A New Reality

In this workshop we will explore using web technologies to create exciting VR & AR applications.

Pre-requisites

You will require the following:

- A basic knowledge of HTML & JavaScript
- A web server of some description (we have included a node/express one)

VR with Web Technologies

But why use web technologies to develop VR experiences in the first place?

Web technologies have several advantages over other approaches such as Unity framework:

- Utilize what you know already e.g. JavaScript & HTML knowledge
- Make use of existing libraries such as d3.js and HTTP infrastructure e.g. caching
- Cross-platform & runs on many mobile & tablet devices
- Uses open standards
- No plug-ins required
- Embed content within web pages

Of course, there are also some scenarios where web technologies are probably not the best choice for developing VR experiences:

- Where performance is important
- If you need to protect the content from distribution
- When you want to benefit from the distribution mechanisms such as Steam
- If you are not happy to build functionality yourself e.g. Unity contains a lot of functionality around collision detection that you will have to develop yourself

Foundations

Pretty much all web based VR experiences make use of the following technologies:

- JavaScript & HTML
- WebGL to render content
- WebVR to get device positioning information
- Device orientation events (mobile devices only)

Working with WebGL & WebVR directly is difficult (but not impossible!) so most developers will choose to use a framework.

In this work shop we will use a framework called A-Frame. There are other options such as ReactVR but A-Frame is probably the most mature & easy to use.

A-Frame

A-Frame was developed originally by Mozilla and has the following advantages:

- It's declarative and very easy to use
- The same A-Frame code will work on all device from tablet to phone to full headset
- A-Frame is very extensible and takes an approach called Entity Component System (ECS) which promotes composition of functionality
- A-Frame is built on the three.js library & makes three.js functionality available should you need it
- Lots of extensions are available (we have included a few in the assets folder)

Foundations

First up you will need to copy the workshop code onto your machine so please copy it from the USB stick or the URL given.

When you have copied the folder you will find it contains the following:

- node_modules
- workshop (this is the main directory we will work with today)
- app.js (a tiny bit of node code to run an express webserver used for the examples)
- Workshop.docx (instructions in Word format)
- Workship.pdf (instructions in PDF format)
- Slides (presentations given at start of workshop)
- package.json
- package-lock.json

Within the workshop directory are the following files:

- ar (this contains special image files used as a placeholder for augmented reality)
- assets (images & textures we will use)
- completed (working versions of the examples in case you need to refer to them)
- lib (short for library and contains A-Frame & A-Frame-Ar.js files and some third party extensions)
- template.htm (a file containing a reference to A-Frame library and scene element to base examples on)
- buzzConf.htm (a more complex example)

If you want to call your directories something else that's fine but remember to change the paths in the examples otherwise it won't work so probably best to leave it as is – you have been told..

Example 1 - Hello A-Frame

Let's get started with probably the simplest example possible.

Within the workshop directory create a new html file called 01.html

Add a doctype (this isn't strictly necessary but good practice):

```
<!doctype html>
```

After this add a reference to the A-Frame Library in the head element:

```
<head>
<script src="libs/aframe.js"></script>
</head>
```

Now below the head element add the A-frame scene element (a-scene). This special element acts as container for all our A-Frame objects and handles basic scene setup stuff:

```
<a-scene>
</a-scene>
```

Within the a-scene element let’s add a box/cube with the a-box element:

```
<a-box
width="2"
height="2"
depth="2"
color="#ff0000"
position="0 1 -5"
rotation="0 45 0"
scale="1 1 1"
></a-box>
```

You should have ended up with the following code:

```
<!doctype html>
<head>
	<script src="libs/aframe.js"></script>
</head>
```

```
<a-scene>

        <a-box

        width="2"

        height="2"

        depth="2"

        color="#ff0000"

        position="0 1 -5"

        rotation="0 45 0"

        scale="1 1 1"

        ></a-box>

</a-scene>
```

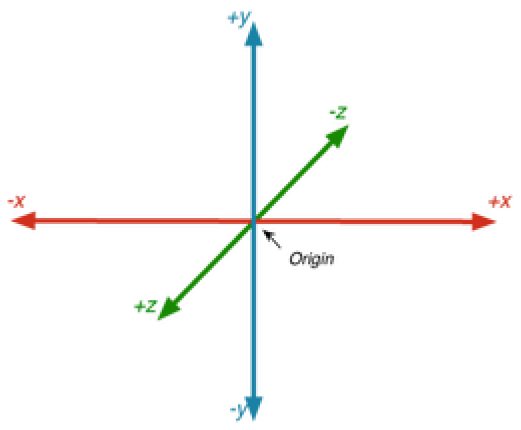
In this example we have used an A-Frame primitive (a-box) & defined the width, height & depth attributes to define the boxes size, colour and rotated it on the y axis.

Let's talk about A-Frame attributes - some of A-Frame's attributes such as position, rotation and scale take 3 parameters – x, y and z values.

A-Frame views a screen as follows:

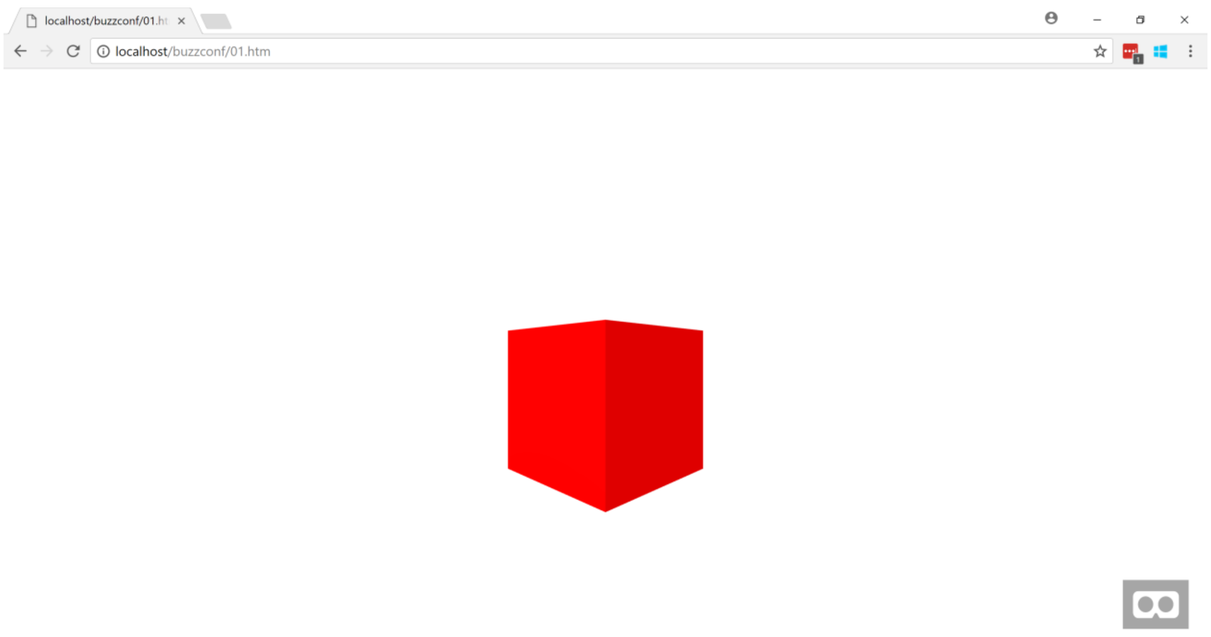
- X is across the screen
- Y up and down
- Z towards you

These values can be positive or negative as illustrated by the diagram below:



Source: https://mayaposch.files.wordpress.com/2012/12/opengl_coordinate_system.png?w=262&h=215

Now save the file, open it in a browser and you should see something like:

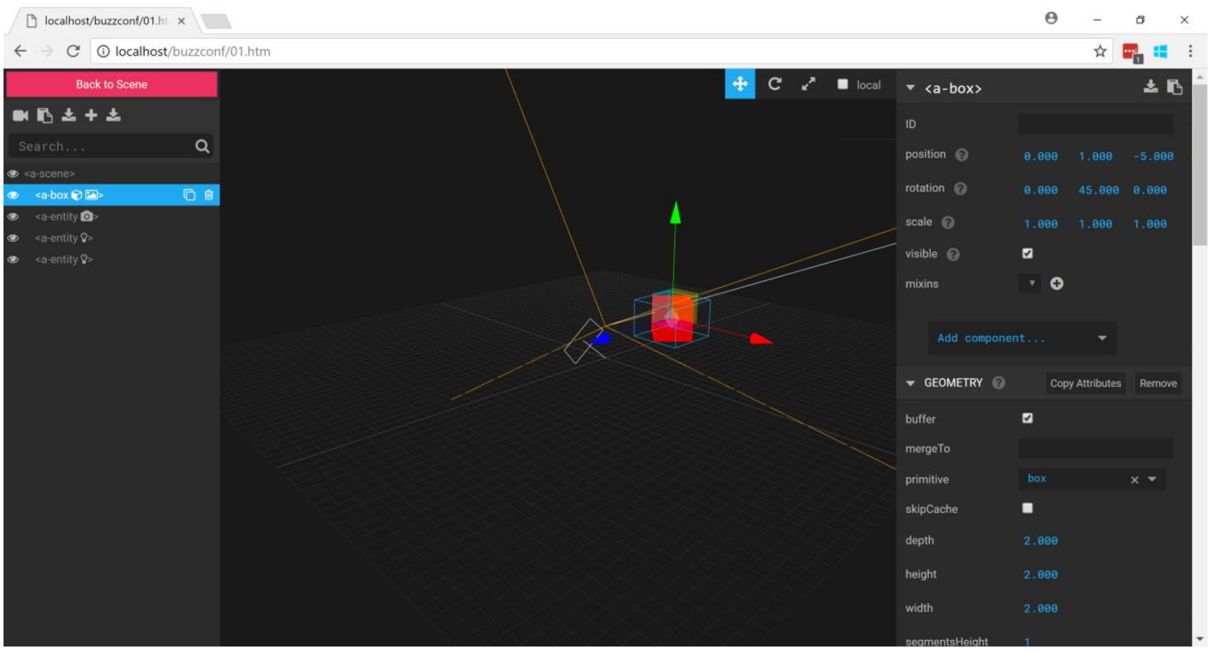


To make sure you are happy with how this works do the following:

- Have a play with some of the attributes such width, depth and height, rotation and scale
- Move the cube towards you
- Position the cube to the right of the scene
- Change the cubes colour

Example 2 – A-Frame GUI Scene Editor

A-Frame contains an inbuilt GUI scene editor which can be useful for debugging and creating more complex scenes – let's open this now by pressing **ctrl + alt + I** (note you need internet connectivity) and you should see something like:



ID			
position	0.000	1.000	-5.000
rotation	0.000	45.000	0.000
scale	1.000	1.000	1.000
visible	✓		
mixins			
buffer	✓		
mergeTo			
primitive	box		
skipCache	☐		
depth	2.000		
height	2.000		
width	2.000		
segmentsHeight	1		

Cool eh?

You can use it to create new and modify existing elements and even export your modifications.

Try the following out:

- Re-position the cube using the GUI editor
- Add a new object to the scene
- Export the scene

Example 3 - Primitives

A-Frame contains many inbuilt primitive shapes you can utilize & you can even create your own.

Copy template.htm and name it 03.html (don't worry you didn't miss a workshop – in example 2 we just looked at the GUI editor).

Add the following mark-up within the a-scene element:

```
<a-entity position="0 0 5">
    <a-camera></a-camera>
</a-entity>
```

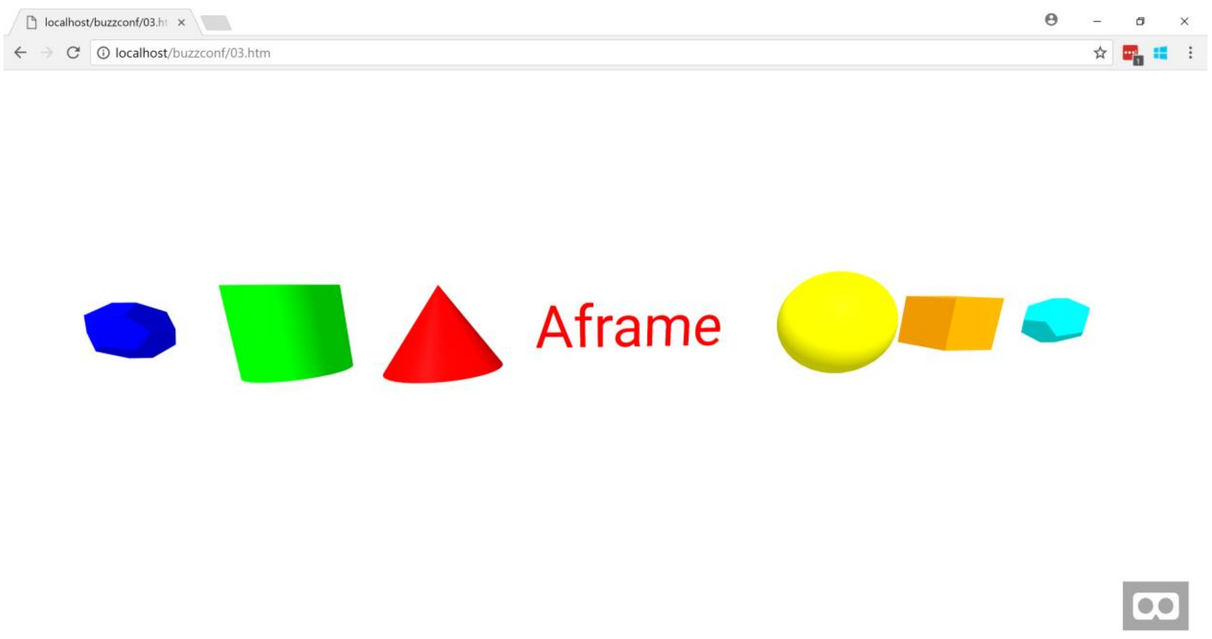
Here we have declared new camera element that has been wrapped by an entity element to position the camera slightly back from its default position. Notice how we can nest a-frame objects.

Next let's add some of the primitives that A-Frame supports: beneath the entity element we added:

```
<a-dodecahedron position="-15 0 -5" color="#0000ff" radius="1"></a-dodecahedron>
```

```
<a-cylinder position="-10 0 -5" color="#00ff00" height="3" radius="1.5"></a-cylinder>

<a-cone position="-5 0 -5" color="#ff0000" radius-bottom="2" radius-top="0" height="3"></a-cone>

<a-text position="-2 0 -5"  width="50" height="50" color="#ff0000"  value="Aframe"></a-text>

<a-ring position="5 0 -5" color="green" radius-inner="2" radius-outer="2"></a-ring>

<a-sphere position="10 0 -5" color="yellow" radius="2"></a-sphere>

<a-box position="15 0 -5" color="orange" depth="2" height="2" width="2"></a-box>

<a-dodecahedron position="20 0 -5" color="#00ffff" radius="1"></a-dodecahedron>
```

You should see something like:



Try out the following:

- See if you can stack some objects on top another
- Create a Christmas tree out of the primitive shapes complete with baubles!

Example 4 - Textures and Animation

Time to play with textures and add a simple animation.

Again copy template.htm and name the new file 04.html.

We will need a way to load textures and for this we will use A-Frames a-assets element.

This is a special A-Frame element that will take care of the loading & caching of various file types such as images and sounds.

We declare the image we want to use as a texture with a standard good old HTML img element:

```
<a-assets>
```

```
        <!-- https://freestocktextures.com/texture/brick-wall-renovated,760.html -->
        <img id="brick-texture" src="assets/brick.jpg">
  </a-assets>
```

Next let’s add a box:

```
<a-box position="0 1 -5" rotation="0 45 0" width="4" height="4" depth="4"></a-box>
```

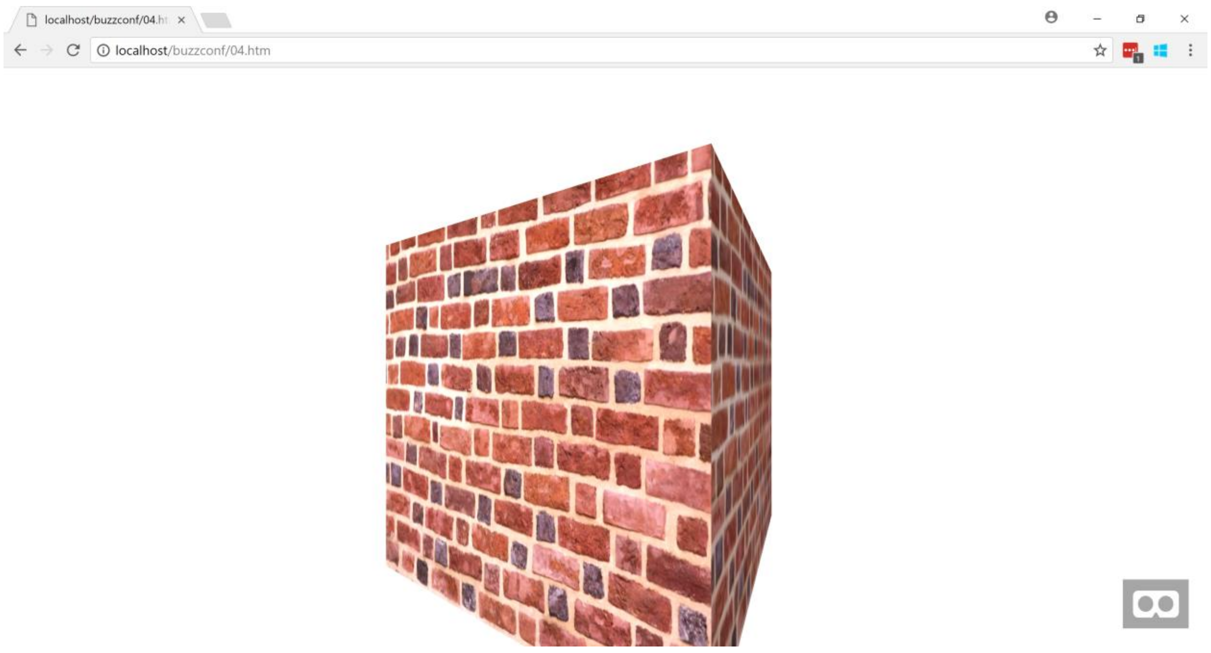
Now let’s apply our texture to the box with the material attribute - note the use of the id selector to define which image to use:

```
<a-box position="0 1 -5" rotation="0 45 0" width="4" height="4" depth="4" material="src: #brick-
texture">
```

And finally let’s add a simple declarative animation with the a-animation element by adding the following to inside the box element:

```
<a-animation attribute="rotation" repeat="indefinite" to="0 360 0" fill="forwards"
dur="10000"></a-animation>
```

Your example should look like the following:



Example 5 – Interaction

In this example we will see how to add interactivity using standard DOM events by changing the colour of a box when a button is clicked.

Again copy template.htm and name the new file 05.html.

Outside of the scene element add a standard html button:

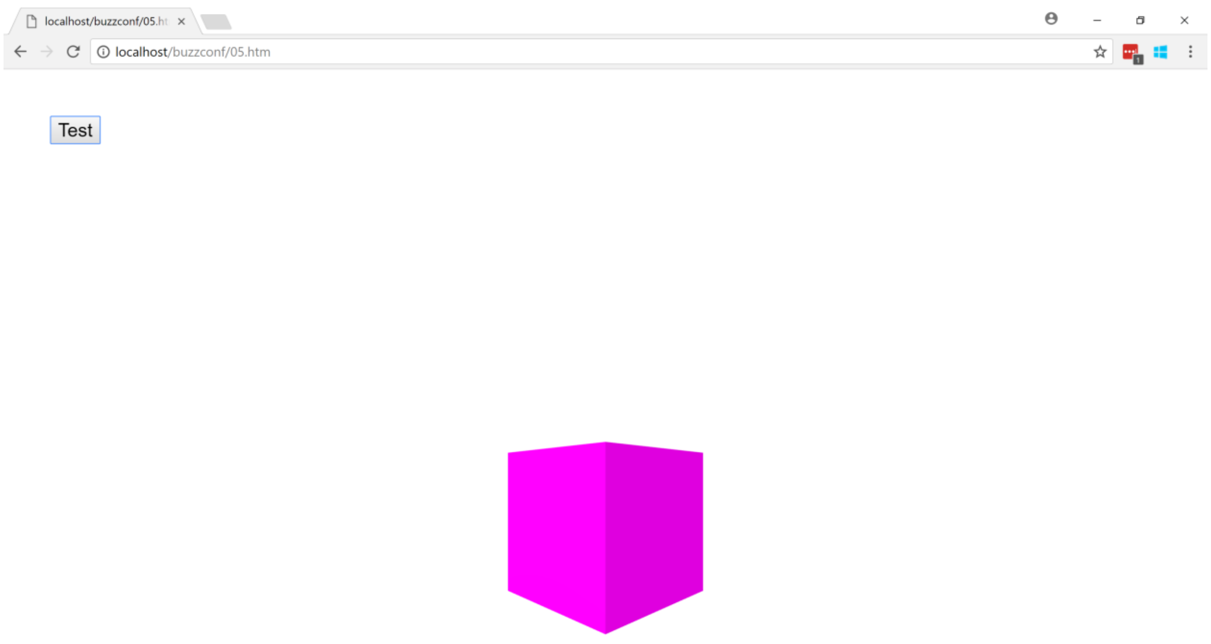
```
<input type="button" id="myButton" value="Test" style="margin:50px; font-size: 20px;"></input>
```

Next add the following within the scene element:

```
<a-box id="myBox" position="0 1 -5" rotation="0 45 0" width="2" height="2" depth="2"
color="#ff0000"></a-box>
```

Next add a script block below the box with the following code to handle the click of the button – note how we use standard DOM methods to get the box and change the colour attribute:

```
<script>
        var button=document.querySelector('#myButton');
        button.addEventListener('click', function() {
                var scene = document.querySelector('a-scene'),
                myBox = document.querySelector('#myBox');
                myBox.setAttribute('color', '#ff00ff');
        });
</script>
```



Example 6 – Components and interactivity

A strength of A-Frame is reusability and composition.

We can easily make the previous functionality generic by using components.

Start off by copying the 01.html file, clear the scene element and name the new file 06.html.

Now instead of interacting with the scene by clicking a button we will allow the user to interact with it directly.

We need something to allow the user to select items and for this we use the A-Frames cursor - this will simulate click events within the scene and take care of some complex logic collison and projection logic.

Note on a desktop this selection can be controlled with the mouse and keyboard and when used with a headset or mobile device this selection will be based on the users gaze.

We will place the cursor within the camera element, so it follows the field of view:

```
<a-camera position="0 2 4">
    <a-cursor color="#4CC3D9" fuse="true" timeout="10"></a-cursor>
</a-camera>
```

Now clear the contents of the scene element and add the following:

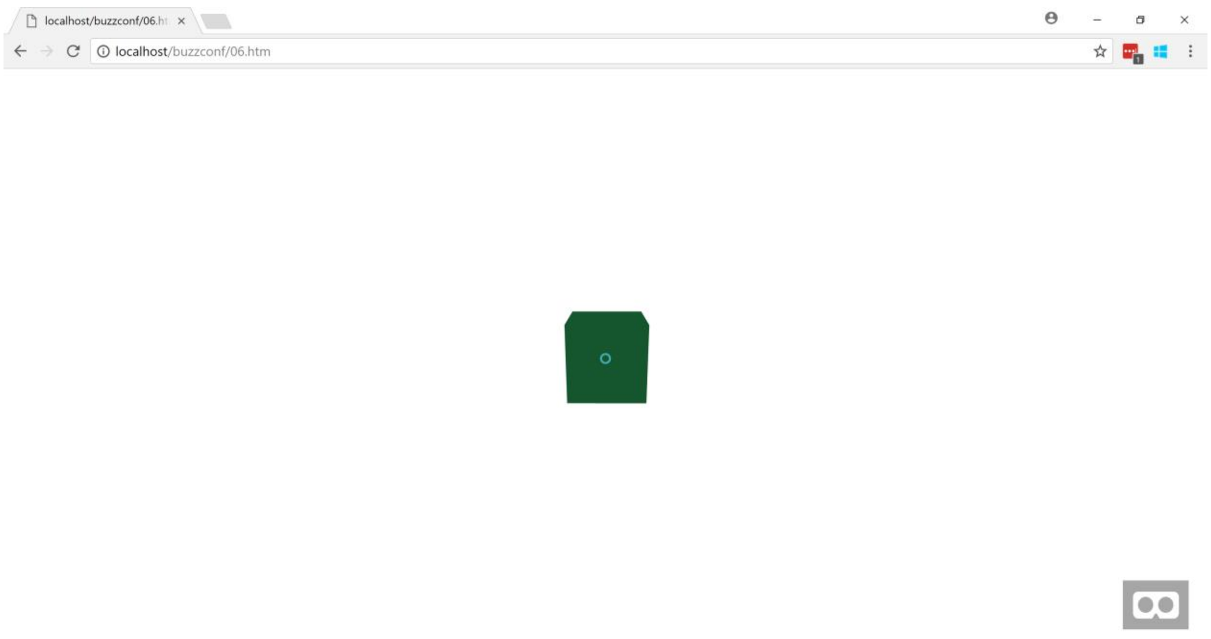
```
<a-box random-color-on-click position="0 1 -5" width="2" height="2" depth="2"
color="#ff0000"></a-box>
```

Note the random-color-on-click attribute – this isn't an A-Frame functionality we will create this ourselves with the register component method.

Add the following script block before the a-scene element to register this functionality:

```
<script>
  AFRAME.registerComponent('random-color-on-click', {
   schema: {default: ''},
   init() {
    this.el.addEventListener('click', () => {
     //http://stackoverflow.com/questions/5092808/how-do-i-randomly-generate-html-hex-color-codes-using-javascript
     var randomColor = '#' + (Math.random() * 0xFFFFFF << 0).toString(16);
     this.el.setAttribute('color', randomColor);
    });
   }
  });
 </script>
```

The first param is the name of the functionality and schema is for defining the structure of input parameters e.g. we could allow user to define the colour.



A-Frame Behind the Scenes

Feel free to skip this section if you are not interested in how A-Frame works behind the scenes but for those that are read on!

In the previous example we looked at creating reusable functionality with the registerComponent API. A-Frame uses a similar approach to provide core functionality.

Below is the A-Frame framework code for the position attribute that behind the scenes uses three.js functionality to set an objects position:

```
var registerComponent = require('../core/component').registerComponent;

module.exports.Component = registerComponent('position', {

        schema: {type: 'vec3'},

        update: function () {

          var object3D = this.el.object3D;

          var data = this.data;

          object3D.position.set(data.x, data.y, data.z);

        }

});
```

Whilst we are here A-Frames registerGeometry API is similar and can be used to create reusable components.

Below is the framework code for the box element that we used earlier:

```
var registerGeometry = require('../core/geometry').registerGeometry;

var THREE = require('../lib/three');

        registerGeometry('box', {

         schema: {

          depth: {default: 1, min: 0},

          height: {default: 1, min: 0},

          width: {default: 1, min: 0},

          segmentsHeight: {default: 1, min: 1, max: 20, type: 'int'},

          segmentsWidth: {default: 1, min: 1, max: 20, type: 'int'},

          segmentsDepth: {default: 1, min: 1, max: 20, type: 'int'}

         },

         init: function (data) {

          this.geometry = new THREE.BoxGeometry(
                data.width, data.height, data.depth,

                data.segmentsWidth, data.segmentsHeight, data.segmentsDepth);

            }

});
```

Next

For a more complex example please refer to buzzConf.htm.

This shows a rainy/windy camping scene that will change when the button is selected.

This example demonstrates the following:

- Collision detection/scene interaction
- Audio
- Lighting
- Animation
- Loading models & textures

So now you know the basics - go create!