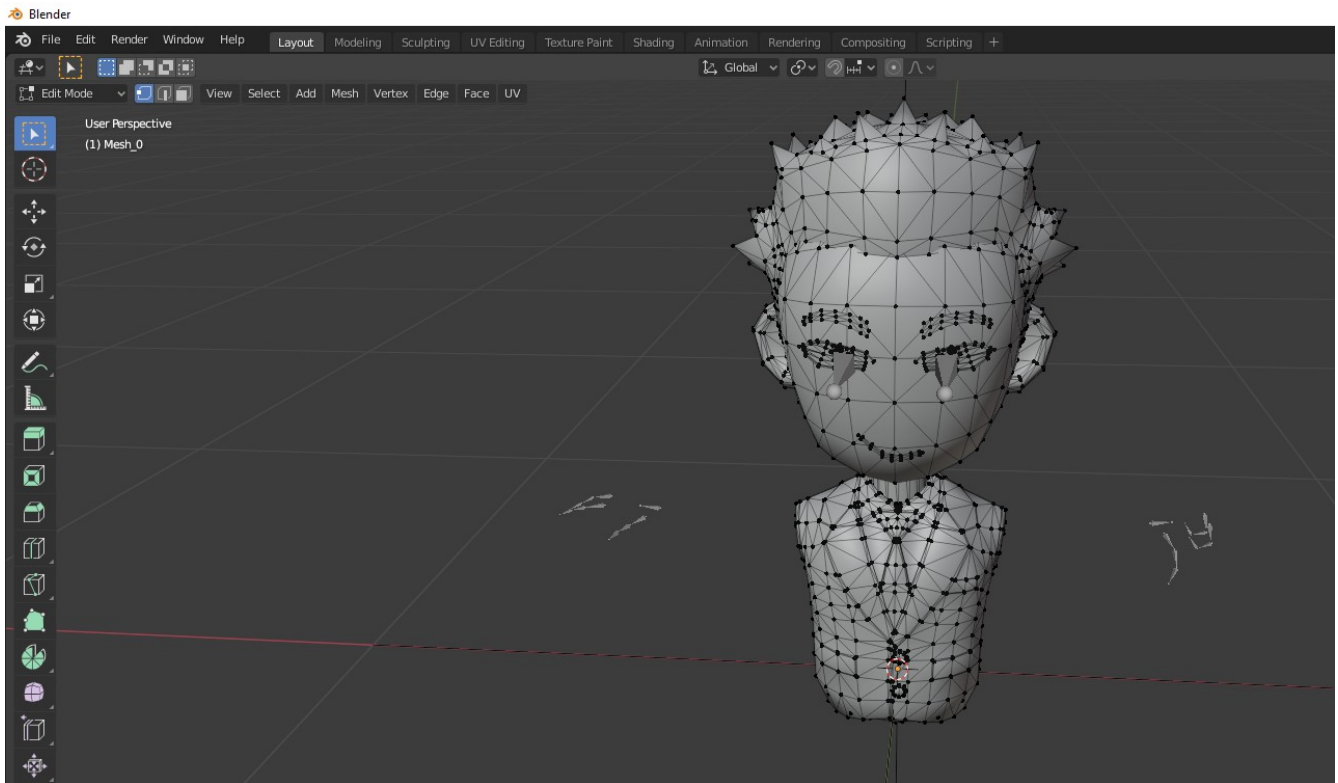
Editing 3D object models with Blender can be a little difficult. [You may need some basic instructions on how to use and get around in blender.](#) You can find that information [here](#).



Select and remove both hands, we are not going to use them from this model.

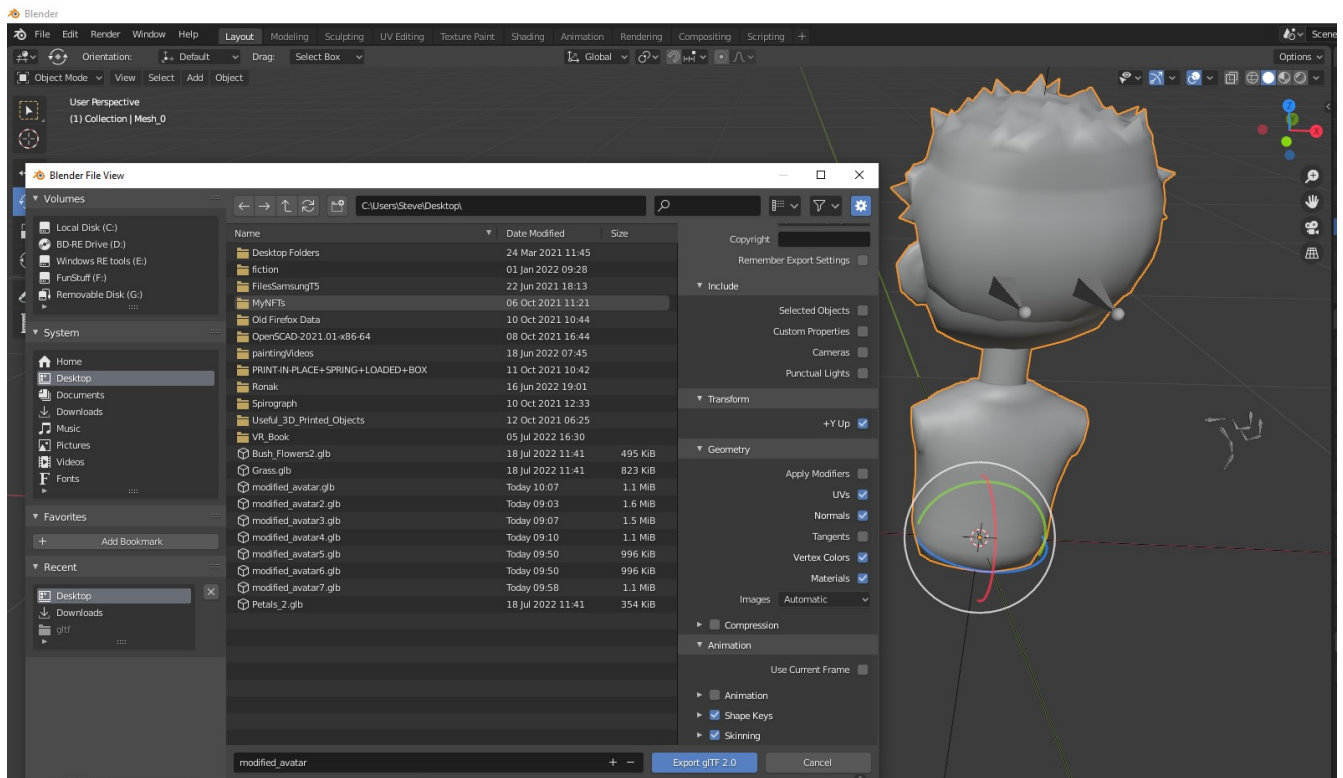


You do that by first selecting the model and then selecting the Edit Mode with the mesh displayed, as in the image above. Then change your view perspective of the avatar 3D model so you are able to select and drag on the hand only. Press the delete key on the keyboard. Then delete the vertices, edges and faces repeatedly until they are all gone. Keep going till all you have left is some skeleton like objects. Don't worry about these, leave them alone for now. Rotate the perspective and delete the other hand the same way.



Now physically rotate the model 180 degrees so its back is facing towards you. Then export the 3D model with these settings:

No animation checkbox.



Deselect the Animation checkbox. This will remove the animations, but also reduce the over all size of the model, causing it to load much quicker.



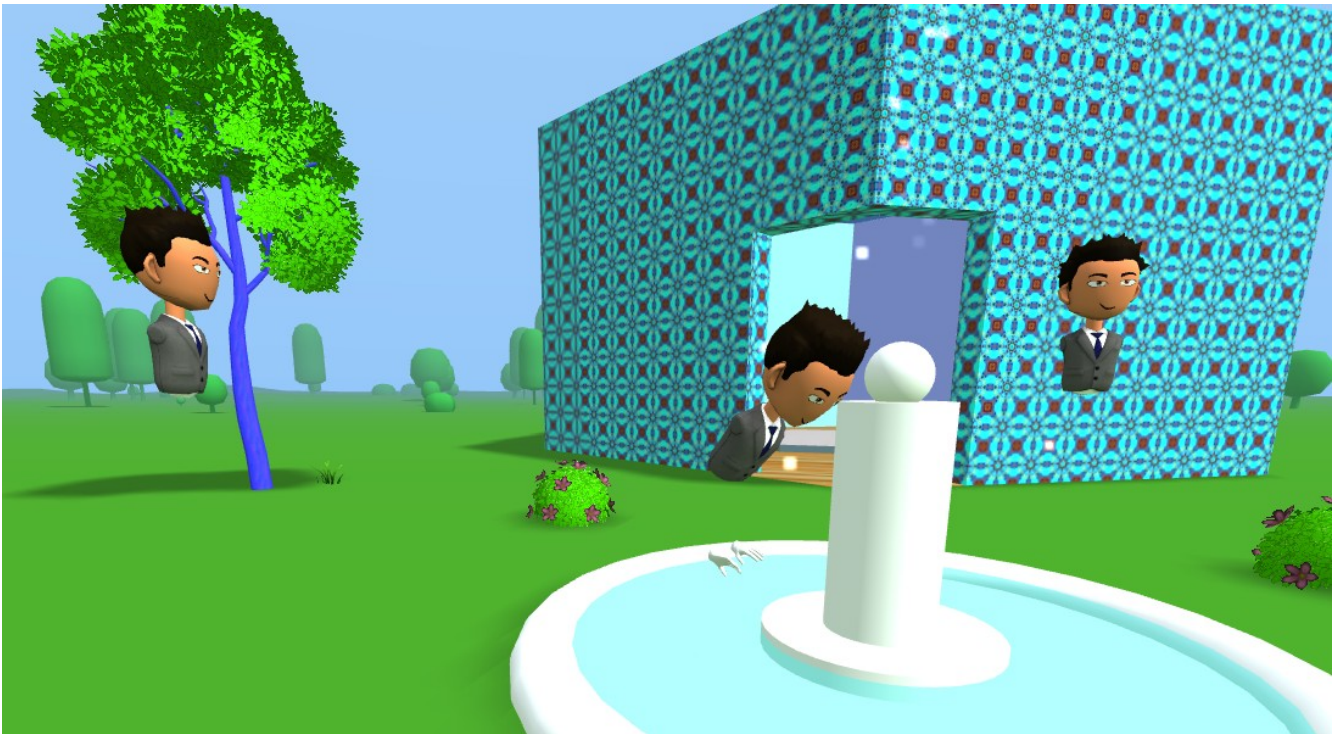
Here is what you get when you apply the same modified avatar to RobotHead in our previous Hexoplex scene. Remember to change the scale = “1 1 1” on the head-template as well to increase avatar model size. See the code snippet below as an example.

```
<!-- Templates -->

<!-- As a substitution for a proper avatar we just use a robot head and hands for now -->
<a-asset-item id="RobotHead" src="assets/gltf/modified_avatar.glb"></a-asset-item>
<a-asset-item id="LHand" src="assets/gltf/leftHandLow.glb"></a-asset-item>
<a-asset-item id="RHand" src="assets/gltf/rightHandLow.glb"></a-asset-item>
<template id="avatar-template">
  <a-entity networked-audio-source></a-entity>
</template>
<template id="head-template">
  <a-entity class="cam" gltf-model="#RobotHead" scale="1 1 1"></a-entity>
</template>
```



So it might seem like our avatars are working out fine when not wearing our VR headset, but once we compare between desktop browser and VR headset you will notice that is not the case.



Our avatar body is a whole head and torso attached, and when we move our head forward and backward inside the VR headset, the whole torso body moves along with the head. This of course doesn't look natural at all, so now we will need to separate the head (*basically amputate it!*) from the torso inside of Blender and save both parts separately. Then to fix this problem, we will need to add the head and torso separately back into our avatar template. *Yes, creating Avatars is much more work than you would think.*



Head only avatars.

Code examples of the [Hexoplex Garden with an attached avatar head can be found here on Github](#). You can attach a selection of different pre-configured avatar heads to other HTML files, which will reference the same social VR room space in NAF, making them all appear in the same space. This will give variation of avatars in the space, and allow for individualization.

Coding up an extension to the Mozilla Avatar Editor which automatically inserts the desired custom avatars into a room is beyond the scope of examples in this book, but can certainly be achieved. You'll also might want to have a login system set up to the server (hence the extra database and PHP installed earlier in Chapter 5), and many other additional features for visitors to your social VR space.

In browser (Chrome) <https://funbit64.com:3025/HexoplexAvatar.html>

Art Room Gallery

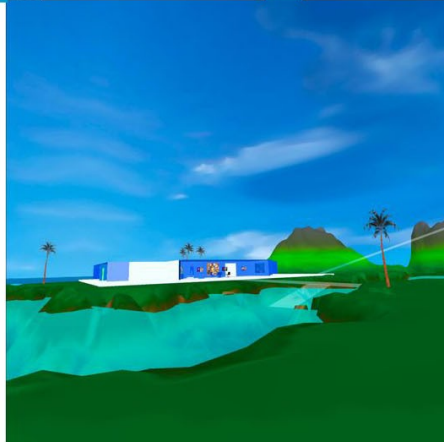
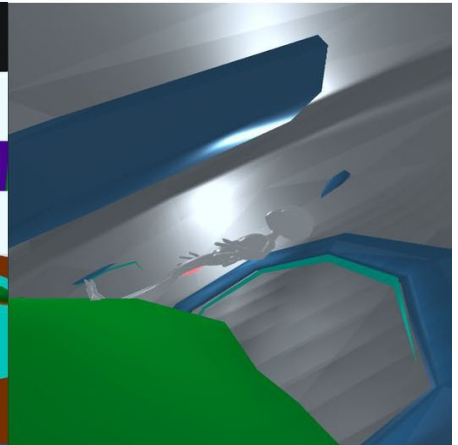
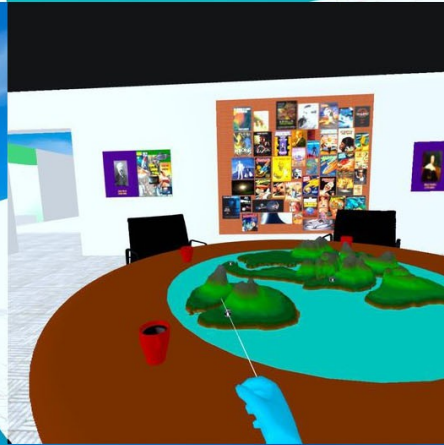
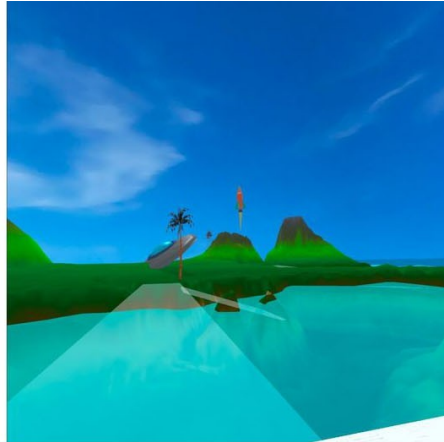


If you are an artist and wish to share and sell your work over the internet, but don't have a physical art gallery to show your artwork in. [Not a problem! Make a virtual reality art gallery. I did.](#) Of course you will be limited to those who visit your web page, and maybe those that have VR headsets alone, but traffic can only increase over time, as more, and more people enter VR to view, buy, and collect art.

[You can see an example of a generic art room gallery on Github here](#) using [open sourced art from the US National Gallery Of Art.](#)

In browser <https://funbit64.com:3025/ArtRoomGalleryNAF.html>

Sci-Fi Themed Island



My VR History

More than a decade and a half ago I created a themed science fiction virtual space on the then very popular Virtual World social VR [platform call Second Life](#). [Later I moved it to another open source](#)

[platform on OSGrid](#) using [Open Simulator](#), and then once again, to my own server setup using [Diva Distro](#) call [Viradu](#).

Now finally, after all this time, I give you a themed Great Science Authors Island VR Social Club Meeting Space that doesn't need a special viewer download installed, just a WebXR enabled browser, which works with most VR headset hardware, and [all the source code can be found on GitHub for free here](#).

You could also re-theme that space as a VR Office Complex on a private island, Club meeting space, Shopping Mall, etc. Currently you can pass through walls on all of these examples, but we will resolve that in our next chapter.

In browser <https://funbit64.com:3025/IslandThemed.html>

Other Avatar Systems

There are other avatar systems besides the one we used in the examples. They are more extensive, with rules and possibly fees. The links below should be of help in further researching.

<https://readyplayer.me/>

<https://docs.readyplayer.me/ready-player-me/integration-guides/web>

<https://avatarsdk.com/>

<https://discord.com/channels/758943204715659264/850840979389153280/994592993510182914>

Personalized Avatars



This is a glTF file version of my custom avatar.

Another avatar type would be a customized avatar from a personal face scan (mine was actually from an MRI machine, [but that's another story](#)). I then processed my medical data through mesh editing software to simplify the number of vertices, and to reduce the overall file size. I replaced my hair because the MRI scan didn't resolve it properly. Also in my case some added texture painting to give skin tone and hair color, which I haven't done just yet. There are also my closed eyelids, which need to be modified, and eyeballs put in! I chose not to go any further with my own custom avatar.

Building your own custom avatar can really be some work, fortunately in most VR use cases you won't need this level of realism, because most people will accept a cartoon-like avatar. That may change in the near future as more avatar uses come into play, specifically with say business VR applications where you might actually meet the person in real reality later. There is also the possibility of [face tracking](#).

[Here is a link for more current information on starting to building your own custom avatar.](#)

<https://vrscout.com/news/create-your-own-vr-avatar-for-the-web-based-metaverse/>

Video Avatars



Screenshot of me as a Video Avatar using an earlier version of Networked-Aframe.

Desktop browser based applications for video conference meetings are useful and very popular when everyone can't be in the same physical space. **If you don't require everyone to have a VR headset to participate**, one of the things the easyrtc adapter for NAF can do is video streaming. You can use a video conferencing camera stream displayed on a cube or other object surface to substitute as an avatar.



Although bandwidth limitations will quickly slow down over an internet connection using a Mesh Network topology; simply because all participants in the browser VR are steaming video to each other simultaneously. The more people video streaming in the same VR space, the more congested the network connection will become. This is usually not a problem if you have a few people participating over the internet or more rarely even if you are using a fast Mbps LAN.

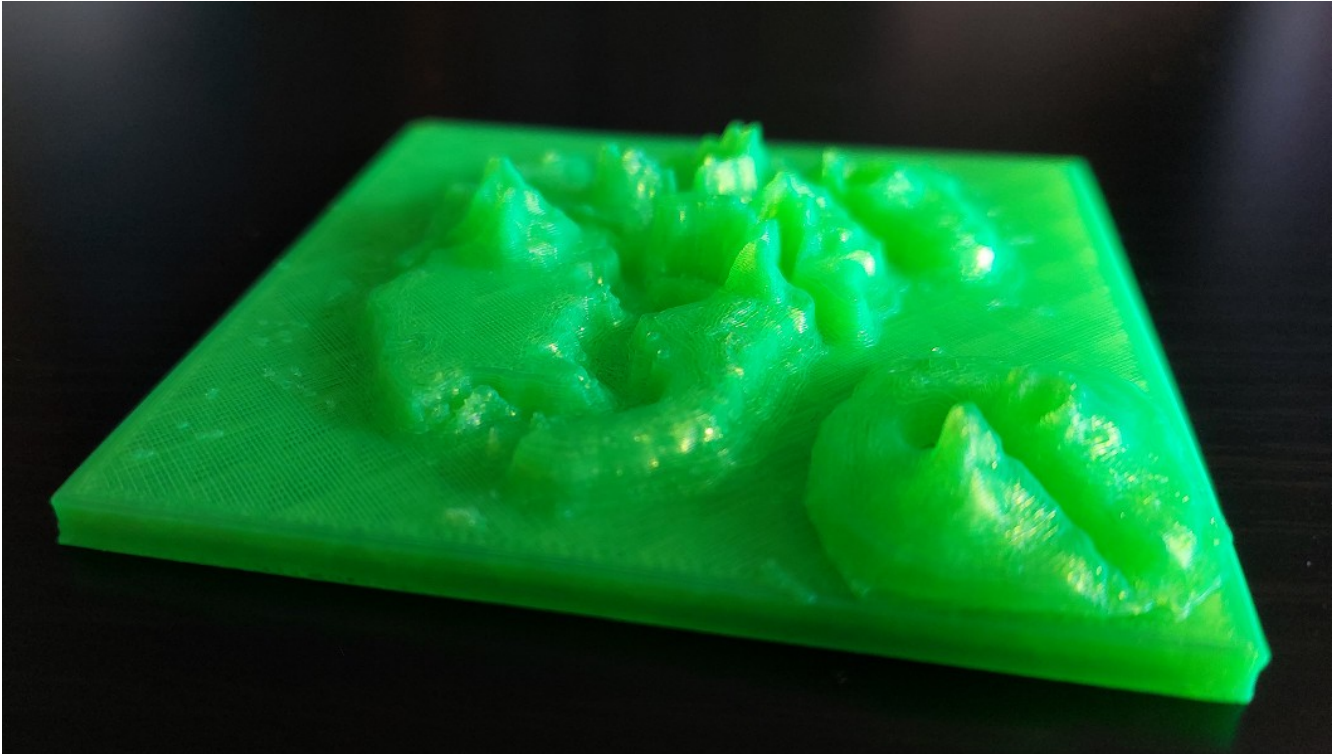
However, by changing network topologies this problem can be mitigated. Switch to a Star Network topology and a dedicated media server to support this. Using the Janus adapter is a good solution for this.

Here is a code example snippet I used to video stream with the easyrtc adapter.

```
134 <a-scene networked-scene="
135   room: ArtIslandComplex2Video;
136   debug: true;
137   adapter: easyrtc;
138   audio: true;
139   video: true;
140 ">
141 <a-assets>
142   <a-asset-item id="optimerBoldFont" src="assets/fonts/optimer_bold.typeface.json"></a-asset-item>
143   
144   <a-asset-item id="MeetingComplex" src="assets/gltf/complexB.glb" nav-agent="speed: 1.5; active:
145     true"></a-asset-item>
146   <a-asset-item id="buildingModel" src="assets/gltf/rooflessModel.glb"></a-asset-item>
147   <a-asset-item crossorigin="anonymous" id="Island" src="assets/gltf/Islands.glb"></a-asset-item>
148   <a-asset-item id="RobotHead" src="assets/gltf/RoboHeadRed.glb"></a-asset-item>
149   <a-asset-item id="Rocket" src="assets/gltf/rocket4.glb"></a-asset-item>
150   <a-asset-item id="Sofa3seat" src="assets/gltf/GreySofa.glb"></a-asset-item>
151   <a-asset-item id="SwivelChair" src="assets/gltf/chairLeatherAluminum.glb"></a-asset-item>
152   <a-asset-item id="Table1" src="assets/gltf/tableFixed14.glb"></a-asset-item>
153   <a-asset-item id="LHand" src="assets/gltf/leftHandLow.glb"></a-asset-item>
154   <a-asset-item id="RHand" src="assets/gltf/rightHandLow.glb"></a-asset-item>
155   
156   
157   
158   
159   
160
161   <template id="avatar-template">
162     <a-entity class="avatar" networked-audio-source>
163       <a-box color="#FFF" position="0 1.4 0" material="" networked-video-source="" geometry=""
164         scale="0.325 0.325 0.325"></a-box>
165     </a-entity>
166   </template>
```

My example is not a perfect solution and could use some more work, but it demonstrates the possibilities. The key things to notice are that video is set to true on line 139, and that networked-video-source is added to the a-box on line 162.

Gifting 3D Printable Assets

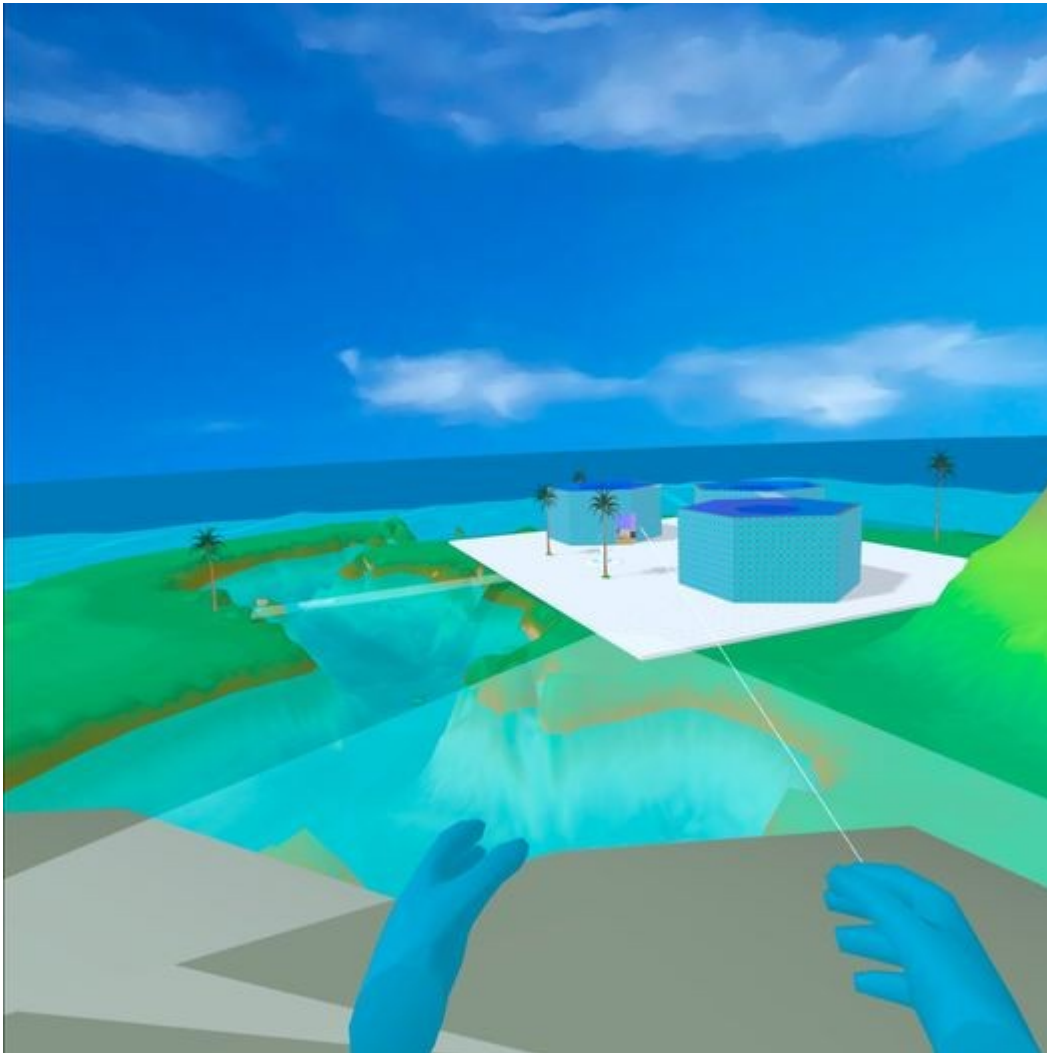


3D printed island topographical model gift

Most of our 3D models are finally converted to the glTF format for loading into VR scenes, but they can also be stored in other formats, [such as STL](#). Above you will see a picture of a 3D printed model of the island used in a previous example.

It makes sense to gift the downloadable model to someone who has a 3D printer as well as a VR headset. Why? Because it makes the virtual reality experience have a real world footprint or presence. Something that is free and physically real, that can be used to advertise, and show your VR scene to others. *Something that you can actually touch, feel, and talk about in the real world!*

Chapter 7 – Bringing It All Together



In this final chapter we are going to bring it all together with some of our previous examples, but it will be more complex, possibly a challenge for some of you, because frankly, it was for me too!

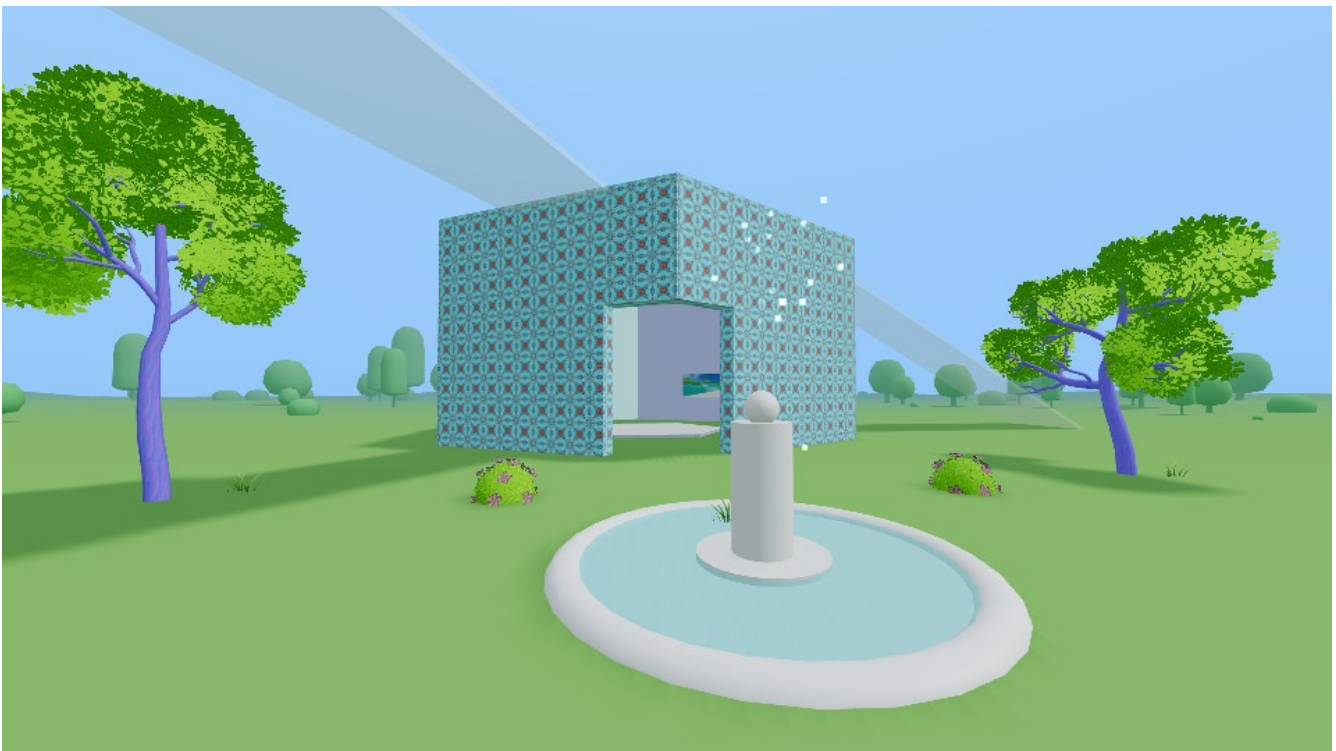
We will be using Simple-Navmesh taken from an open source example here on GitHub (thank you to Ada for this excellent excellent example). Try climbing the latter with your controllers (so cool). *I think [Mario](#).*

<https://github.com/AdaRoseCannon/aframe-xr-boilerplate>

This XR example is very very complex and can be used even as a template for AR, Hand Tracking, and Physics. However, since we only need the simple-navmesh-constraint part, we will put that together with our Hexoplex Garden example in which the walls created by the use of a navmesh can be passed through on a browser desktop or laptop, but not inside the VR headset itself, in which it will be constrained by the glTF file. I also added in the teleport Blink code originally taken from here.

<https://github.com/jure/aframe-blink-controls>

Using the toggle stick, and left trigger, inside a VR headset you can move around and teleport. Try going up the ramp to the roofs of the complex, something again you won't be able to do without a VR headset (*sorry, I just couldn't get that part to work effectively to my satisfaction, along with the teleport, and without a toggle panel selection, used on the left controller in the XR example, which I thought was a bit too cumbersome myself, to quick and easy navigation, so I just didn't include it in the example*).



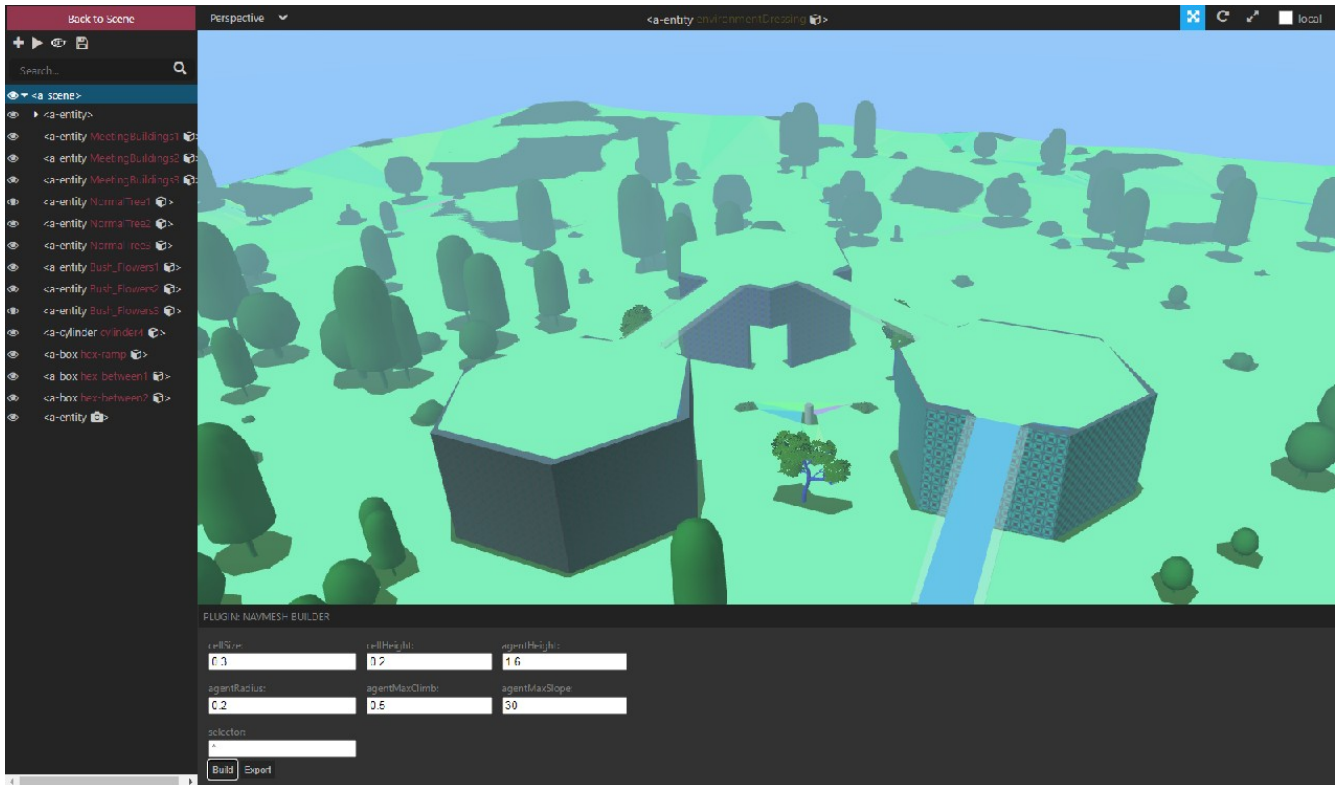
Although Social VR is enabled with this networked-aframe example link below. You could add network-aframe to the boilerplate XR example above as well.

<https://funbit64.com:3025/GardenXR.html>

I also added links to other projects using code snippet inside one of the hex buildings as well as video I shot with my phone at a Maker Faire here in the California Bay Area a number of years back.

Now the link below explains how I built the glTF file for the navmesh.

<https://github.com/donmccurdy/aframe-inspector-plugin-recast>



To do that I had to strip almost ever thing out of my Hexoplex HTML example after adding in the recast JavaScript plugin and putting the component on the <scene inspector-plugin-recast>, then enter the a-frame inspector. Here is a link to what I had to do to build the navmesh and export it.

<https://funbit64.com:3025/GardenXRnavmesh.html>

<https://github.com/Mike-McAnally/SocialVR/blob/main/GardenXRnavmesh.html>

You will need to strip everything related to networked-aframe, only the things you don't want persons to walk through. If you have problems try removing things, and changing the parameters of the build

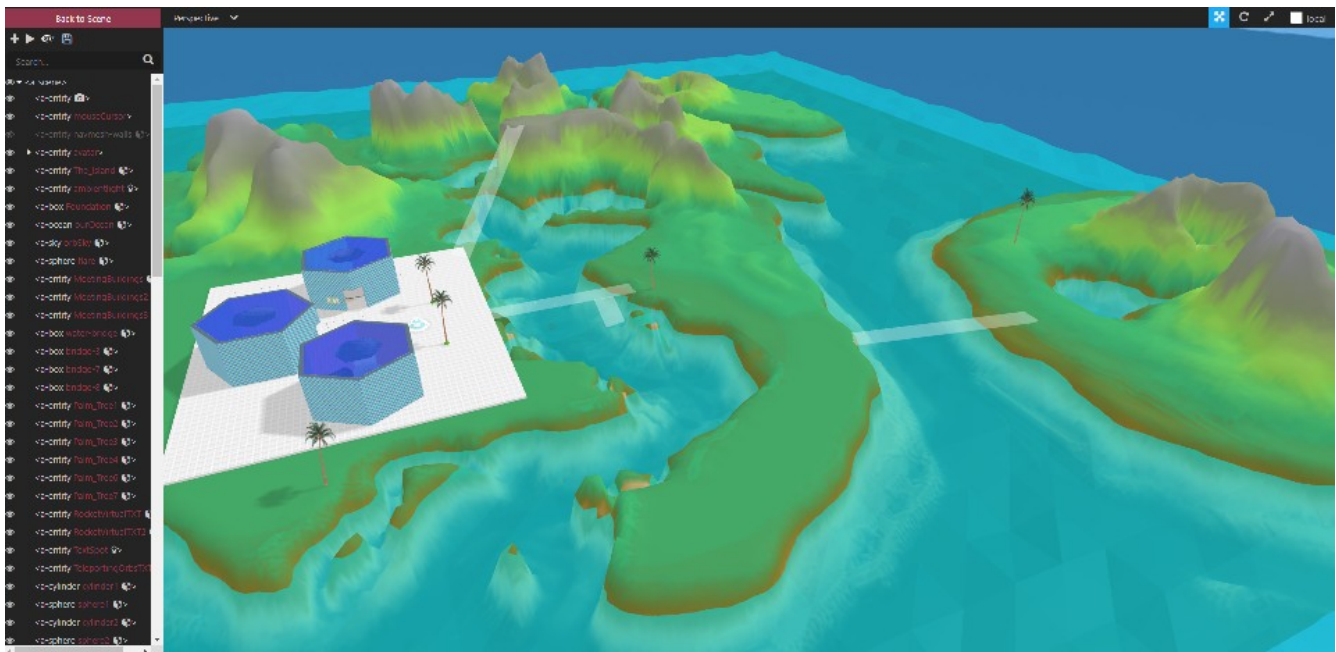
until it works! I had problems with my meeting table, so I removed it. If I really wanted to stop people from walking through it, I would have to replace it with a simpler entity like a larger cylinder which takes up the same space as the table. Most likely the resulting navmesh would then allow people to teleport on top of the table top.

Ultimately if you need to edit the navmesh export, you should be able to do that in Blender. I can think of ways you can create secret doorways and passages into mountain caves and other things with my island example. Navmesh allows for some creativity as shown in the ramps to the roof tops of the Hex buildings in this example.

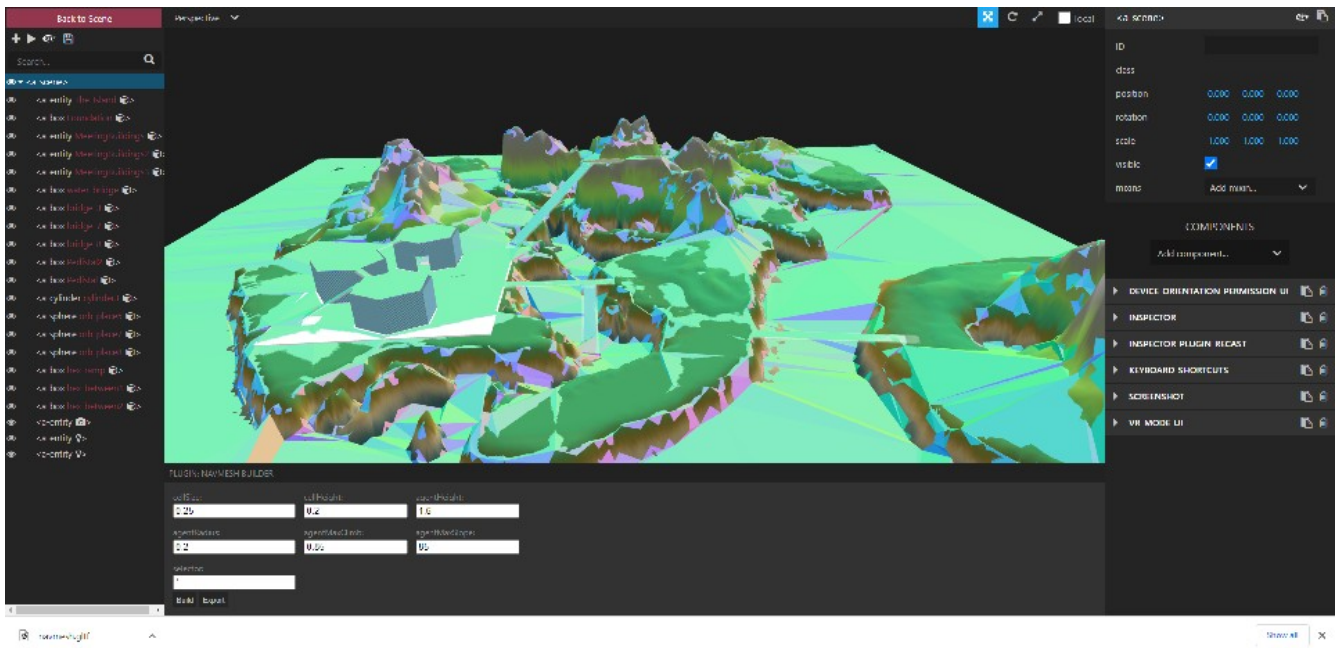
Here is the HTML/code.

<https://github.com/Mike-McAnally/SocialVR/blob/main/GardenXR.html>

Treasure Island Art Gallery



Treasure Island Art Gallery is another example. There is a the lagoon, with a submerged treasure chest which is full of simulated gold!



Check it out here:

<https://funbit64.com:3025/Tisland.html>

Source on GitHub:

<https://github.com/Mike-McAnally/WebXR-Misc/blob/main/Tisland.html>

Source of Navmesh generator HTML (notice the parameters used above in the image):

<https://github.com/Mike-McAnally/WebXR-Misc/blob/main/Tislandnavmesh.html>

3rd Party Examples

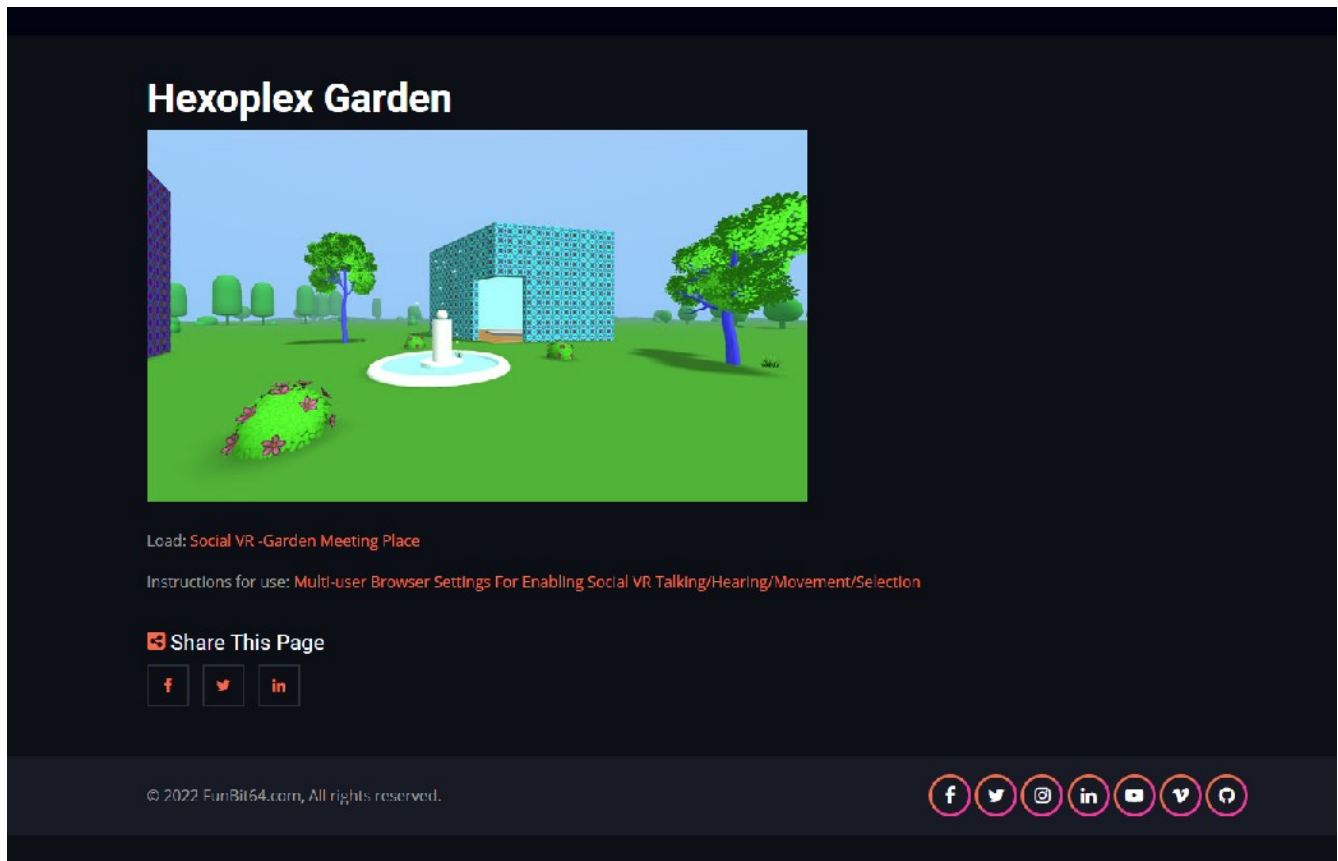
Now finally, I would like to include a few links to some really helpful third party open source repositories on GitHub:

<https://stemkoski.github.io/A-Frame-Examples/>

<https://github.com/diarmidmackenzie>

<https://gitlab.com/zach-geek/aframe-enviropacks>

VR Launch Pages

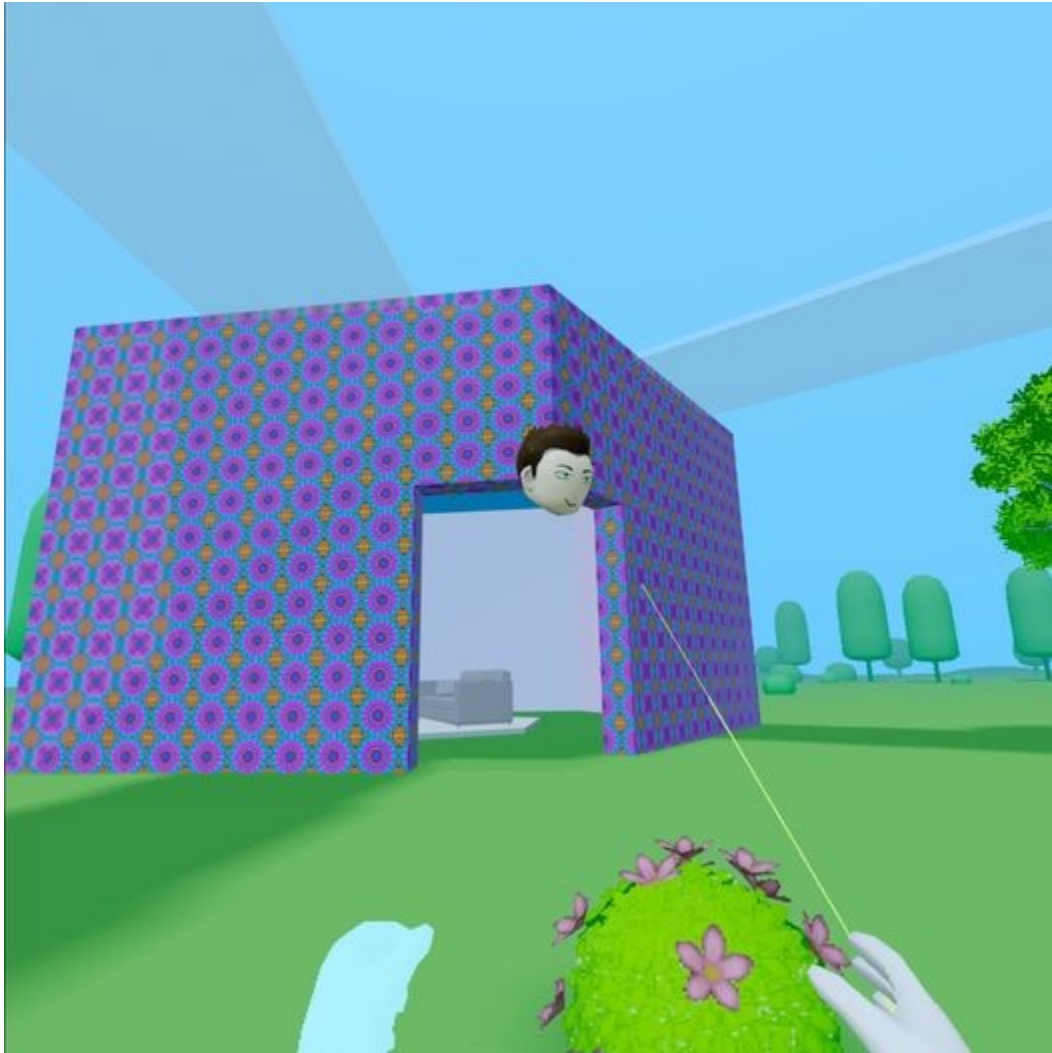


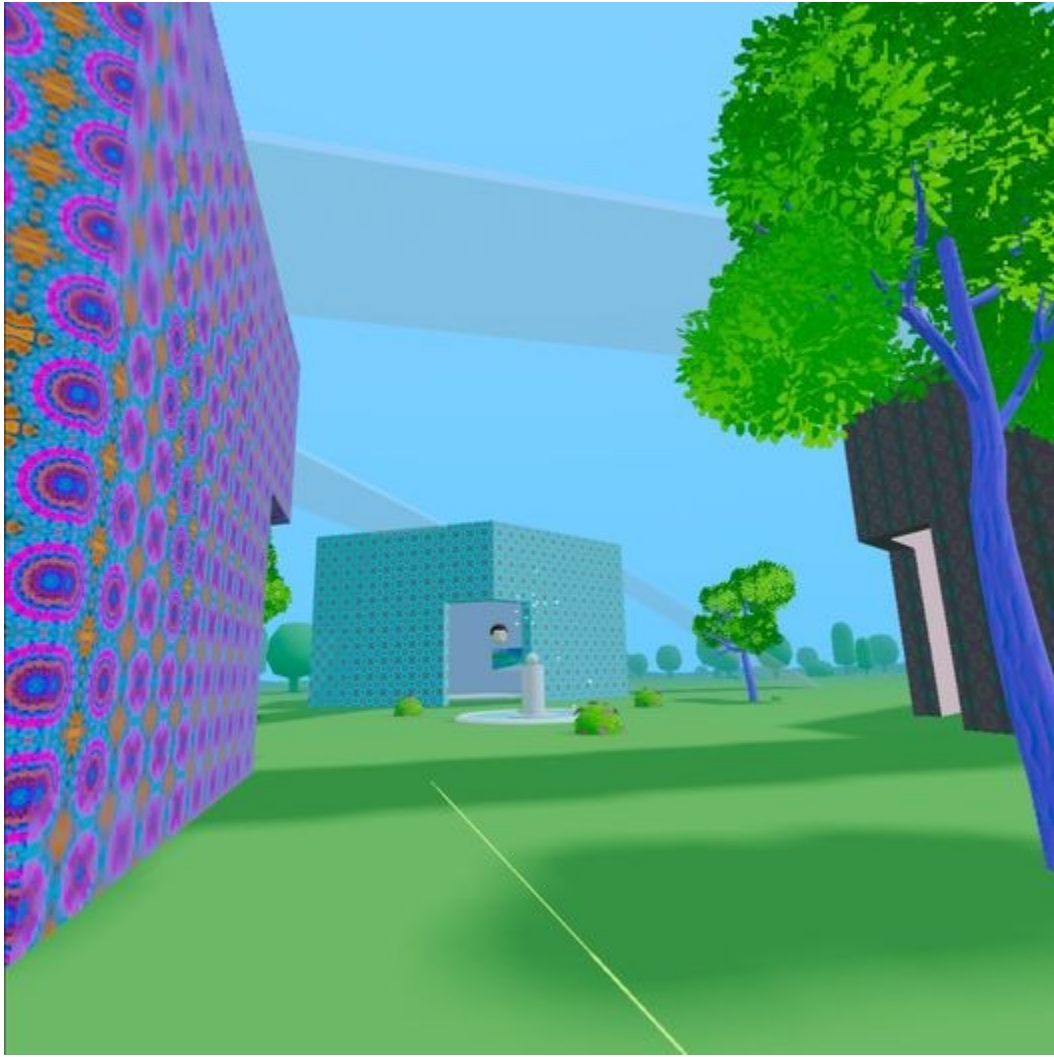
At some point you may need to have Launch Pages for your WebXR examples. Jumping from page to page inside a VR headset is less than optimal because of the wait times for loading of assets. Providing a launch page mitigates the user experience somewhat. Because we have installed a LAMP stack earlier, Drupal is an excellent solution to allow you to customize a VR webserver and content management system. With many features and a stable mature platform, Drupal you can control logins, and create content quickly, gluing together your work, as I have done in the Funbit64 examples.

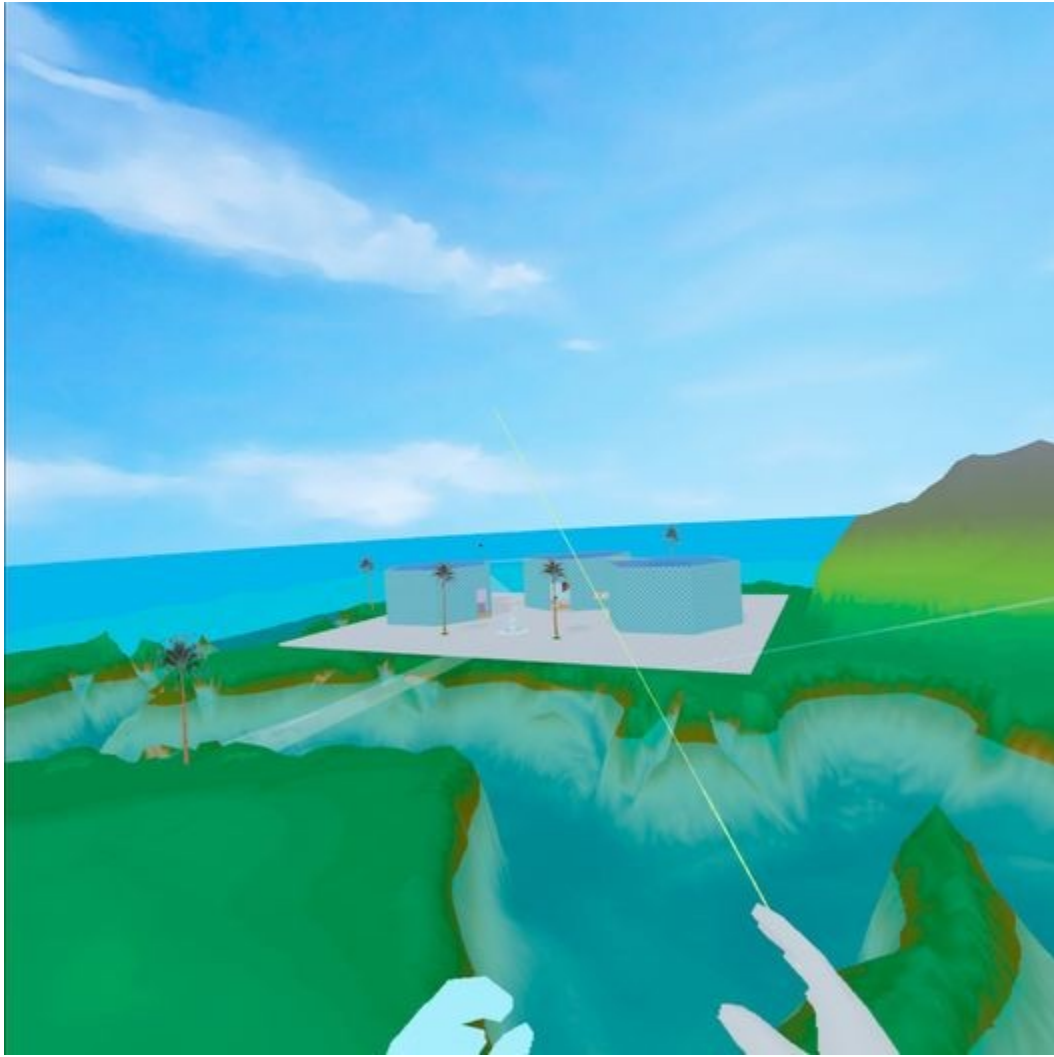
Here is a template you can also adapt for the front page:

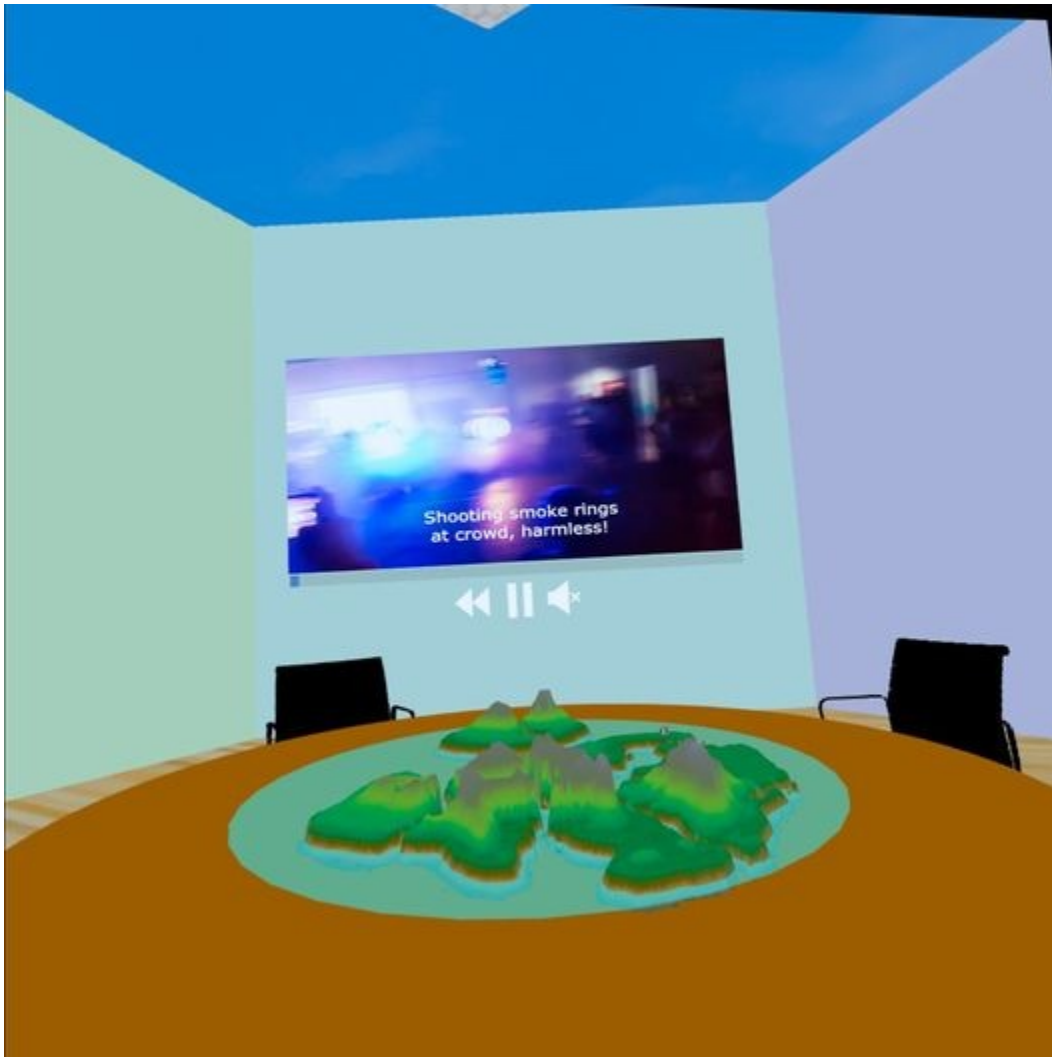
<https://github.com/Mike-McAnally/ServerMenu>

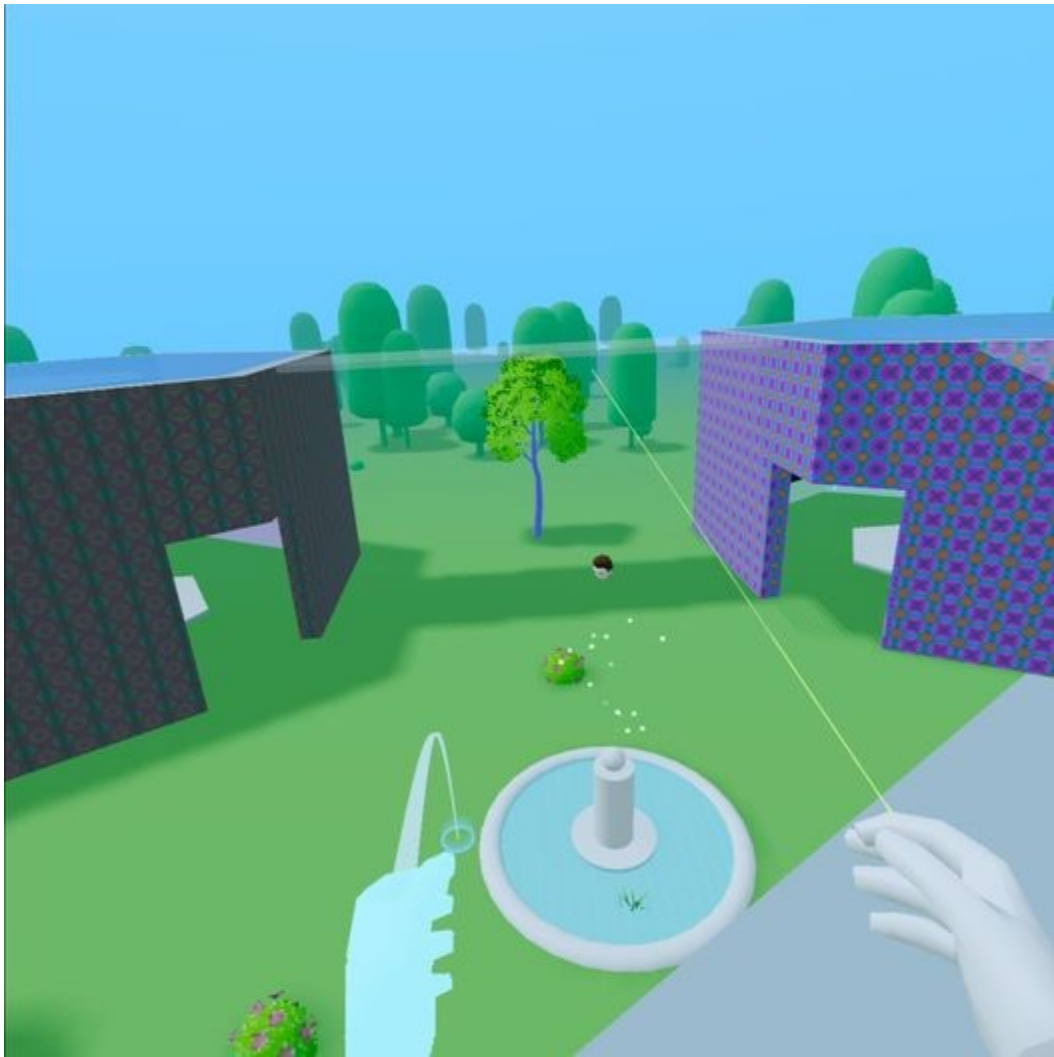
Inside VR Screen Shots Of Final Examples

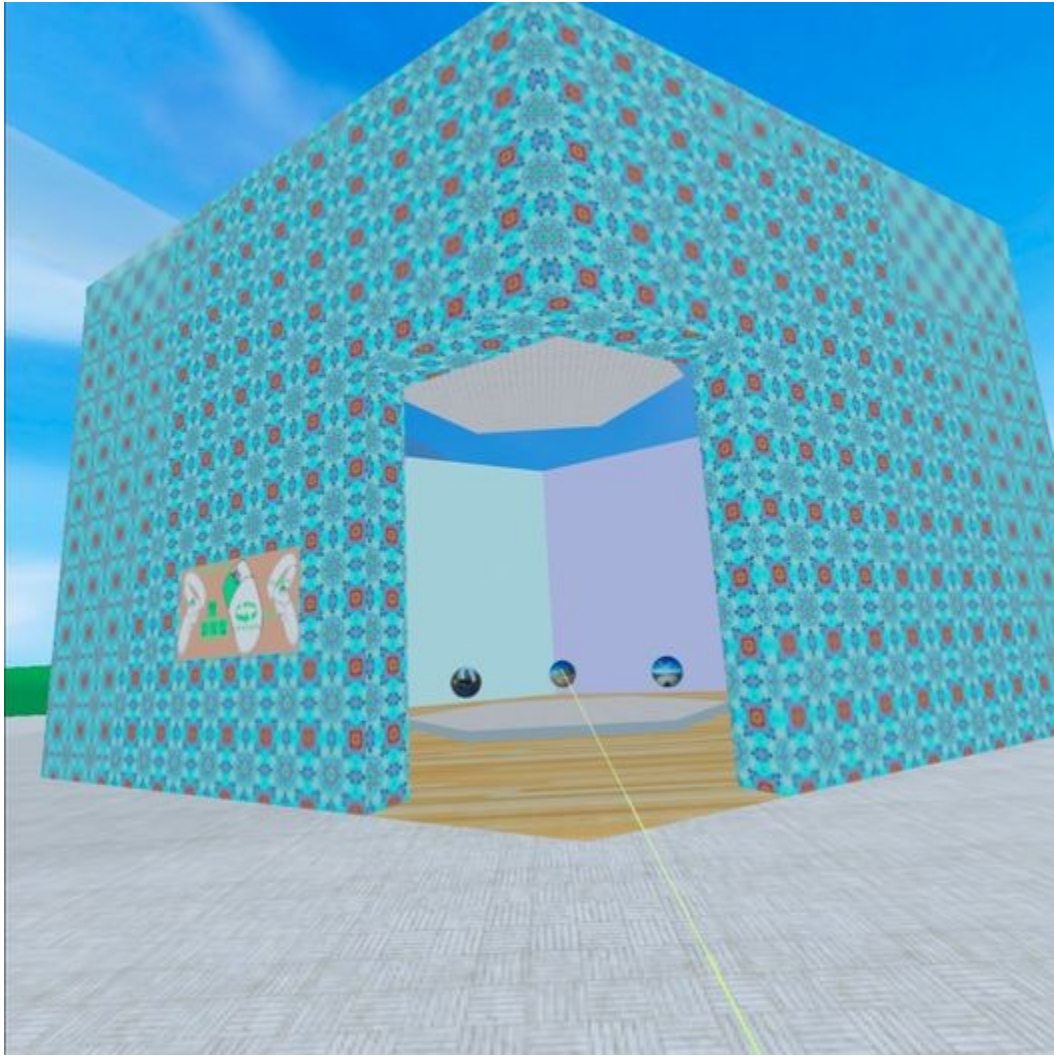




















Addendum (optional)

Installing Drupal 9

<https://linuxhostsupport.com/blog/how-to-install-drupal-9-cms-on-ubuntu-20-04/>

add to the Apache default.conf:

```
<Directory /var/www/html/drupal>
    Options Indexes FollowSymLinks

    AllowOverride All

    Require all granted

    RewriteEngine on

    RewriteBase /

    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule ^(.*)$ index.php?q=$1 [L,QSA]
</Directory>
```

use localhost

add ufw port under advanced for tcp and udp then reboot

Grant all privileges to drupal9_user

Recommend using this dark theme

<https://www.drupar.com/theme/zuvipro>