Lab 3: WebXR

CS410/510: VR/AR Development

Ehsan Aryafar earyafar@pdx.edu

Sam Shippey sshippey@pdx.edu

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

Introduction

- WebXR is a technology which enables developers to build VR applications on the web beyond simulated desktops
- Full games can be built into the browser

Introduction

- This lab
  - Brief introduction to many different technologies
  - Quick, non-graded, optional demo(s)
- This is probably a totally different workflow if you're unfamiliar with web development, so don't worry if you feel lost

Introduction

- VR ecosystem is very large
- Many different headsets: 3DoF/6DoF
- Many operating systems: Windows, Linux, OSX, Android, iOS…
- Many underlying libraries: OpenXR, OpenVR/SteamVR, Oculus…

Oculus

HTC Vive

Valve Index

Introduction

- VR ecosystem is full of competing, not cooperating technologies
- Underlying libraries are different enough that different ports must be developed
- Underlying libraries must be ported to different operating systems
- Different headsets have different capabilities

Introduction

- This problem is not unique to VR
- Common solution: Put everything in a browser, export app later (ideally as a “native web app”) if ever
- Examples
  - Chat/IM platforms (Slack, Discord, Element, Zulip…)
  - Email clients (gmail, web clients)
  - Games (agar.io)
- Browsers run on basically all operating systems/hardware

Introduction

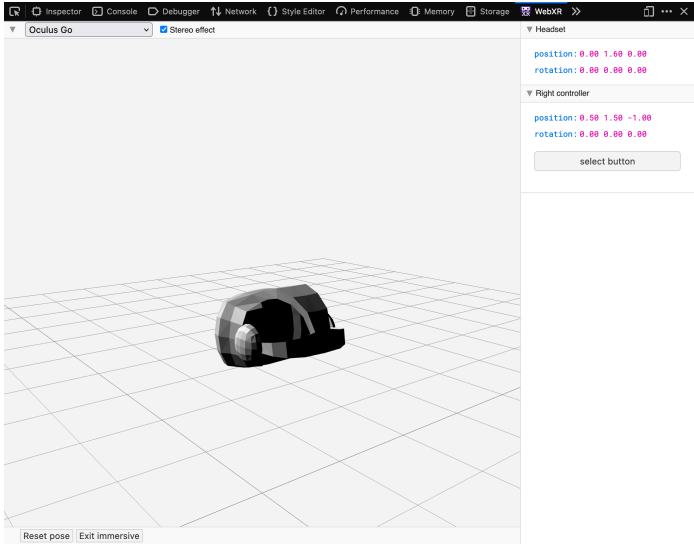
- WebXR: Render 3d scenes and pass through VR pipeline
- Ways to interact with this
  - A-Frame: Very simple HTML-only library for rendering 3d shapes and meshes
  - ThreeJS: JavaScript 3d rendering library, has VR interaction tools
  - BabylonJS: TypeScript rendering library, has a VR mode

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

WebXR Emulator

- Web browser extension that works on both Chrome and Firefox
- Works by adding a new tab to the inspector
- Right click in website using WebXR -> Inspect -> Select WebXR Emulator

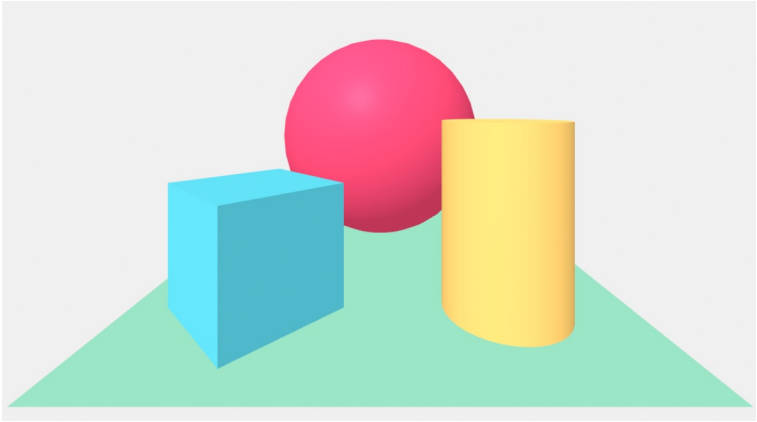


Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

A-Frame

- Very simple, easy to use HTML-only library for webXR
- https://aframe.io, https://glitch.com/~aframe
- JavaScript can interact with HTML, so we can manipulate it from there



A-Frame

- Example code, note HTML-style open/close braces

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <script src="https://aframe.io/releases/1.4.1/aframe.min.js"></script>
5     </head>
6     <body>
7       <a-scene>
8         <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
9         <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
10        <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"></a-cylinder>
11        <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>
12        <a-sky color="#ECECEC"></a-sky>
13      </a-scene>
14    </body>
15  </html>
```

Import A-Frame

Built-in meshes

A-Frame

- Workflow
  - Import A-Frame by adding the <script> tag shown on previous slide
    - Can also host locally
  - Build website with <a-scene> embedded somewhere in it
  - Write description of scene in tag
  - Manipulate scene with JavaScript/generate scene with server-side rendering/etc

A-Frame

- More example code: Loading models

```
<script src="https://aframe.io/releases/1.4.1/aframe.min.js"></script>
<a-scene background="color: #ECECEC">
<a-assets>
    <a-asset-item id="cityModel" src="https://cdn.aframe.io/test-models/models/glTF-2.0/virtualcity/VC.gltf"></a-asset-item>
</a-assets>
<a-entity gltf-model="#cityModel" modify-materials></a-entity>
</a-scene>
```

A-Frame

- Demo
  - Create a similar scene to the first one, with different 3d objects
  - Use the WebXR emulator to view your scene
- Steps:
  - Create a file called “demo-1.html”
  - Add boilerplate
  - Add objects
  - <a-*shape*></a-*shape*>

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://aframe.io/releases/1.4.1/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
    </a-scene>
  </body>
</html>
```

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

JavaScript

- JavaScript is a commonly used programming language that runs in browsers
- Originally meant to allow interactivity on webpages it has now spread to the backend as well (through nodejs)

JavaScript

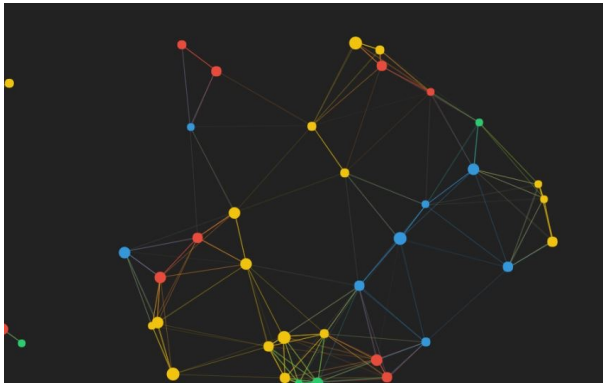
- <script></script> tags in html
- Basic syntax:
  - let x = 2;
  - let str = ‘string’;
  - if (value) { … } else { … }
  - function f(x_1, x_2 … x_n) { return x_1+x_2; }
- These are ungraded, you don’t have to know JavaScript, but it’s not that hard to pick up

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

WebGL

- HTML5 introduced the <canvas> element allowing for the drawing of arbitrary 2d graphics on webpages
- WebGL (Web Graphics Library) is an extension of the notion of the canvas that allows for 3d rendering



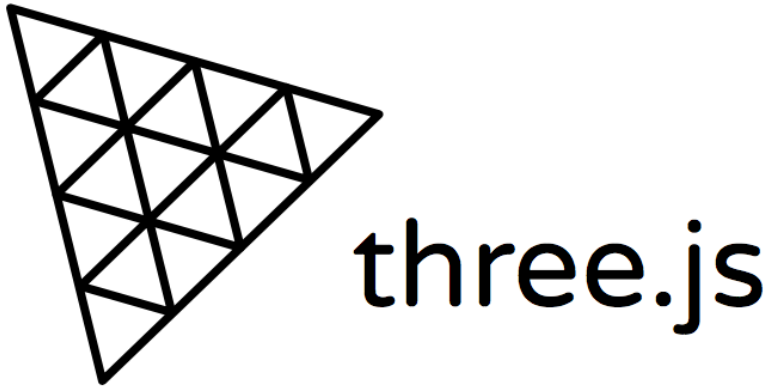
Moving canvas drawing
https://codepen.io/indieklem/pen/mdJONg

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

ThreeJS

- Underlying library of A-Frame
- Includes much more detail for manipulating camera controls
- Potentially much more complicated workflow



ThreeJS

- Example code
- https://github.com/mrdoob/three.js/blob/master/examples/webxr_vr_ballshooter.html

```
scene = new THREE.Scene();
scene.background = new THREE.Color( 0x505050 );

camera = new THREE.PerspectiveCamera( 50, window.innerWidth / window.innerHeight, 0.1, 10 );
camera.position.set( 0, 1.6, 3 );

room = new THREE.LineSegments(
        new BoxLineGeometry( 6, 6, 6, 10, 10, 10 ),
        new THREE.LineBasicMaterial( { color: 0x808080 } )
);
room.geometry.translate( 0, 3, 0 );
scene.add( room );

scene.add( new THREE.HemisphereLight( 0x606060, 0x404040 ) );

const light = new THREE.DirectionalLight( 0xffffff );
light.position.set( 1, 1, 1 ).normalize();
scene.add( light );

const geometry = new THREE.IcosahedronGeometry( radius, 3 );
```

```
for ( let i = 0; i < 200; i ++ ) {

        const object = new THREE.Mesh( geometry, new THREE.MeshLambertMaterial( { color: Math.random() * 0xffffff } ) );

        object.position.x = Math.random() * 4 - 2;
        object.position.y = Math.random() * 4;
        object.position.z = Math.random() * 4 - 2;

        object.userData.velocity = new THREE.Vector3();
        object.userData.velocity.x = Math.random() * 0.01 - 0.005;
        object.userData.velocity.y = Math.random() * 0.01 - 0.005;
        object.userData.velocity.z = Math.random() * 0.01 - 0.005;

        room.add( object );

}

//

renderer = new THREE.WebGLRenderer( { antialias: true } );
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.outputEncoding = THREE.sRGBEncoding;
renderer.xr.enabled = true;
document.body.appendChild( renderer.domElement );

//

document.body.appendChild( VRButton.createButton( renderer ) );
```

ThreeJS

- Workflow
  - Import ThreeJS (see below)
  - Add script tag with your code
  - Build out on top of this

```
<!-- Import maps polyfill -->
<!-- Remove this when import maps will be widely supported -->
<script async src="https://unpkg.com/es-module-shims@1.3.6/dist/es-module-shims.js"></script>

<script type="importmap">
        {
                "imports": {
                        "three": "../build/three.module.js",
                        "three/addons/": "./jsm/"
                }
        }
</script>
```

Outline

- Introduction
- WebXR Emulator
- A-Frame
- JavaScript
- WebGL
- ThreeJS
- BabylonJS and TypeScript

TypeScript

- Typed language that compiles to JavaScript
- JavaScript is very weakly typed and this can make development hard
- Typescript solves this at the cost of introducing another compilation step.

TypeScript

- All legal JavaScript is legal TypeScript
  - TypeScript is a *superset* of JavaScript
- If you know JavaScript, you can start adding Types to your existing code and it will compile correctly.

TypeScript

- Workflow
  - Take a JavaScript project, add tsconfig.json
    - Contains compiler options
  - Run compiler on TypeScript file
  - Get out JavaScript file of the same name

BabylonJS

- Similar library to ThreeJS with support for TypeScript
- Almost exactly the same workflow

```
var createScene = function () {
  var scene = new BABYLON.Scene(engine);

  // Add a camera to the scene and attach it to the canvas
  // Add a lights to the scene

  //Your Code

  return scene;
};
```