

Immersive Systems I

Hardware & Software

Developing Immersive Applications

Created by: Chek Tien TAN



Learning Objectives:

- describe common software and hardware components across XR devices
- explain general system architecture for immersive applications
- describe the software component types in WebXR applications -
differentiate device capabilities and form factors

Common Software Components

At its core, an immersive application is a real-time interactive simulation.

1. Rendering System
2. Physics System
3. Input Handler
4. Audio Processor
5. AI System
6. etc.

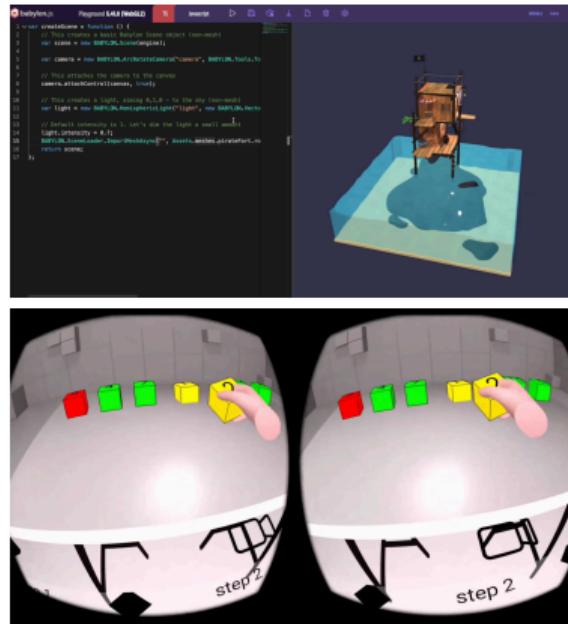
1. Rendering System

Responsibilities:

- Drawing graphics to the display
- Managing 3D models, textures, materials
- Configuring lighting and cameras
- Post-processing effects (bloom, DOF, etc.)
- **Stereo rendering** for VR

Under the Hood: Scene graph traversal, culling, clipping, shaders, rasterization

In WebXR: Babylon.js handles stereo rendering automatically.



Video: <https://youtu.be/OKD4jrnn4WE?t=1339>

2. Physics System

Responsibilities:

- Collision detection and response
 - Rigid body dynamics
 - Gravity and forces
 - Constraints and joints
 - Raycasting for interaction

In WebXR: Babylon.js 6.0+ integrates Havok (recommended) via WebAssembly. Also supports Cannon.js and Ammo.js.



```
babylon.js          Playground SALO    75    landscape      100% 100%
```

```
01 //Opening scene
02
03 const base = BABYLON.SceneLoader.create("base", "I\\wings_1", 1, http://
04 baseUrl, false, true, 0, 1000);
05
06 const camera = base.createCamera("camera", "I\\wings_1");
07 const light = base.createLight("light", "I\\wings_1");
08
09 const ground = base.createGround("ground", "I\\wings_1");
10
11 const box = base.createBox("box", "I\\wings_1");
12
13 const box2 = base.createBox("box2", "I\\wings_1");
14
15 const box3 = base.createBox("box3", "I\\wings_1");
16
17 const box4 = base.createBox("box4", "I\\wings_1");
18
19 const box5 = base.createBox("box5", "I\\wings_1");
20
21 const box6 = base.createBox("box6", "I\\wings_1");
22
23 const box7 = base.createBox("box7", "I\\wings_1");
24
25 const box8 = base.createBox("box8", "I\\wings_1");
26
27 const box9 = base.createBox("box9", "I\\wings_1");
28
29 const box10 = base.createBox("box10", "I\\wings_1");
30
31
32 const box11 = base.createBox("box11", "I\\wings_1");
33
34 const box12 = base.createBox("box12", "I\\wings_1");
35
36 const box13 = base.createBox("box13", "I\\wings_1");
37
38 const box14 = base.createBox("box14", "I\\wings_1");
39
40 const box15 = base.createBox("box15", "I\\wings_1");
41
42 const box16 = base.createBox("box16", "I\\wings_1");
43
44 const box17 = base.createBox("box17", "I\\wings_1");
45
46 const box18 = base.createBox("box18", "I\\wings_1");
47
48 const box19 = base.createBox("box19", "I\\wings_1");
49
50 const box20 = base.createBox("box20", "I\\wings_1");
51
52
53 const box21 = base.createBox("box21", "I\\wings_1");
54
55 const box22 = base.createBox("box22", "I\\wings_1");
56
57 const box23 = base.createBox("box23", "I\\wings_1");
58
59 const box24 = base.createBox("box24", "I\\wings_1");
60
61 const box25 = base.createBox("box25", "I\\wings_1");
62
63 const box26 = base.createBox("box26", "I\\wings_1");
64
65 const box27 = base.createBox("box27", "I\\wings_1");
66
67 const box28 = base.createBox("box28", "I\\wings_1");
68
69 const box29 = base.createBox("box29", "I\\wings_1");
70
71 const box30 = base.createBox("box30", "I\\wings_1");
72
73
74 const box31 = base.createBox("box31", "I\\wings_1");
75
76 const box32 = base.createBox("box32", "I\\wings_1");
77
78 const box33 = base.createBox("box33", "I\\wings_1");
79
80 const box34 = base.createBox("box34", "I\\wings_1");
81
82 const box35 = base.createBox("box35", "I\\wings_1");
83
84 const box36 = base.createBox("box36", "I\\wings_1");
85
86 const box37 = base.createBox("box37", "I\\wings_1");
87
88 const box38 = base.createBox("box38", "I\\wings_1");
89
90 const box39 = base.createBox("box39", "I\\wings_1");
91
92 const box40 = base.createBox("box40", "I\\wings_1");
93
94
95 const box41 = base.createBox("box41", "I\\wings_1");
96
97 const box42 = base.createBox("box42", "I\\wings_1");
98
99 const box43 = base.createBox("box43", "I\\wings_1");
100
101 const box44 = base.createBox("box44", "I\\wings_1");
102
103 const box45 = base.createBox("box45", "I\\wings_1");
104
105 const box46 = base.createBox("box46", "I\\wings_1");
106
107 const box47 = base.createBox("box47", "I\\wings_1");
108
109 const box48 = base.createBox("box48", "I\\wings_1");
110
111 const box49 = base.createBox("box49", "I\\wings_1");
112
113 const box50 = base.createBox("box50", "I\\wings_1");
114
115
116 const box51 = base.createBox("box51", "I\\wings_1");
117
118 const box52 = base.createBox("box52", "I\\wings_1");
119
120 const box53 = base.createBox("box53", "I\\wings_1");
121
122 const box54 = base.createBox("box54", "I\\wings_1");
123
124 const box55 = base.createBox("box55", "I\\wings_1");
125
126 const box56 = base.createBox("box56", "I\\wings_1");
127
128 const box57 = base.createBox("box57", "I\\wings_1");
129
130 const box58 = base.createBox("box58", "I\\wings_1");
131
132 const box59 = base.createBox("box59", "I\\wings_1");
133
134 const box60 = base.createBox("box60", "I\\wings_1");
135
136
137 const box61 = base.createBox("box61", "I\\wings_1");
138
139 const box62 = base.createBox("box62", "I\\wings_1");
140
141 const box63 = base.createBox("box63", "I\\wings_1");
142
143 const box64 = base.createBox("box64", "I\\wings_1");
144
145 const box65 = base.createBox("box65", "I\\wings_1");
146
147 const box66 = base.createBox("box66", "I\\wings_1");
148
149 const box67 = base.createBox("box67", "I\\wings_1");
150
151 const box68 = base.createBox("box68", "I\\wings_1");
152
153 const box69 = base.createBox("box69", "I\\wings_1");
154
155 const box70 = base.createBox("box70", "I\\wings_1");
156
157
158 const box71 = base.createBox("box71", "I\\wings_1");
159
160 const box72 = base.createBox("box72", "I\\wings_1");
161
162 const box73 = base.createBox("box73", "I\\wings_1");
163
164 const box74 = base.createBox("box74", "I\\wings_1");
165
166 const box75 = base.createBox("box75", "I\\wings_1");
167
168 const box76 = base.createBox("box76", "I\\wings_1");
169
170 const box77 = base.createBox("box77", "I\\wings_1");
171
172 const box78 = base.createBox("box78", "I\\wings_1");
173
174 const box79 = base.createBox("box79", "I\\wings_1");
175
176 const box80 = base.createBox("box80", "I\\wings_1");
177
178
179 const box81 = base.createBox("box81", "I\\wings_1");
180
181 const box82 = base.createBox("box82", "I\\wings_1");
182
183 const box83 = base.createBox("box83", "I\\wings_1");
184
185 const box84 = base.createBox("box84", "I\\wings_1");
186
187 const box85 = base.createBox("box85", "I\\wings_1");
188
189 const box86 = base.createBox("box86", "I\\wings_1");
190
191 const box87 = base.createBox("box87", "I\\wings_1");
192
193 const box88 = base.createBox("box88", "I\\wings_1");
194
195 const box89 = base.createBox("box89", "I\\wings_1");
196
197 const box90 = base.createBox("box90", "I\\wings_1");
198
199
200 const box91 = base.createBox("box91", "I\\wings_1");
201
202 const box92 = base.createBox("box92", "I\\wings_1");
203
204 const box93 = base.createBox("box93", "I\\wings_1");
205
206 const box94 = base.createBox("box94", "I\\wings_1");
207
208 const box95 = base.createBox("box95", "I\\wings_1");
209
210 const box96 = base.createBox("box96", "I\\wings_1");
211
212 const box97 = base.createBox("box97", "I\\wings_1");
213
214 const box98 = base.createBox("box98", "I\\wings_1");
215
216 const box99 = base.createBox("box99", "I\\wings_1");
217
218 const box100 = base.createBox("box100", "I\\wings_1");
219
220
221 const box101 = base.createBox("box101", "I\\wings_1");
222
223 const box102 = base.createBox("box102", "I\\wings_1");
224
225 const box103 = base.createBox("box103", "I\\wings_1");
226
227 const box104 = base.createBox("box104", "I\\wings_1");
228
229 const box105 = base.createBox("box105", "I\\wings_1");
230
231 const box106 = base.createBox("box106", "I\\wings_1");
232
233 const box107 = base.createBox("box107", "I\\wings_1");
234
235 const box108 = base.createBox("box108", "I\\wings_1");
236
237 const box109 = base.createBox("box109", "I\\wings_1");
238
239 const box110 = base.createBox("box110", "I\\wings_1");
240
241
242 const box111 = base.createBox("box111", "I\\wings_1");
243
244 const box112 = base.createBox("box112", "I\\wings_1");
245
246 const box113 = base.createBox("box113", "I\\wings_1");
247
248 const box114 = base.createBox("box114", "I\\wings_1");
249
250 const box115 = base.createBox("box115", "I\\wings_1");
251
252 const box116 = base.createBox("box116", "I\\wings_1");
253
254 const box117 = base.createBox("box117", "I\\wings_1");
255
256 const box118 = base.createBox("box118", "I\\wings_1");
257
258 const box119 = base.createBox("box119", "I\\wings_1");
259
260 const box120 = base.createBox("box120", "I\\wings_1");
261
262
263 const box121 = base.createBox("box121", "I\\wings_1");
264
265 const box122 = base.createBox("box122", "I\\wings_1");
266
267 const box123 = base.createBox("box123", "I\\wings_1");
268
269 const box124 = base.createBox("box124", "I\\wings_1");
270
271 const box125 = base.createBox("box125", "I\\wings_1");
272
273 const box126 = base.createBox("box126", "I\\wings_1");
274
275 const box127 = base.createBox("box127", "I\\wings_1");
276
277 const box128 = base.createBox("box128", "I\\wings_1");
278
279 const box129 = base.createBox("box129", "I\\wings_1");
280
281 const box130 = base.createBox("box130", "I\\wings_1");
282
283
284 const box131 = base.createBox("box131", "I\\wings_1");
285
286 const box132 = base.createBox("box132", "I\\wings_1");
287
288 const box133 = base.createBox("box133", "I\\wings_1");
289
290 const box134 = base.createBox("box134", "I\\wings_1");
291
292 const box135 = base.createBox("box135", "I\\wings_1");
293
294 const box136 = base.createBox("box136", "I\\wings_1");
295
296 const box137 = base.createBox("box137", "I\\wings_1");
297
298 const box138 = base.createBox("box138", "I\\wings_1");
299
299
```

3. Input Handler

Responsibilities:

- Reading controller button presses
- Tracking hand/controller positions
- Processing gestures and interactions
- Managing input events
- Supporting multiple input devices

In WebXR: WebXR Device API provides standardized access to VR/AR controllers and hand tracking.

WebXR Input: XRSession & XRInputSource

Accessing XR input sources:

```
// Get XR session from Babylon.js
const xr = await scene.createDefaultXRExperienceAsync();
const session = xr.baseExperience.sessionManager.session; // XRSession

// Listen for input source changes (XRInputSource events)
session.addEventListener('inputsourceschange', (event) => {
  for (const inputSource of event.added) { // inputSource is XRInputSource
    console.log('Input added:', inputSource.handedness); // 'left'/'right'
    console.log('Target ray mode:', inputSource.targetRayMode);
  }
});
```

Key WebXR input interfaces:

- `XRSession` — manages the XR experience lifecycle
- `XRInputSource` — represents controllers, hands, gaze
- `XRInputSourceArray` — collection of active inputs

4. Audio Processor

Responsibilities:

- Spatial audio, e.g., Head-Related Transfer Function (HRTF) models how sound waves interact with the head and ears to create realistic directional audio cues)
- Sound effects playback
- Music and ambient audio
- Audio attenuation with distance

In WebXR: Web Audio API provides spatial audio capabilities for immersive sound experiences.

5. AI System

Responsibilities:

- NPC behavior and decision-making
- Pathfinding (A* algorithm, navigation meshes)
- Behavior trees for complex AI
- Procedural content generation
- Adaptive difficulty

In WebXR: Babylon.js provides **navigation mesh** (navmesh) and crowd agents for autonomous NPC pathfinding.

Note: Not all immersive applications need AI — it depends on your design goals.

Component Roles in EDIS



Experience Dementia in Singapore

What components do you need here:

- **Rendering** — displays the 360° kitchen environment
- **Input Handler** — gaze-based navigation between hotspots
- **Audio** — plays ambient sounds and caregiver narration
- **Physics** — is there any?
- **AI** — Scripted linear experience

Designing Your Architecture

Don't just use every component—design your unique mix:

1. Understand Your Requirements

- What experience are you creating? (game, training, visualization, social)
- What interactions does your design demand?
- What are your performance constraints?

2. Research Available Options

- Explore existing frameworks, libraries, and patterns
- Learn from similar applications—what did they use?
- Watch tutorials, read documentation, study code samples

3. Curate Your Architecture

- Select only the components that serve your design
- Avoid “kitchen sink” syndrome—every system adds complexity
- Create custom components when existing ones don’t fit

Entity Component System (ECS)

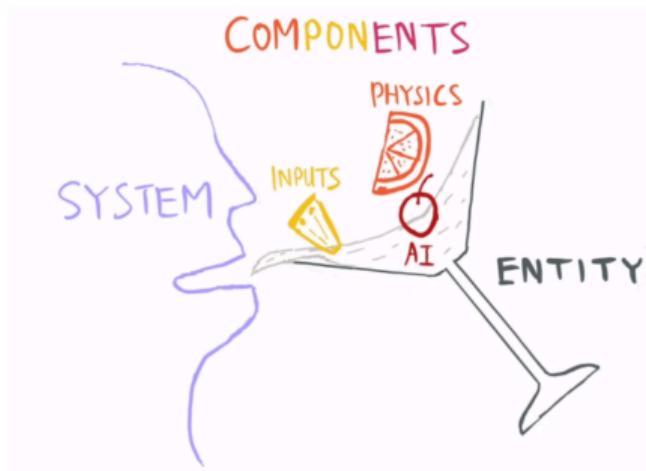
Modern simulations/games often use **ECS architecture**

Traditional OOP approach:

- Objects inherit from base classes
- Tight coupling between behaviors
- Hard to add new behaviors dynamically

ECS approach:

- **Entities** — unique IDs (player, enemy, light)
- **Components** — data containers (position, mesh, physics)
- **Systems** — logic processors (rendering, physics, AI)



Why ECS Over OOP?

1. Flexibility

- Add/remove behaviors at runtime
- No rigid inheritance hierarchies

2. Performance

- Data-oriented design enables cache-friendly processing
- Systems process components in batches

3. Maintainability

- Clear separation of data and logic
- Easier to test and debug
- Teams work on different systems independently

Top reason: Composition enables flexible, reusable, performant code

A-Frame: ECS in Practice

A-Frame is a WebXR framework built on top of Three.js that implements ECS

Entities (HTML elements):

<a-entity> — empty container, just an ID

Components (HTML attributes):

position="1 2 3" geometry="primitive: box" material="color: red"

Systems (JavaScript):

```
AFRAME.registerSystem('game', {init: fn, tick: fn});
```

Key insight: HTML attributes = Components, making ECS intuitive for web developers

A-Frame: Defining and Using Components

Defining a custom component (JavaScript):

```
AFRAME.registerComponent('spin', {  
  schema: { speed: { default: 1 } },  
  tick: function (time, delta) {  
    this.el.object3D.rotation.y += this.data.speed * delta / 1000;  
  }  
});
```

Attaching components to an entity (HTML):

```
<a-entity  
  geometry="primitive: box"  
  material="color: red"  
  position="0 1.5 -3"  
  spin="speed: 2">  
</a-entity>
```

Key: Components are data (attributes), entities are containers, systems process them.

Snap Lens Studio: AR Development Platform

Architecture: Visual editor + TypeScript scripting engine

- Component-based: Scripts attach to Scene Objects
- Event-driven lifecycle (onAwake, onUpdate, onDestroy)
- Deploys to Snapchat app and Spectacles AR glasses

TypeScript Example:

```
@component
export class TapHandler extends BaseScriptComponent {
  @input sceneObject: SceneObject;
  onAwake() {
    this.createEvent('TapEvent').bind(() => this.onTap());
  }
  onTap() {
    this.sceneObject.enabled = !this.sceneObject.enabled;
  }
}
```



Source: AWE Asia 2026, Snap Engineer Atley Loughridge

Babylon.js: Bring Your Own Architecture

Babylon.js provides powerful tools but **no enforced architecture**

What Babylon.js provides:

- Scene graph with meshes, cameras, lights
- Component-like behaviors (attachable to meshes)
- Observables for event handling, WebXR session management

What developers must design:

- How to structure logic (classes, modules, ECS?)
- How to manage state and separate concerns

Result: More flexibility, but requires conscious architectural decisions

WebXR Architecture: A-Frame vs Babylon.js

A-Frame (ECS built-in)	Babylon.js (BYO Architecture)
Declarative HTML approach	Programmatic JavaScript approach
ECS architecture enforced	Architecture up to developer
Easier for beginners	More control for experts
Three.js under the hood	Direct WebGL/WebGPU rendering
Smaller community, fewer resources	Larger community, extensive docs
Good for rapid prototyping	Good for complex applications

Bottom line: A-Frame gives you structure; Babylon.js gives you freedom – choose based on team experience and project needs.

WebXR Application Structure

Typical WebXR app structure:

1. Scene Graph

- Hierarchical tree of objects, parent-child transformations

2. Rendering Loop

- `requestAnimationFrame` or `XRSession.requestAnimationFrame`
- Update logic → render frame (60–90+ FPS)

3. Component System

- Meshes, materials, lights, cameras, physics bodies
- Custom behaviors and scripts attached to objects

XR Rendering: Time Budget

Why frame timing matters in XR:

Target FPS	Time per frame	Typical use
60 FPS	16.7ms	Desktop VR, Mobile AR
72 FPS	13.9ms	Quest default mode
120 FPS	8.3ms	High-end PC VR

Each frame must complete:

- Input processing (controller, hand tracking)
- Physics simulation and collision detection
- Game logic / AI updates
- **Stereo rendering** (render twice — left & right eye!)

Miss the deadline → dropped frames → motion sickness!

From Software to Hardware

Now that we understand software architecture, let's explore the hardware that runs these immersive applications.

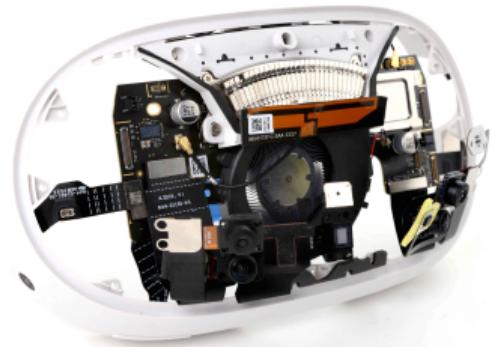
Key question: What physical components make XR possible?

- What's inside a VR headset?
- How do lenses create the VR illusion?
- What different types of AR devices exist?

Inside an HMD: Meta Quest 3

Key Hardware Components:

- Dual LCD displays (2064 × 2208 per eye, 120Hz)
- Pancake lenses — thinner/clearer than fresnel lenses
- Touch Plus controllers
- Motion tracking sensors
- Cameras (inside-out tracking, passthrough, depth sensor)
- Snapdragon XR2 Gen 2, 8GB RAM



Source: iFixit Meta Quest 3 Teardown — ifixit.com/News/84572

Smartphone vs HMD Hardware

Similarities:

- Display screen, CPU/GPU, Motion sensors (IMU), Cameras, Battery

Key Difference: **Magnifying Lenses**

- HMDs use lenses to magnify the close display
- Creates wide field of view, enables virtual image at optical infinity
- Smartphone cardboard viewers add lenses to phones

Without lenses, you'd just see a tiny screen up close — lenses create immersion.

AR Hardware Landscape

Augmented Reality devices come in various form factors:

- 1. AR Glasses** — lightweight, everyday wearable
- 2. AR Headsets** — higher capability, professional use
- 3. Mobile AR** — smartphone/tablet cameras

Let's explore recent AR hardware across these categories

AR Glasses

Meta Ray-Ban Smart Glasses

- Form: Consumer glasses with integrated electronics (49g)
- Features: Cameras, speakers, mic, AI
- Display: *No display*



Meta Ray-Ban Meta display

- Form: Consumer glasses with micro-display
- Features: Same as before but with EMG input
- Display: Micro-LED monocular display



Xreal Air / Air 2 Pro

- Form: Consumer glasses with integrated virtual display (tethered)
- Features: 130" screen, USB-C
- Display: 1080p OLED, 46–49° FOV

Sources: meta.com, xreal.com, ray-ban.com

AR H/W Example: Snap Spectacles (5th Gen)

Positioned between smart glasses and full AR headsets

- Form: Standalone untethered glasses (226g)
- Display: 46° FOV stereo waveguide, LCoS projectors
- Compute: Dual Snapdragon processors, 45 min battery
- Input: Hand tracking, voice, 6DoF inside-out tracking
- 37 PPD, 13ms motion-to-photon latency

Takeaway: True stereo AR in glasses form factor — sits between smart glasses (Meta Ray-Ban) and full-blown AR headsets (HoloLens).



AR vs VR Hardware: Key Differences

Component	AR (e.g., HoloLens, Spectacles)	VR (e.g., Quest 3)
Display Tech	Waveguide/combiner (see-through)	LCD/OLED panels (opaque)
Optics	Diffractive waveguide, birdbath	Fresnel or pancake lenses
FOV	40–70° (limited by waveguide)	90–120° (larger panels possible)
Processor	Mobile SoC (power constrained)	XR-optimized SoC (more headroom)
Cameras	RGB + depth for world understanding	Tracking + passthrough + depth
Depth Sensing	Often ToF or structured light	ToF or stereo depth
Control Input	Hand tracking, gaze, voice	Controllers, hand tracking
Weight Target	<300g (glasses form factor goal)	400–600g (headset acceptable)

Key insight: AR requires see-through optics (harder, narrower FOV) while VR uses simpler opaque displays. AR prioritizes lightweight form; VR prioritizes immersion.

AR Headsets

Microsoft HoloLens 2

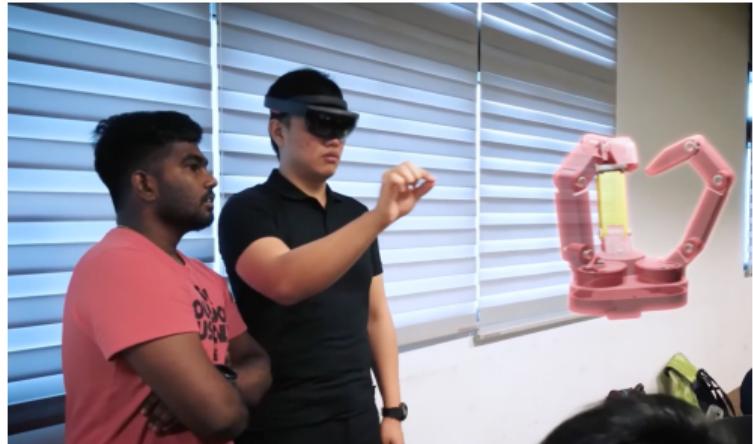
- Form: Optical see-through AR headset
- Features: Hand tracking, enterprise focus
- Display: 52° FOV, waveguide, standalone

Magic Leap 2

- Form: Optical see-through AR headset
- Features: Eye tracking, enterprise/medical
- Display: 70° FOV, waveguide with dimming

Apple Vision Pro

- Form: Video passthrough spatial computer
- Features: Eye + hand tracking
- Display: 23M pixels, micro-OLED



Sources: microsoft.com, magicleap.com, apple.com

AR Hardware: Key Differences

Form Factor Trade-offs:

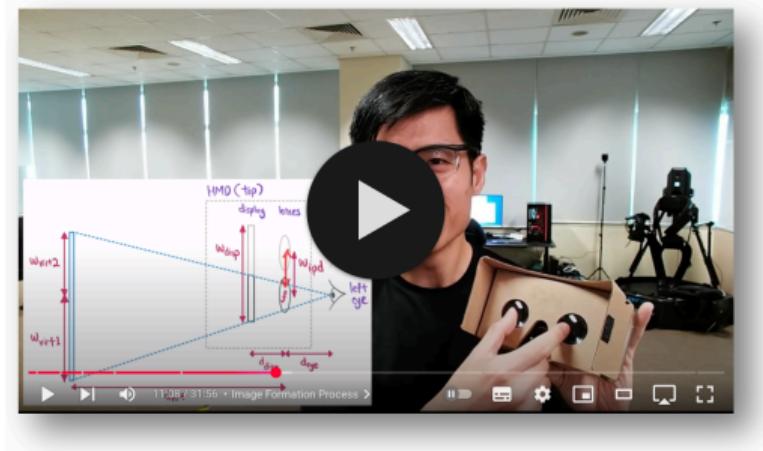
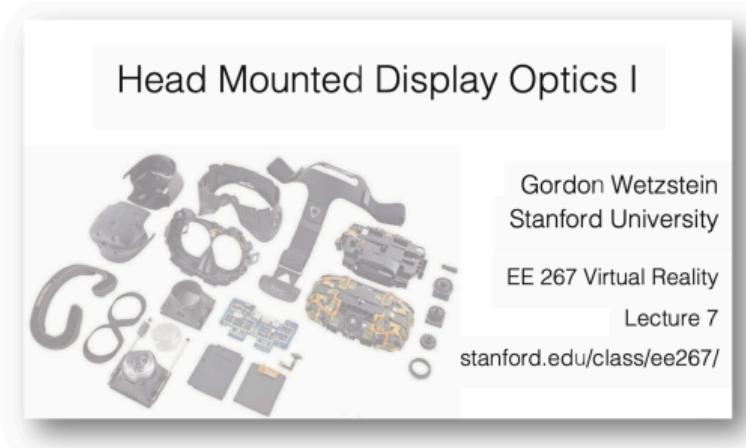
- **Glasses** — lightweight, socially acceptable, limited capability
- **Headsets** — more powerful, wider FOV, bulkier
- **Mobile** — ubiquitous, no extra hardware, limited tracking

Display Technology:

- **Optical see-through** — direct view + overlay (HoloLens, Magic Leap)
- **Video passthrough** — cameras + digital merge (Vision Pro, Quest 3)
- **Virtual display** — screen projection only (Xreal, Ray-Ban)

Use Cases: Enterprise (training, remote assist), Consumer (entertainment), Medical/Industrial

Connecting Hardware to Software

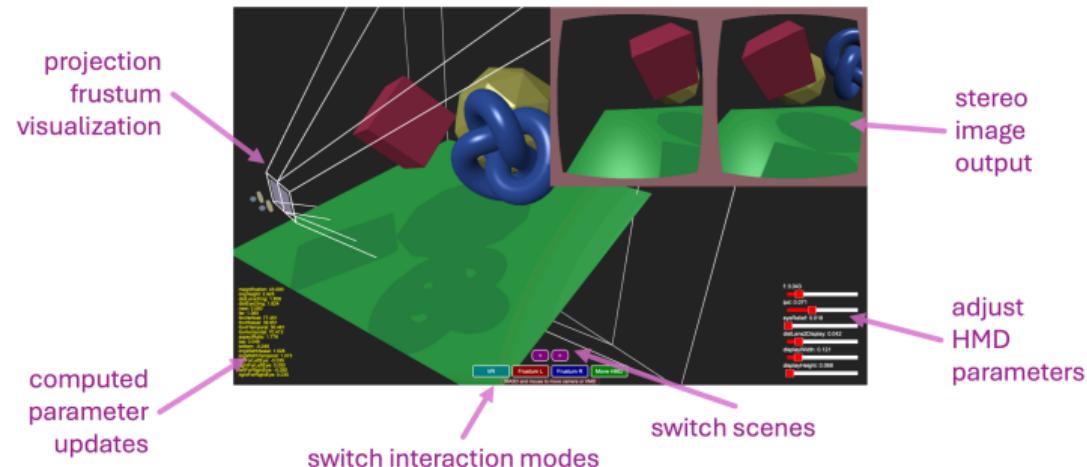


Stanford EE267: <https://stanford.edu/class/ee267/lectures/lecture7.pdf>
Immersification Video: <https://youtu.be/OKD4jrnn4WE>

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



Summary

Today we covered:

- Software components (Rendering, Physics, Input, Audio, AI)
- ECS architecture and its advantages over OOP
- A-Frame vs Babylon.js architectural approaches
- HMD hardware components (Quest 2 example)
- AR hardware landscape (glasses, headsets, mobile)

Next: Week06 – Deep dive into HMD optics theory