AR/VR Workshop
Part 5: **Three.JS** and **A-Frame**

Phillip G. Bradford

Learning plan

Introduction	Overview – outline goals Setting up google cardboard with glitch.com system
A-frame basics	Simple 3D a-frame examples Work with glitch.com
Foundations	JavaScript, DOM, events, Web-Components
A-frame components	Defining A-frame components
Three.JS and A-frame	Basics of Three.js for A-frame
Entity component architecture (ECA)	Three.js and ECA with standard OO paradigm
A-frame and planets	Complex 3D a-frame, work with complex a-frame detail and basic planetary math; illustrate ECA, geometries, controls, etc.
A-frame and animations	Goal: show how to do basic animation
Conclusion	Goal: review our learning

Outline

What is Three.JS?

How to use Three.JS with A-frame

Examples

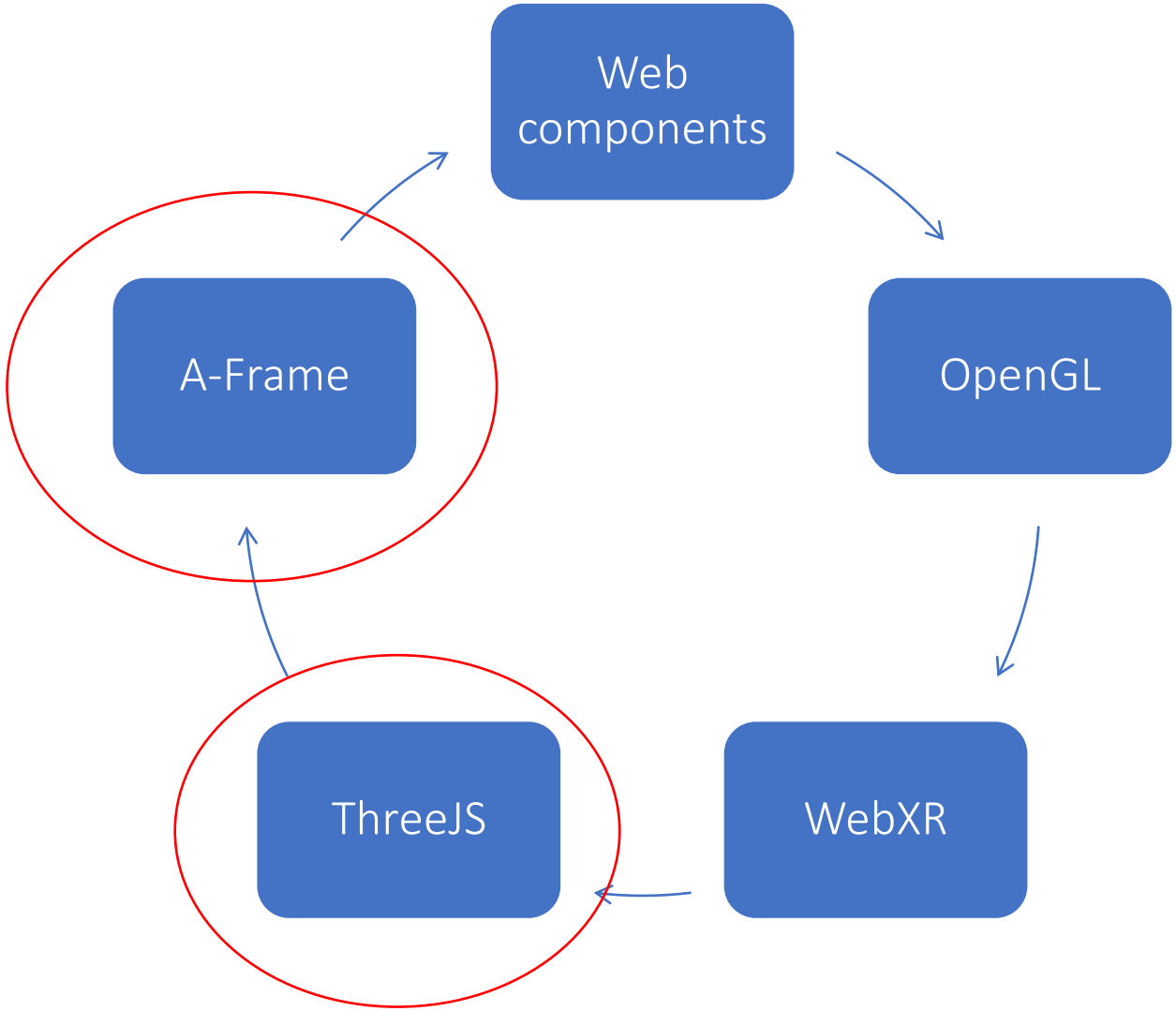
Motivation

A very popular JavaScript graphics library

- Based on WebGL
- Leverages graphics cards

Three.JS is built on WebXR which is built on WebGL which is built on OpenGL

High level view and learning path



Main components

Renderer

Scene

Camera

Set up

Make a directory: js

Download https://threejs.org/build/three.js

Into js

Windows <right-click> save as...

General Three.js code structure

Context – canvas

Camera – create, setup, position

Geometry

Materials

Lights

Mesh

Draw

Animation loop - requestAnimationFrame(render)

Update resize

Animation loop

The function **requestAnimationFrame(render)** applied to render asks the browser to animate when a state change occurs

```
function animate() {
    requestAnimationFrame( animate );
    square.rotation.x += 0.025;   /* state change - browser re-renders */
    renderer.render( scene, camera );
}
animate();
```

Browser re-renders by calling animate(time) !

Changing the speed of the animation loop

```
const interval = setInterval( () => { line.rotation.x += 0.1; }, 1500);
function animate(time) {
    requestAnimationFrame( animate );
    renderer.render( scene, camera );
    //console.log('Animate:' + time.toString());
}
animate();
```

Set up html

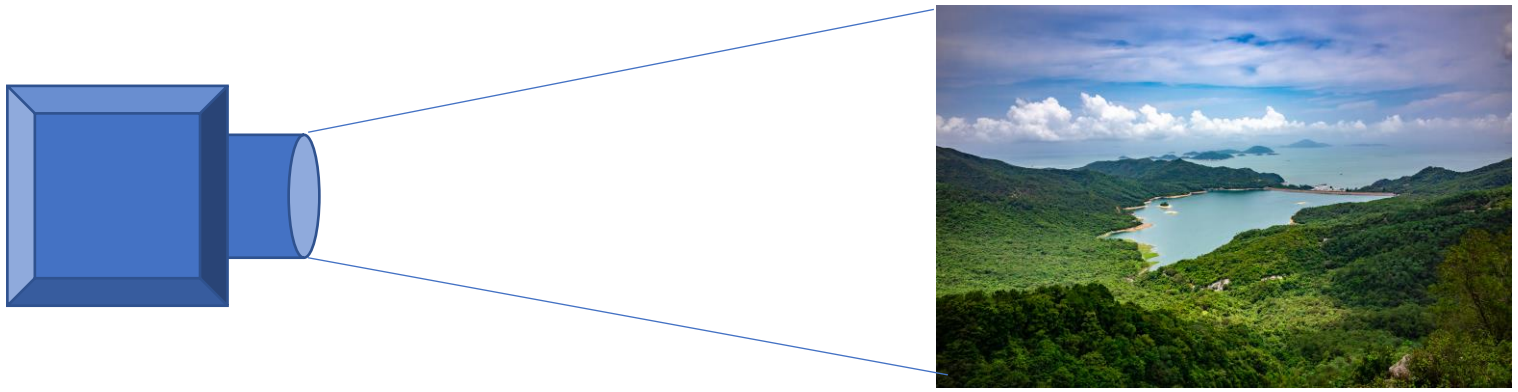
```
<!DOCTYPE html>
    <html>
      <head> <meta charset="utf-8">
       <title>My first three.js app</title>
           <style> body { margin: 0; } </style>
       </head>
     <body> <script src="js/three.js"></script>
 <script> // Our Javascript will go here. </script>
</body>
</html>
```

Camera

Scene

Camera

Renderer



Perspective Camera

```
const camera = new THREE.PerspectiveCamera( 75,
                    window.innerWidth / window.innerHeight, 0.1, 1000 );
```

Field of view: 75 degrees

Aspect ratio: width/height

0.1, 1000: Near, far clipping

Perspective Camera()

```
camera = new THREE.PerspectiveCamera(45,
                                     window.innerWidth / window.innerHeight,
                                     0.1, 1000);
    // position and point the camera to the center
    camera.position.x = 15;
    camera.position.y = 16;
    camera.position.z = 13;
    camera.lookAt(scene.position);
```

THREE.Scene()

This data structure holds elements of a picture

Also holds sub-components

Holds objects, lights, textures,

THREE.BoxGeometry(x,y,z)

Lambert and Phong materials

THREE.WebGLRenderer() – uses canvas tag

```
const renderer = new THREE.WebGLRenderer();
```

Generating image via a model

```
renderer.setSize( window.innerWidth, window.innerHeight );
```

Fit to our screen

```
document.body.appendChild( renderer.domElement );
```

Web component

Object3D

https://github.com/mrdoob/three.js

https://threejs.org/docs/#api/en/core/Object3D

THREE.Line

THREE.CircleGeometry

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
```

References

https://cdn.jsdelivr.net/npm/three-orbitcontrols@2.110.1/