

Development

Developing Immersive Applications

Created by: Chek Tien TAN



Learning Objectives:

- describe common tools for developing immersive applications
- differentiate WebXR and OpenXR standards
- setup programming environment for building WebXR applications
- implement a minimal WebXR scene with Babylon.js

Key APIs

| API | Type | Platform | XR-Specific |
|---------|---------------|------------------|-------------|
| WebXR | Open Standard | Web | Yes |
| OpenXR | Open Standard | Native | Yes |
| Vulkan | Open Standard | Native | No |
| WebGL | Open Standard | Web | No |
| OpenGL | Open Standard | Native | No |
| DirectX | Proprietary | Native (Windows) | No |
| Metal | Proprietary | Native (Apple) | No |

Key Points:

- **WebXR** — web standard for VR/AR, browser-based, no install
- **OpenXR** — cross-platform native standard for VR/AR
- **WebGL** — foundational web graphics API, WebXR builds on it
- Use WebXR for accessibility; OpenXR for native features

Development Frameworks

| Framework | Type | WebXR | Key Features |
|---------------|---------------|---------|---|
| Unity | Native Engine | Partial | Industry standard, C#, asset store |
| Unreal Engine | Native Engine | Partial | High-fidelity graphics, Blueprint |
| Babylon.js | Web Framework | Full | TypeScript, WebXR helpers (this module) |
| Three.js | Web Framework | Manual | Most popular, flexible, manual setup |
| A-Frame | Web Framework | Full | HTML-based, beginner-friendly |
| PlayCanvas | Web Framework | Full | Web editor, collaborative |

Development Frameworks

| Framework | Type | WebXR | Key Features |
|-----------|--------------|-------|-------------------------------|
| CoSpaces | No-Code Tool | N/A | Educational, drag-and-drop |
| Spatial | No-Code Tool | N/A | AR/VR creation without coding |

Choosing a Framework:

- **Maximum accessibility?** WebXR (Babylon.js, Three.js, A-Frame)
- **Novel hardware interaction?** OpenXR SDK in C++
- **High visual fidelity?** Unity or Unreal Engine
- **No coding experience?** CoSpaces or Spatial

WebXR Support Across Browsers

| Feature Name | Standardisation | Chrome | Safari on visionOS | WebXR Viewer | Magic Leap Helio | Samsung Internet | Meta Quest Browser | Microsoft Edge | Wolvic | PI Bro |
|-----------------------|------------------------------------|------------------------|-----------------------|--------------|--|-----------------------|--------------------|--|----------------------|---------|
| WebXR Core | Explainer Spec MDN | Chrome 79 | Behind a feature flag | iOS | Magic Leap Helio 0.98 | Samsung Internet 12.0 | 7.0, December 2019 | Edge 87 on Windows Desktop Edge 91 on Hololens 2 | 0.9.3, February 2022 | Support |
| WebXR AR Module | Explainer Spec MDN | Chrome for Android, 81 | | iOS | Magic Leap Helio 0.98 | Samsung Internet 12.1 | 24.0, October 2022 | Edge 91. HoloLens 2 only | Wolvic Chromium 1.1 | 3.2 |
| WebXR Gamepads Module | Explainer Spec MDN | Chrome 79 | | | Partially supported on Magic Leap Helio 0.98 | Samsung Internet 12.0 | 7.1, December 2019 | Edge 87 on Windows Desktop Edge 91 on Hololens 2 | 0.9.3, February 2022 | Support |

<https://immersiveweb.dev>

Babylon.js vs Three.js

Quick comparison:

- Babylon.js: built-in WebXR helpers, TypeScript-first, feature-rich
- Three.js: manual WebXR setup, lighter core, more DIY flexibility
- Babylon trades size for convenience; Three trades boilerplate for control

Babylon.js vs Three.js

Both frameworks:

- Can be used to build WebXR applications
- Support VR controllers, hand tracking, and AR passthrough
- Can run on Meta Quest 2 using Quest controllers

Project Setup Basics

Development Environment:

- **Node.js + npm** — JavaScript runtime and package manager
- **TypeScript** — Typed superset of JavaScript
- **Vite** — Fast build tool and dev server
- **package.json** — Project dependencies and scripts
- **tsconfig.json** — TypeScript compiler configuration

Basic Workflow:

`npm install` — Install dependencies

`npm run dev` — Start development server

`npm run build` — Build for deployment

Babylon.js App Boilerplate

Engine + render loop setup:

```
^^I^^Iimport { Engine } from "@babylonjs/core";
^^I^^Iimport { App } from "./app";

^^I^^Iconst canvas = document.getElementById("renderCanvas");
^^I^^Iconst engine = new Engine(canvas, true);

^^I^^Iconst app = new App(engine);
^^I^^Iapp.createScene().then(scene => {
^^I^^Iengine.runRenderLoop(() => { scene.render(); });
^^I^^I});

^^I^^Iwindow.addEventListener("resize", () => engine.resize());
```

Babylon.js createScene Snippets

IPA1 basics:

Key steps:

- instantiate a new Scene with the engine
- add a quick console log for debugging
- create default camera and light helpers
- replace with specific cameras later (e.g., ArcRotateCamera)

WebXR Setup in Babylon.js

Adding WebXR to your scene:

```
~~I~~Iconst ground = MeshBuilder.CreateGround("ground", { ... });
~~I~~Iconst xr = await scene.createDefaultXRExperienceAsync({
~~I~~IfloorMeshes: [ground] // Enable teleportation
~~I~~I});
```

What you get automatically:

- VR/AR mode button in the UI
- Controller input handling (grip, trigger, buttons)
- Teleportation system (if floorMeshes provided)
- Hand tracking support (if device supports it)

WebXR Setup: Babylon.js vs Three.js

Compare with manual WebXR setup in Three.js:

```
^^I^^I// Three.js requires manual WebXR setup
^^I^^Iconst session = await navigator.xr.requestSession("immersive-vr");
^^I^^Isession.updateRenderState({
  ^^I^^I^^I^baseLayer: new XRWebGLLayer(session, gl)
})];
^^I^^Iconst referenceSpace = await session.requestReferenceSpace("local");

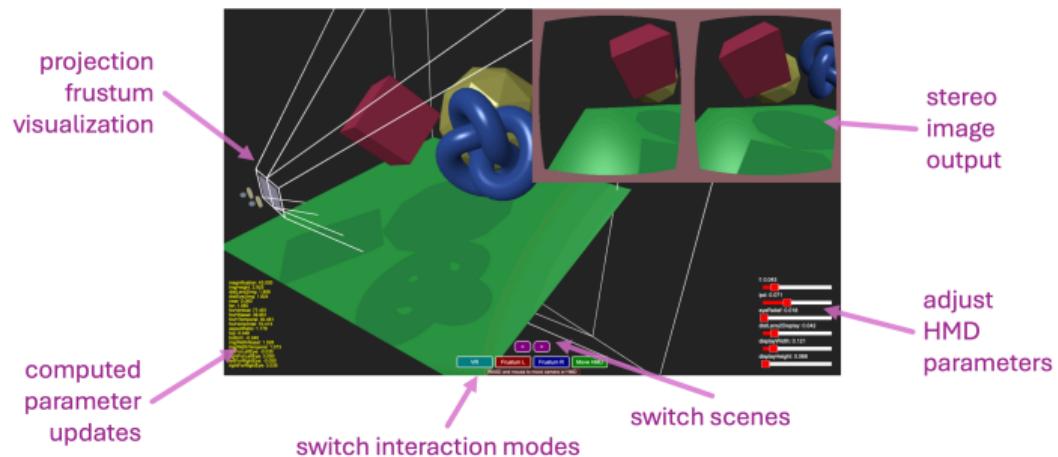
^^I^^I// Manual render loop with XR frame handling
^^I^^Isession.requestAnimationFrame(function onXRFrame(time, frame) {
  ^^I^^I^^Iconst pose = frame.getViewerPose(referenceSpace);
  ^^I^^I^^I^^I// Update camera matrices, render to each eye's viewport
  ^^I^^I^^I^session.requestAnimationFrame(onXRFrame);
})];
^^I^^I// No built-in controllers, teleportation, or hand tracking!
```

Key difference: Babylon.js = 3 lines vs Three.js = 20+ lines

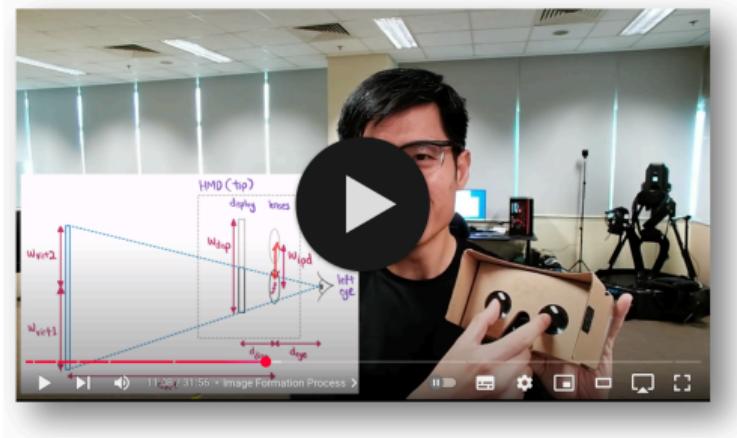
HMD Simulator

Purpose: Understanding Hardware-Software Connection

- Visualize how HMD hardware parameters affect rendering
- Experiment with lens properties, IPD, FOV, eye relief
- See real-time impact on the rendered view



Hardware & Software Connection

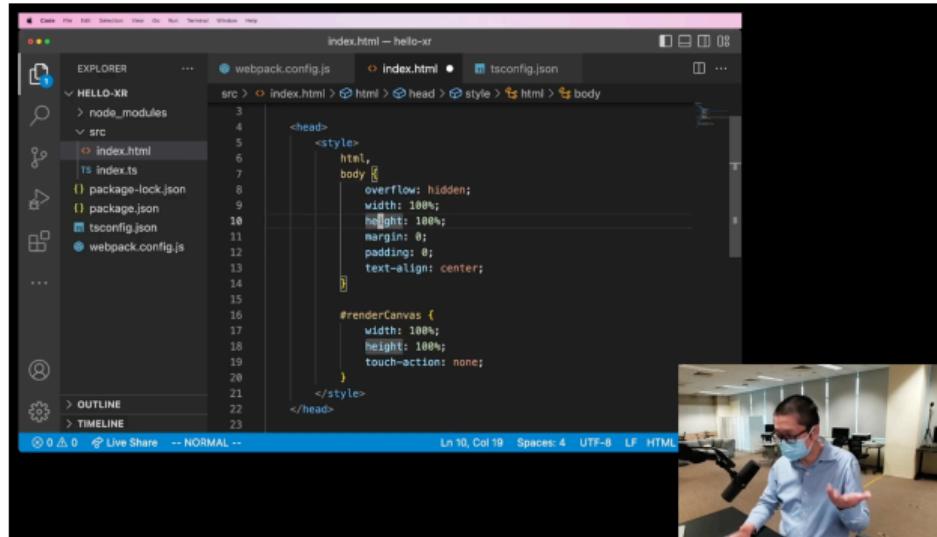


Stanford EE267 | <https://stanford.edu/class/ee267/lectures/lecture7.pdf>

Immersification Video | <https://youtu.be/OKD4jrnn4WE>

Video Tutorial

Live Coding Basics:



<https://youtu.be/iDCnmggNIy8>

Summary

Today we covered:

- Key graphics/XR APIs: WebXR, OpenXR, Vulkan, WebGL, DirectX, Metal
- Development frameworks: Unity, Unreal, Babylon.js, Three.js, A-Frame
- Project setup with Node.js, TypeScript, Vite
- HMD Simulator for hardware-software connection

Next Steps:

- Review project setup video tutorial
- Experiment with HMD Simulator
- Browse Stanford EE267 lecture notes on HMD optics

Further Reading

WebXR & OpenXR Standards:

- WebXR's Immersive Web Working Group
- WebXR Resources (immersiveweb.dev)
- OpenXR API (Khronos Group)

Development Frameworks:

- BabylonJS Documentation
- A-Frame Website
- Mozilla's Hello WebXR

See WEEK04 pre-class material for video tutorials and live coding sessions