Three.js and A-frame

`a-frame` addresses a different problem space in VR and AR. Authoring WebGL is really hard.

We will skeep the raw WebGL version, don't want to scare readers away by talking about shaders and compiling them into Javascript and how to use them. Instead, we'll move straight into the abstractions. I prefer to use three.js, a very powerful abstraction on top of native WebGL implementations... but even as an abstraction the code is still not easy to understand.

The code below assumes you've already downloaded and linked the library using `script` tags.

```
// Step 1
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera( 75,
  window.innerWidth/window.innerHeight,
  0.1, 1000 );

// Step 2
var renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

// Step 3
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );

camera.position.z = 5;

// Step 4
var animate = function () {
  requestAnimationFrame( animate );

  cube.rotation.x += 0.01;
```

```
  cube.rotation.y += 0.01;

  renderer.render( scene, camera );
};

// Step 5
animate();
```

The code draws a cube and rotates in both the X and Y axes. The demo is in this Codepen

I've broken the code into 5 different steps

1. Initialize the scene and the camera we'll use to view it
2. Create the WebGL render, set its size and attach it to the DOM
3. Create the Box and Mesh Object. Then create a cuber using the two objects we just created
4. Set up a Requet Animation Frame loop where we rotate the cube on the X and Y axes
5. Run the animation for the first time so the loop will start

The code is fairly simple as far as Three.js code is concerned, but you can see the difficulties for someone who's just starting or someone who just wants to throw together a quick prototype. The code become more complex when working with AR and more advanced features.

`a-frame` abstract the complexity of the Three.js code into a procedural language. The code below produces a similar result to the Three.js code.

The biggest difference is that `a-frame` uses markup, rather than Javascript, to describe the scene and the interactivity.

```
<!-- 1 -->
<a-scene>
<a-scene><!-- 2 -->box
    height="2" width="2" depth="2"
    position="0 2 -3.5"
    rotation="45 45 0"
```

```
    color="rebeccapurple">

    <!-- 3 -->
    <a-animation
      attrib<!-- 3 -->ion"
      dur="10000"
      to="360 360 0"
      repeat="indefinite"
      easing="linear"></a-animation>
  </a-box>

  <!-- 4</a-animation><!-- 4 -->tion="0 0 -4"
    rotation="-90 0 0"
    width="12"
    height="6"
    color="#7BC8A4"></a-plane>

  <!-- 5 -->
  <a-sky </a-plane><!-- 5 -->
  <a-sky color="pink"></a-sky>
</a-scene>
```

The biggest difference is the declarative markup that hides the Three.js code to achieve the same behavior.

I broke the `a-frame` code in 5 sections:

1. Defines the scene using the a-scene element
2. Creates a cube using the box object
    1. The position attribute is defined using a different coordinate system so you'll have to test the values until you get the result you want
3. The animation for the object is nested inside the object it's animating.
    1. Instead of using a RAF loop, we just tell it what the final values will be in the x, y and z axes (the `to` attribute)
4. Creates a plane and positions it as the bottom of the scene as the "floor" where we place the other objects
5. Defines the sky for the scene

Further ideas

The example in this post uses a single object and a single animation. It is definitely possible to build multiple objects, add lighting and camera effects as well as add multiple animations and interactivity.

A-Frame makes AR/VR easier. You can use AR.js as outlined in Creating Augmented Reality with AR.js and A-Frame