KOSSCON

IMMERSIVE WEB

BYUNGKWON KO
codeimpl@gmail.com

WHO AM I?

Samsung Electronics
Chromium/Blink Member
KOSSLAB Researcher

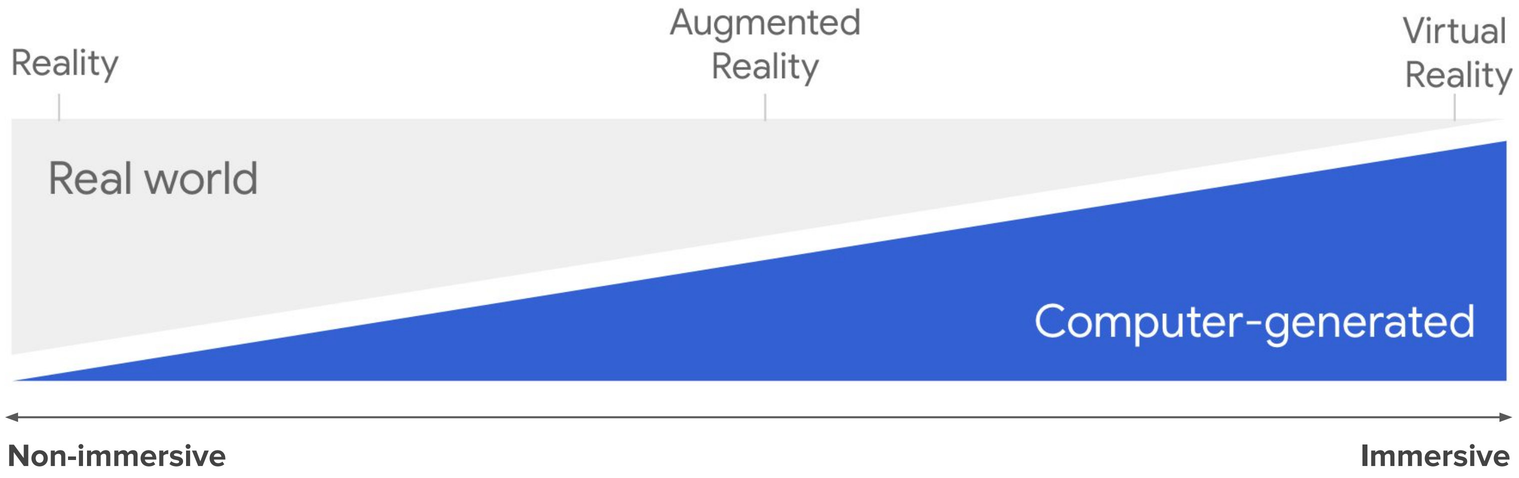
CONTENTS

IMMERSIVE WEB

What’s the **Immersive Web**?

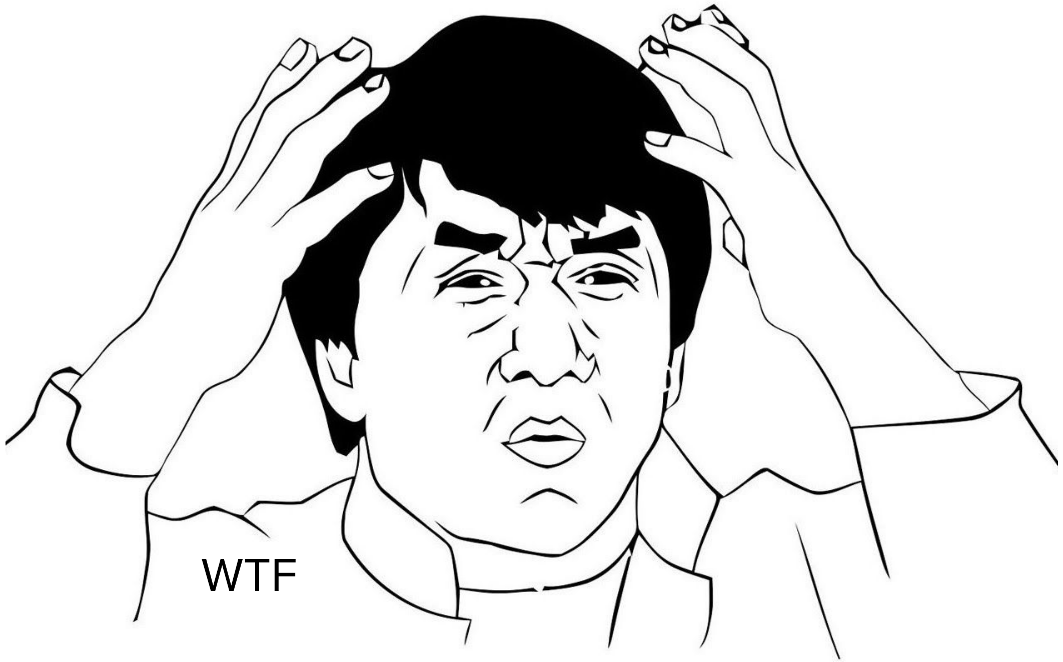
The **immersive web** means **virtual world experiences** hosted through the browser.

Immersive..


Reality
Augmented Reality
Virtual Reality
Real world
Computer-generated
Non-immersive
Immersive

The specification says that..

- Non-immersive
  - Sessions are considered non-immersive (sometimes referred to as inline) if their **output is displayed as an element in an HTML document.**
- Immersive
  - A session is considered to be an immersive session if it’s **output is displayed to the user in a way that makes the user feel the content is present in the same space with them**, shown at the proper scale.


WTF

0 or **1**

Immersive == Fully immersive == VR

Non-Immersive == Less Immersive == AR

WEB XR HISTORY

Do you remember **WebVR**?

It works well in most of modern browsers

BUT there are some problems

WebVR 2.0

and…

WebAR

WebAR is **very similar** to WebVR

WebAR + WebVR + ... + = **Web XR**

What’s in the new API?

- **Enables AR functionality**
- Better forwards compatibility
- Cleaner, more consistent, more predictable
- Enables more optimizations by the browser

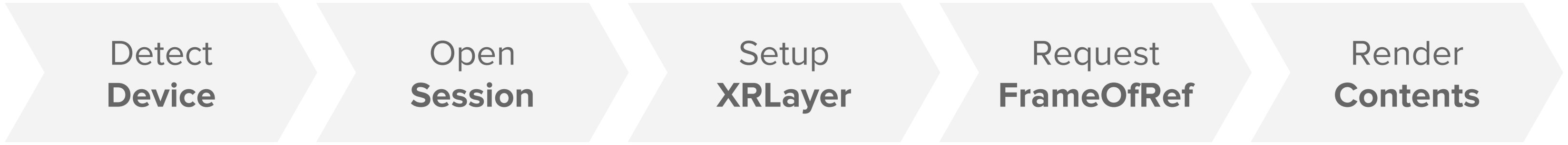
WEB XR DEVICE API

WebXR Device API

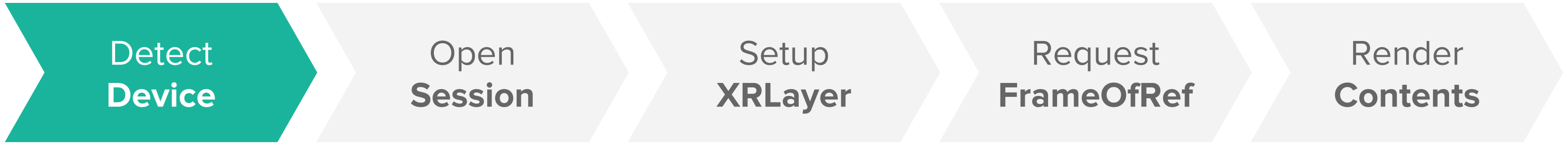
(WebAR + WebVR)

WebXR Device API is not GFX Feature.

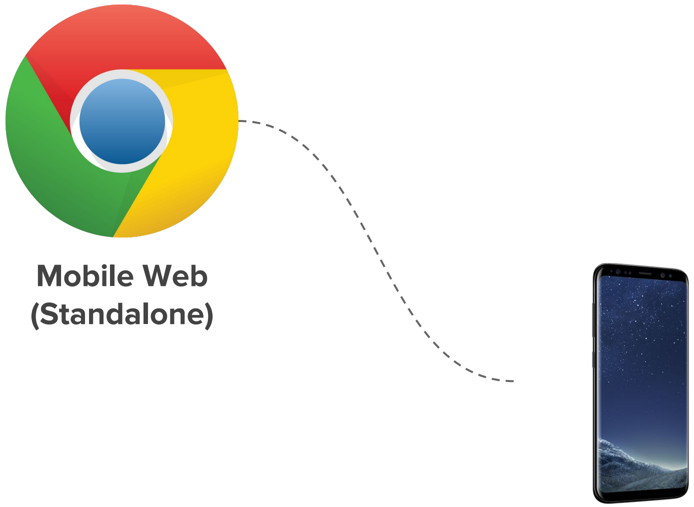
How WebXR works

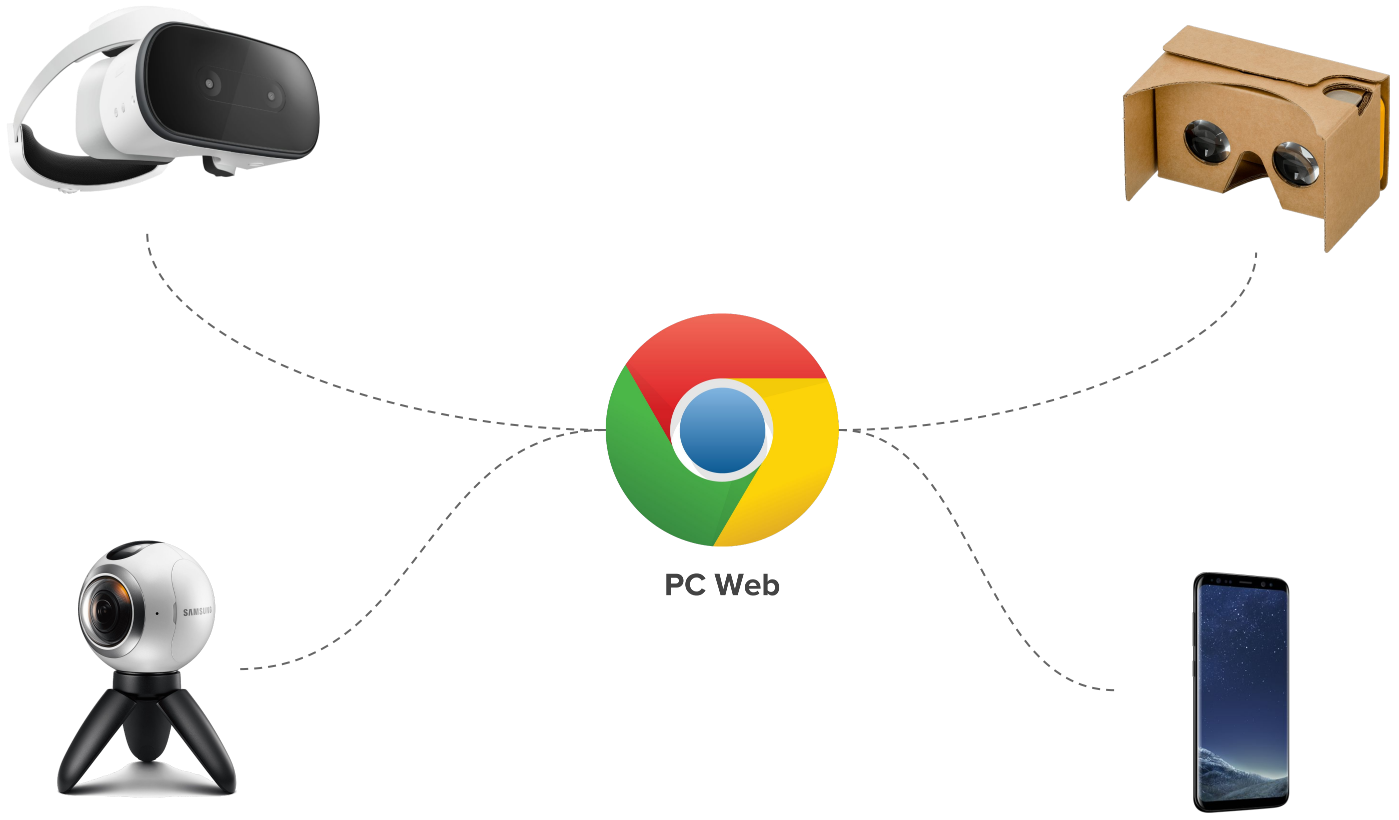


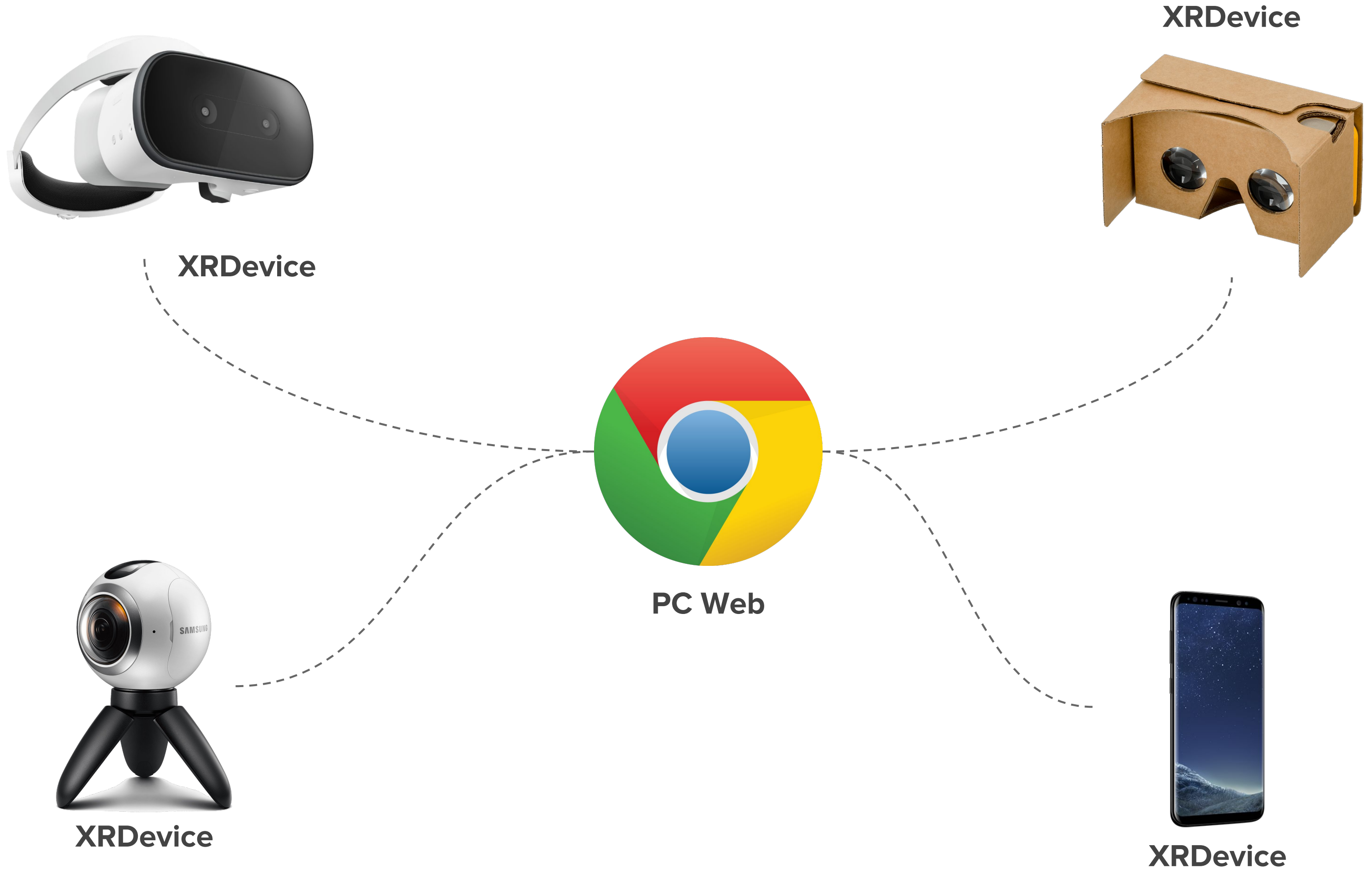
STEP1: Detect Device

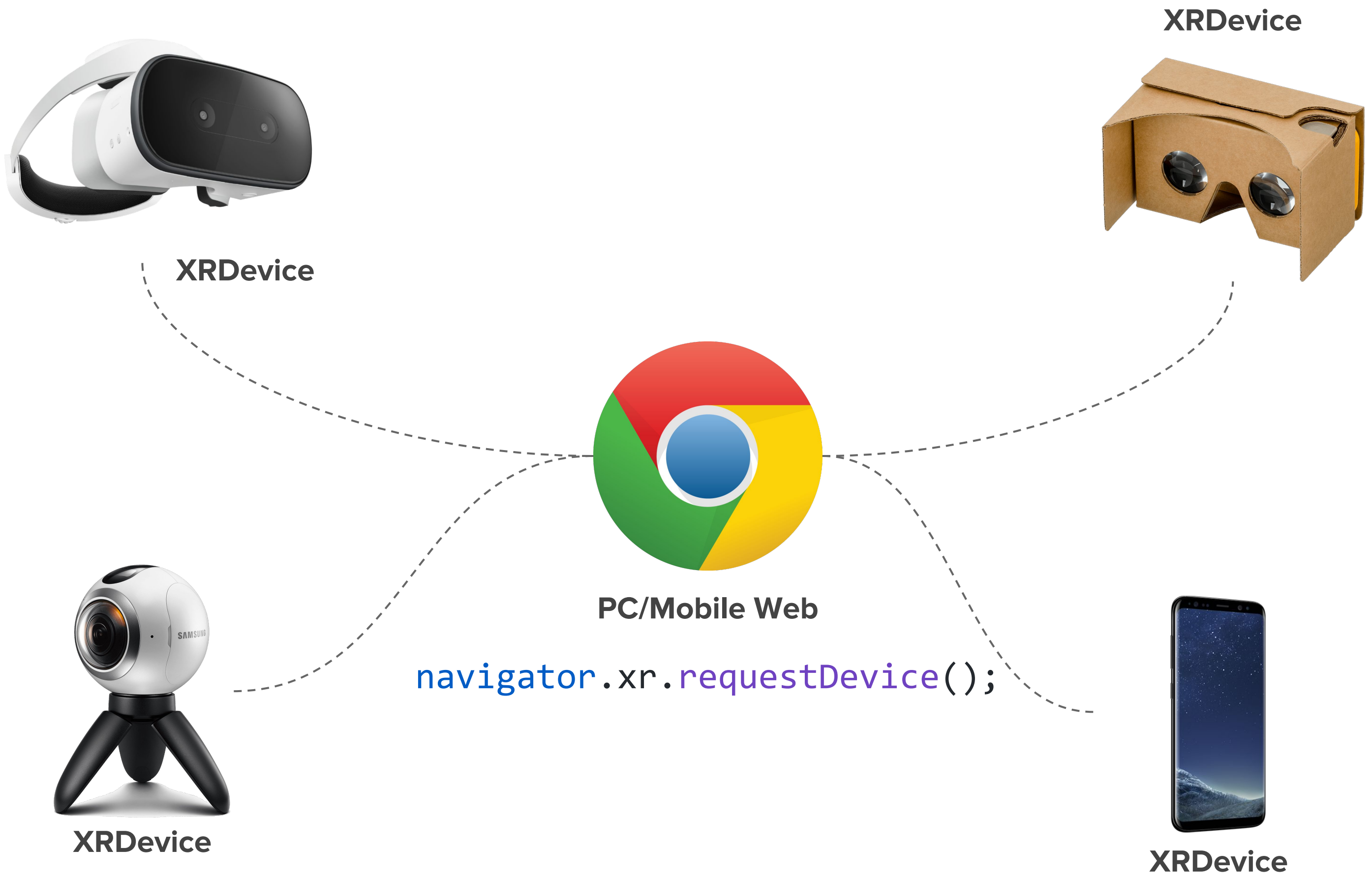


```
navigator.xr.requestDevice();
```

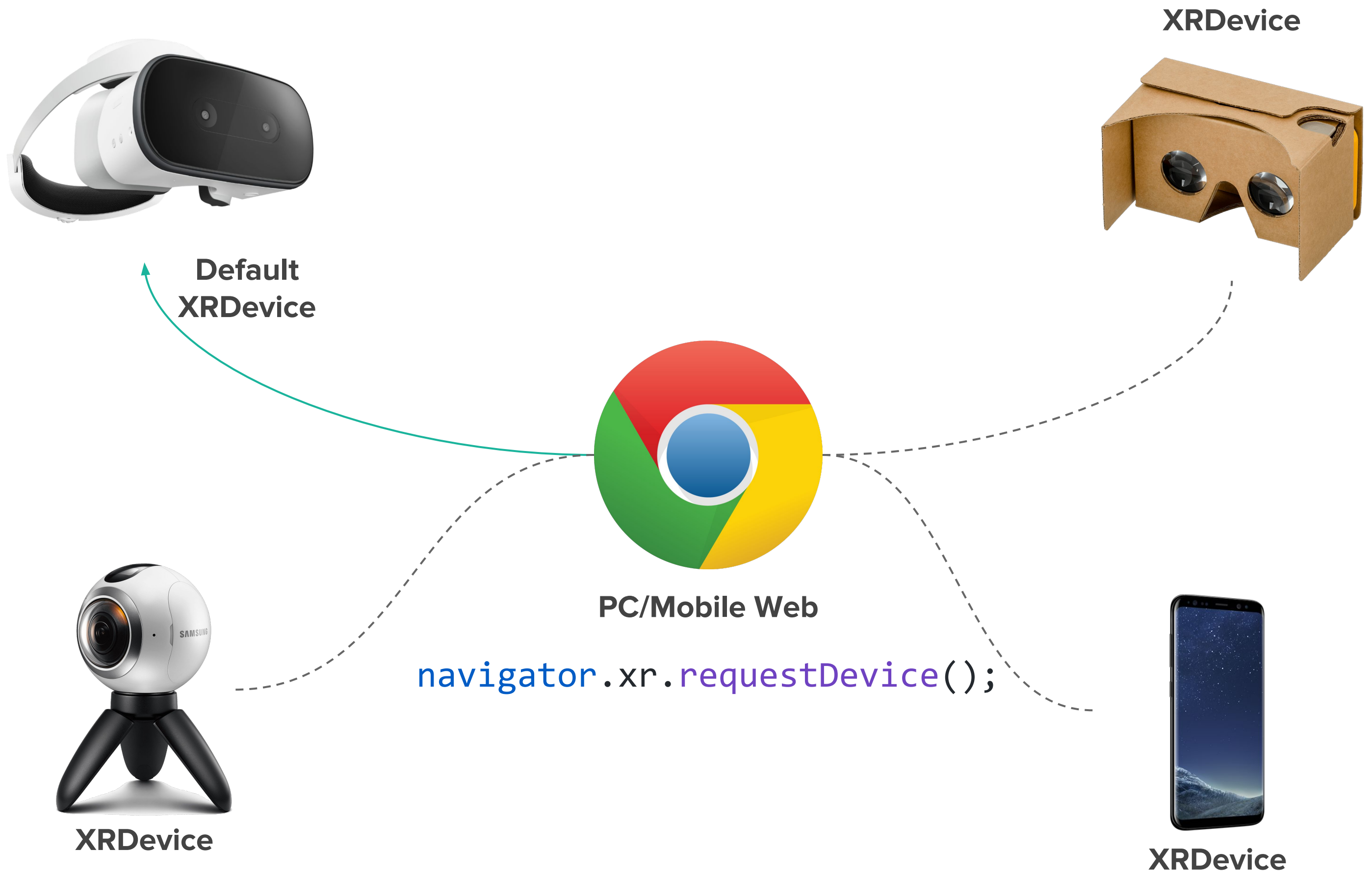

Mobile Web
(Standalone)


SAMSUNG
PC Web


XRDevice
XRDevice
PC Web
XRDevice
XRDevice



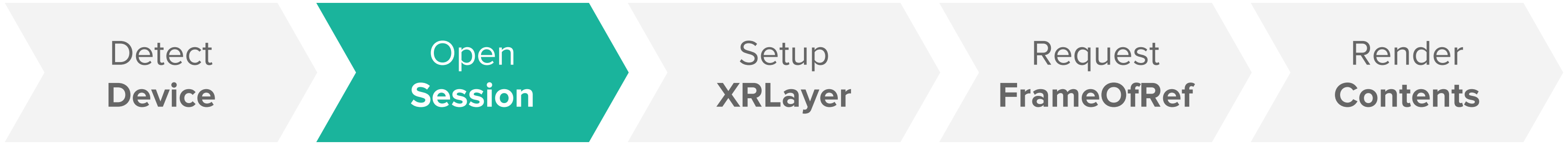
```
navigator.xr.requestDevice();
```



```
navigator.xr.requestDevice();
```

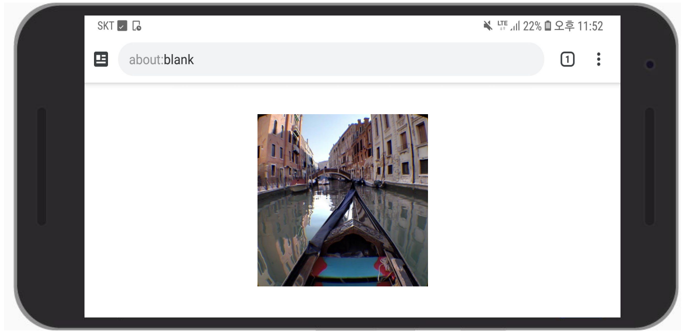
XRDevice is just an abstraction of available XR device

STEP2: Open Session

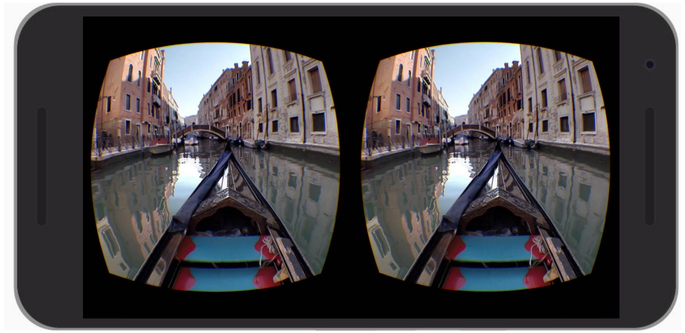


Open **a session(s)**
to **interact with** an XR device

```
device.requestSession();
```



```
device.requestSession({
    immersive: false
});
```



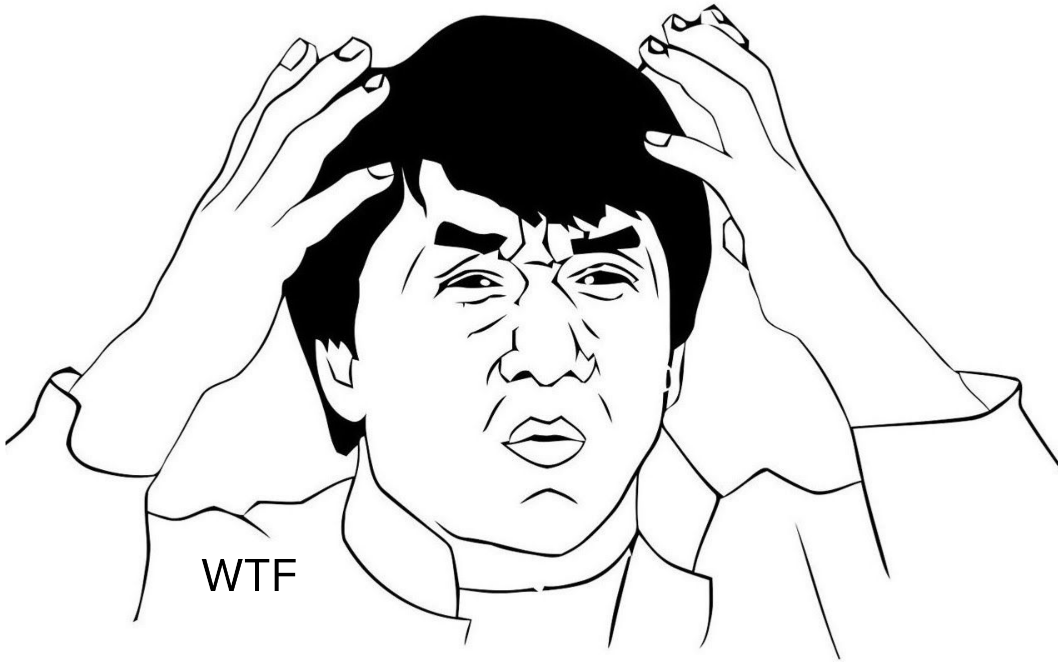
```
device.requestSession({
    immersive: true
});
```

```
device.requestSession({ immersive: true });
```

```
SecurityError: The requested session requires user activation.
```

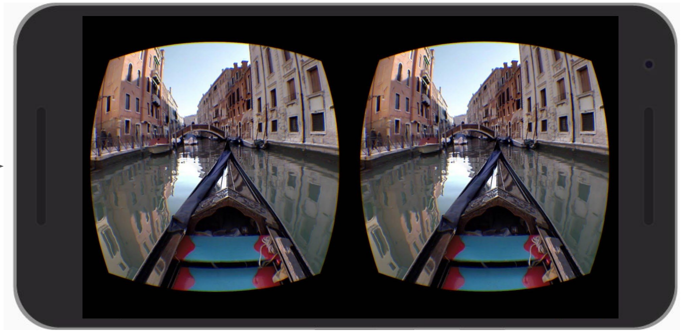
User Activation

- `event.isTrusted` should be `true`
- event’s type is one of:
    - change
    - click
    - contextmenu
    - dblclick
    - mouseup
    - pointerup
    - reset
    - submit
    - touchend

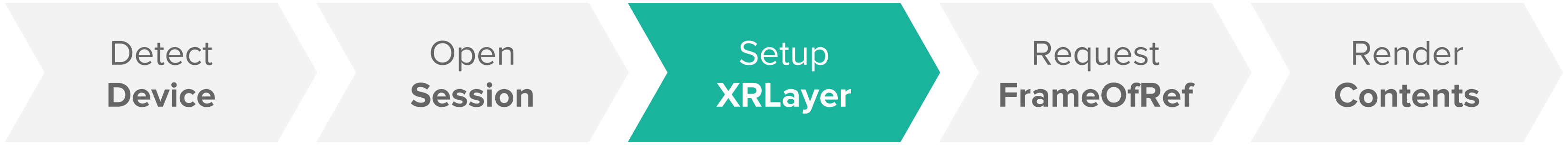

WTF

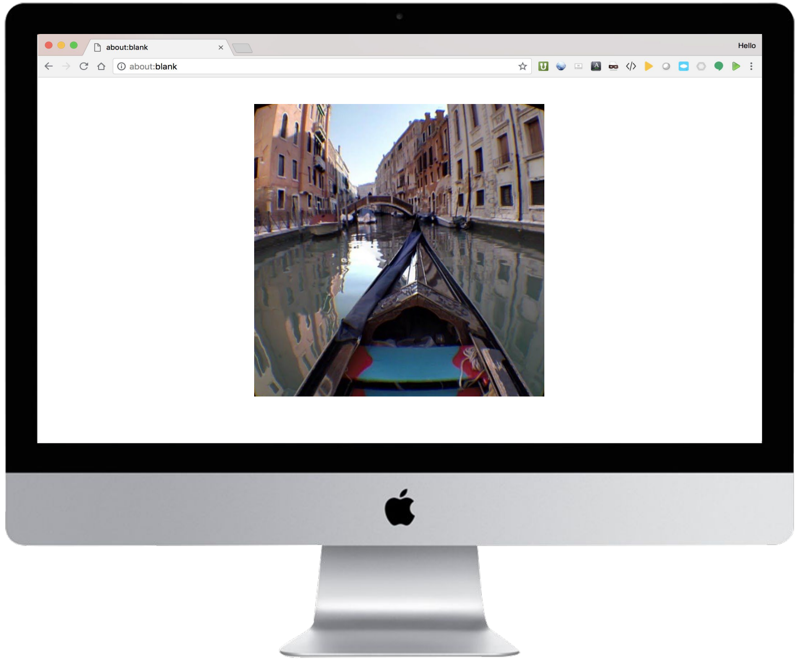
It requires **User Interaction**

```
device.requestSession({
    immersive: true
});
```

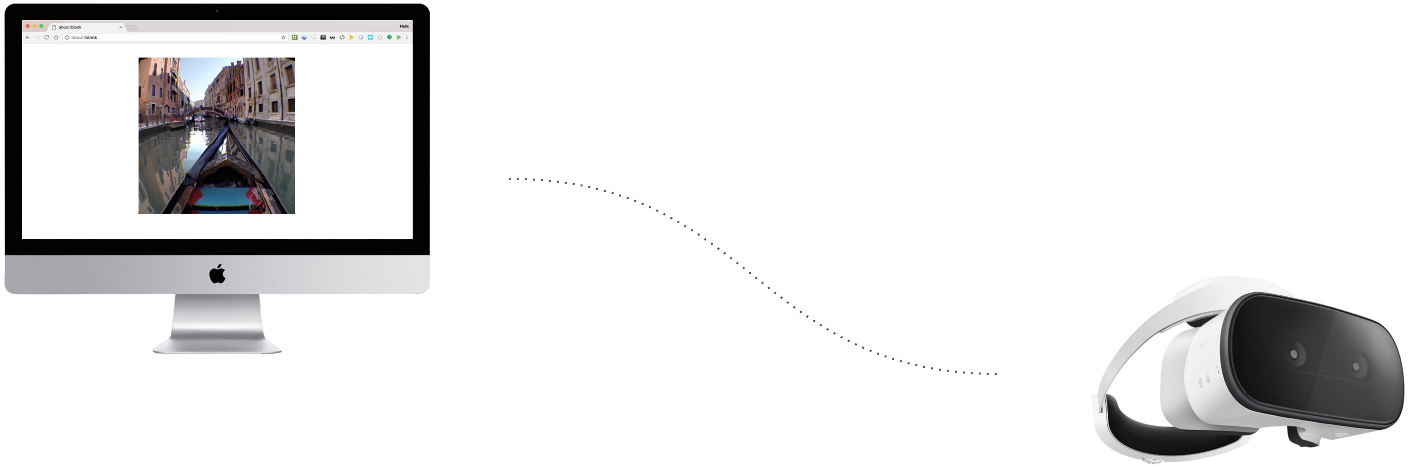


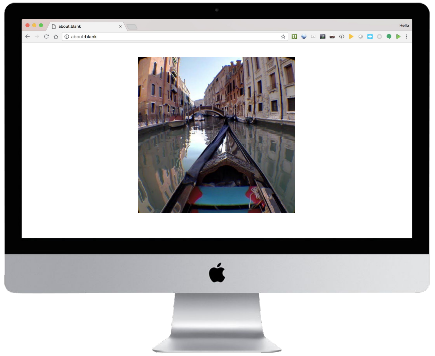
STEP3: Setup XRLayer



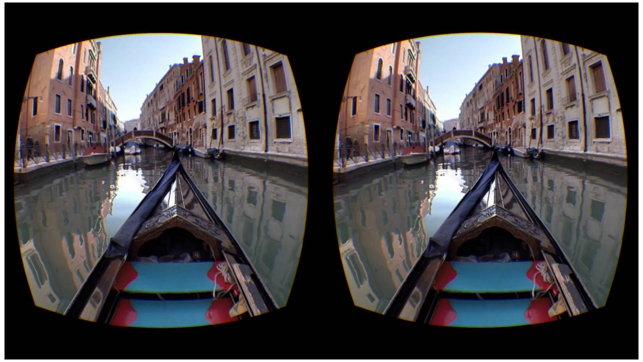

about:blank
about:blank
Hello

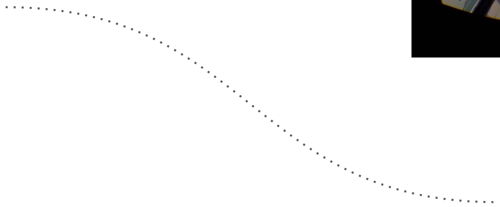
```
<canvas></canvas>
```


about:blank
Hello

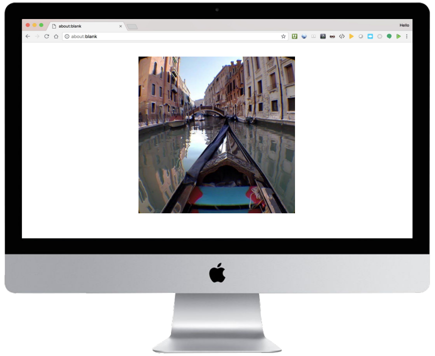


XRLayer



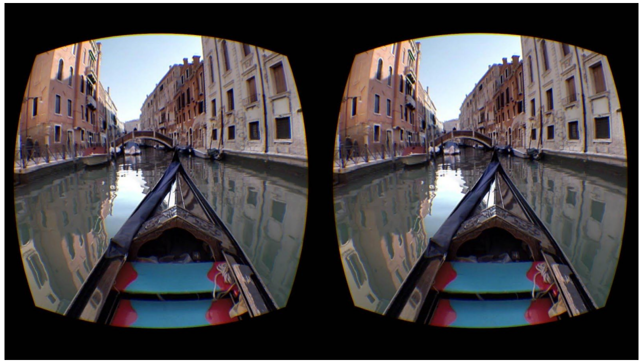


```
session.baseLayer = new XRWebGLLayer(session, gl);
```

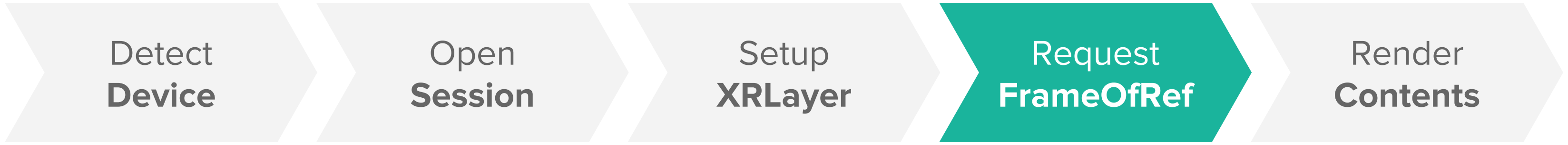


`drawScene(gl);`

XRWebGLLayer



STEP4: Request FrameOfReference



What’s the **FrameOfReference**?

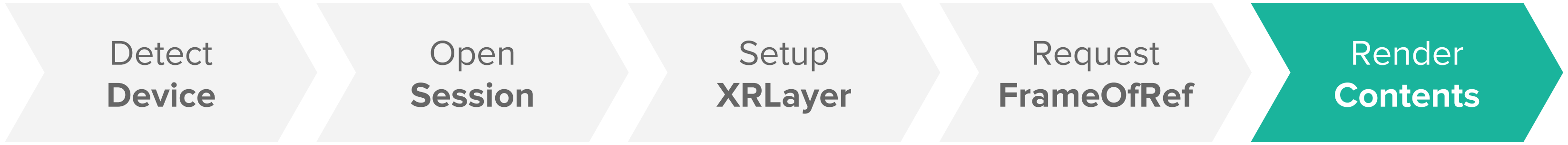
Frame Of Ref == **Ref Coordinate System**

Frame Of Reference Types

- head-model
  - The origin is approximately the location of the viewer's head and does not change if the viewer moves.
- eye-level
  - The origin is the viewer's head and moves with the viewer.
- stage
  - The origin is implied to be the center of the room at floor level and does not change if the viewer moves.

```
session.requestFrameOfReference(‘eye-level’);
```

STEP5: Render Contents



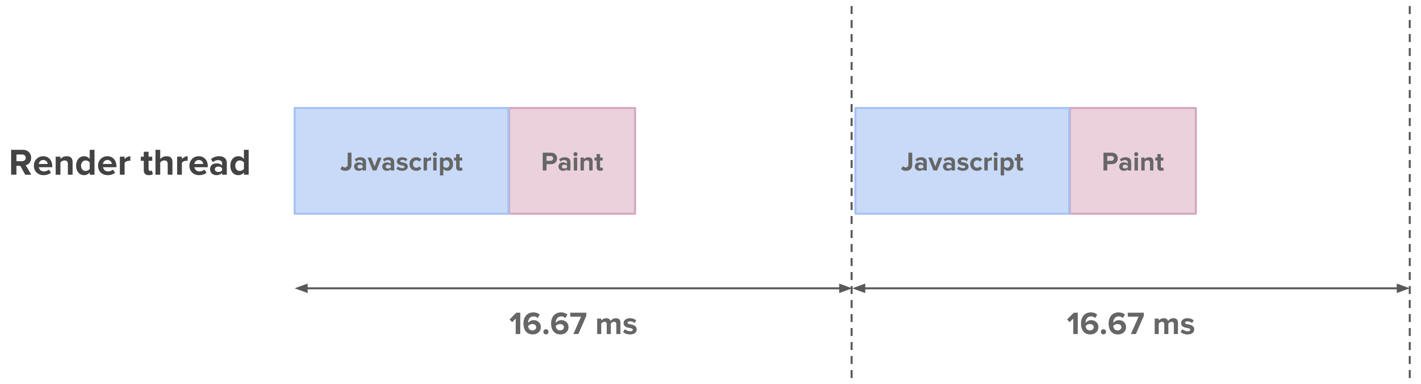
Just draw Canvas by using **WebGL**!

```
requestAnimationFrame(onDrawFrame);
```

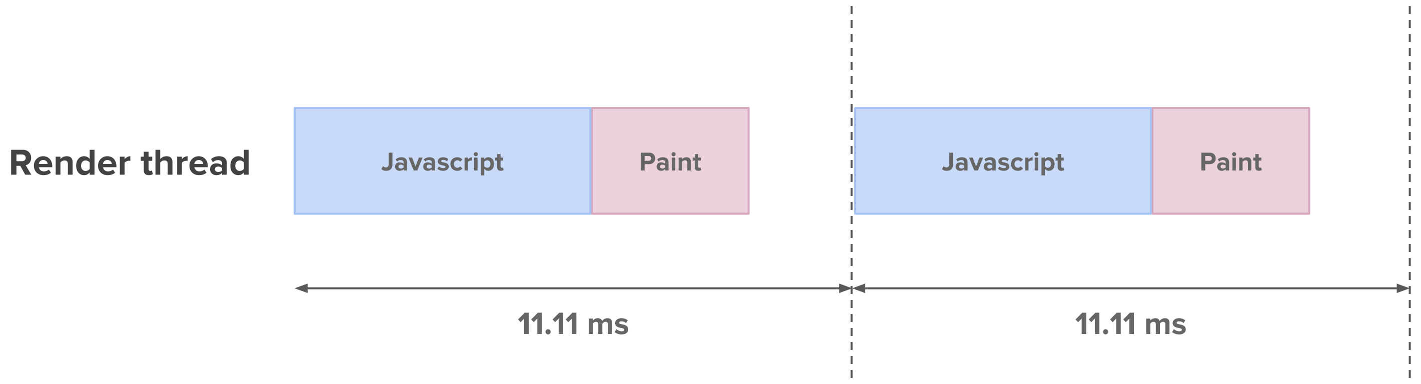
60 MHz

90 MHz

window.requestAnimationFrame()



session.requestAnimationFrame()

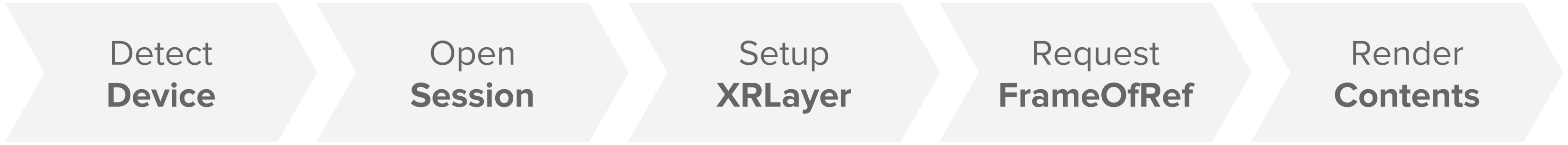


```
function onDrawFrame(time, frame) {
    ...
}
```

XRFrame’s informations

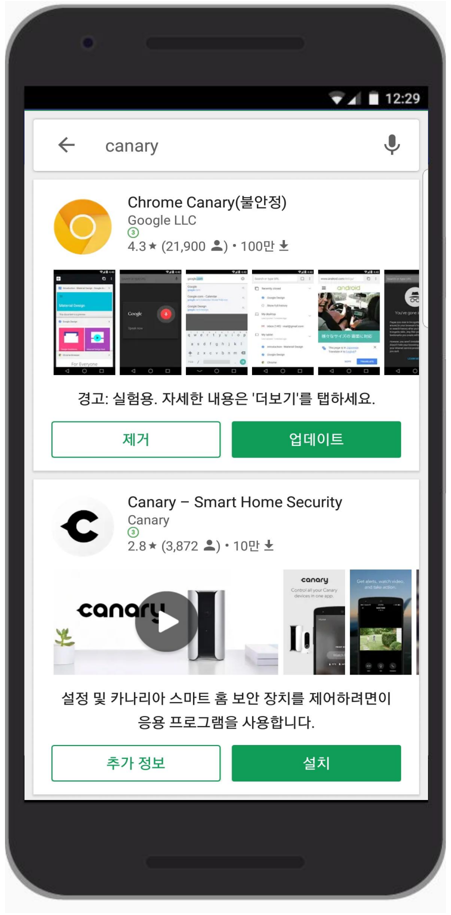
- Pose
  - In general, the position and orientation of a thing in AR/VR is called a pose.
- Views
  - Each view corresponds to a display or portion of a display used by an XR device to present imagery to the user.

Summary: How WebXR Works



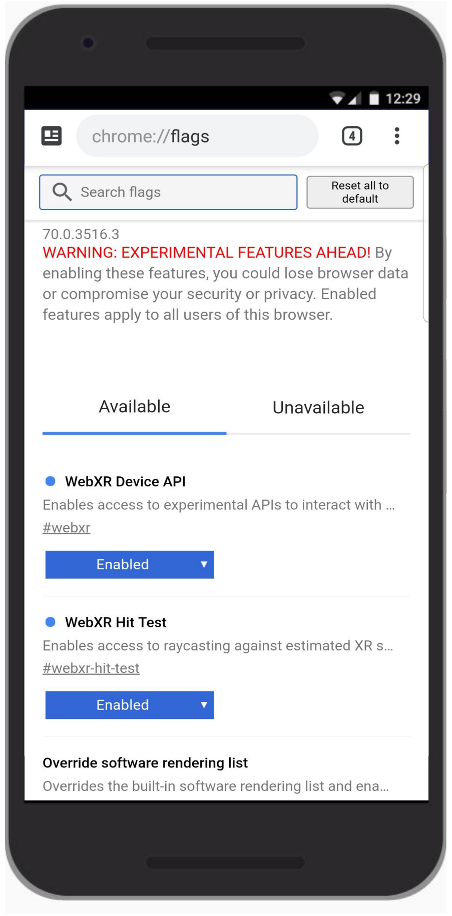
HOW TO IMPLEMENT WEB XR APP

Install Chrome Canary


12:29
canary
Chrome Canary(불안정)
Google LLC
4.3★ (21,900) • 100만
경고: 실험용. 자세한 내용은 '더보기'를 탭하세요.
제거
업데이트
Canary – Smart Home Security
Canary
2.8★ (3,872) • 10만
설정 및 카나리아 스마트 홈 보안 장치를 제어하려면이 응용 프로그램을 사용합니다.
추가 정보
설치

chrome://flags

- Enable the WebXR Device API (**#webxr**) flag
- Enable the WebXR Hit Test (**#webxr-hit-test**) flag



70.0.3516.3
WARNING: EXPERIMENTAL FEATURES AHEAD! By enabling these features, you could lose browser data or compromise your security or privacy. Enabled features apply to all users of this browser.

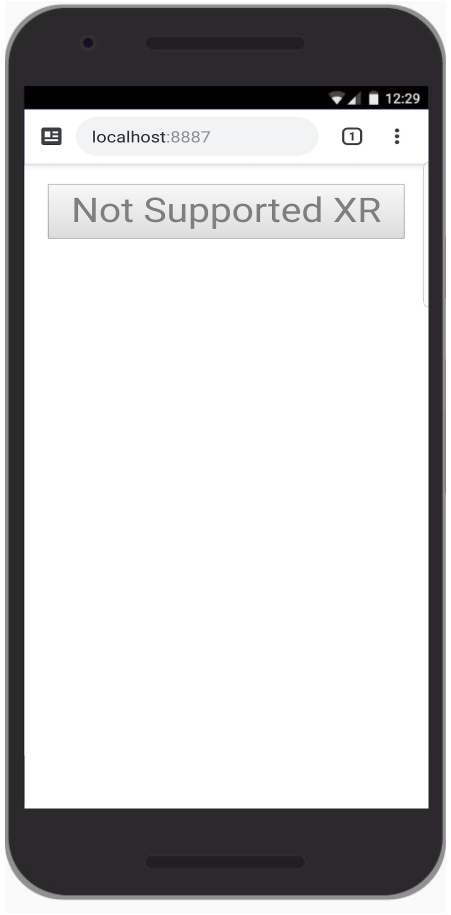
WebXR Device API
Enables access to experimental APIs to interact with ...
#webxr
Enabled

WebXR Hit Test
Enables access to raycasting against estimated XR s...
#webxr-hit-test
Enabled

Override software rendering list
Overrides the built-in software rendering list and ena...

Make Simple HTML Page

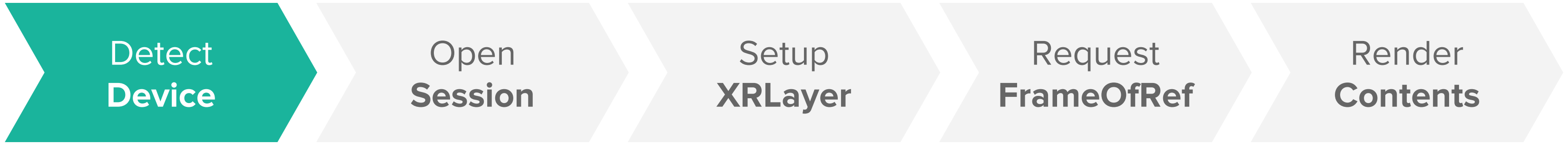
```html
<button disabled>Not Supported XR</button>
```



How WebXR works

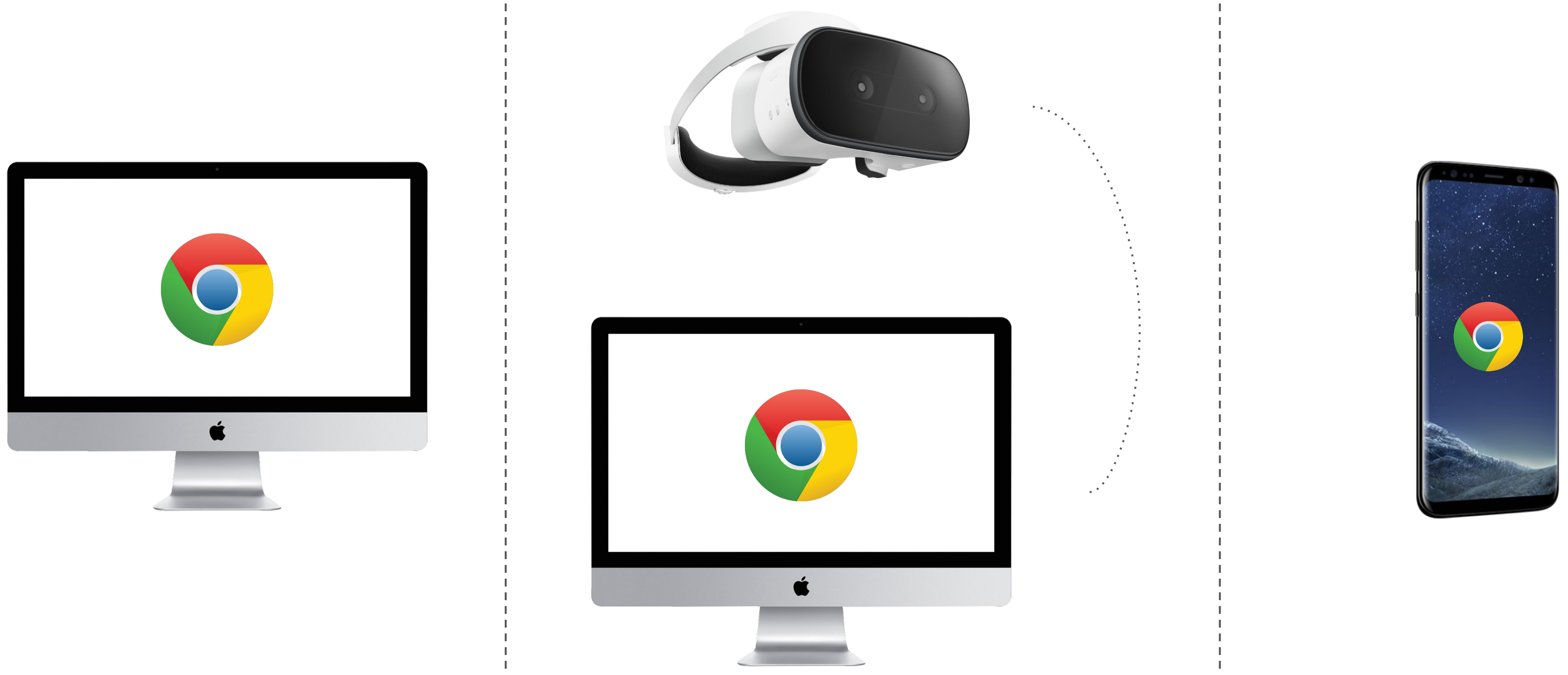
- Detect **Device**
- Open **Session**
- Setup **XRLayer**
- Request **FrameOfRef**
- Render **Contents**

STEP1: Detect Device

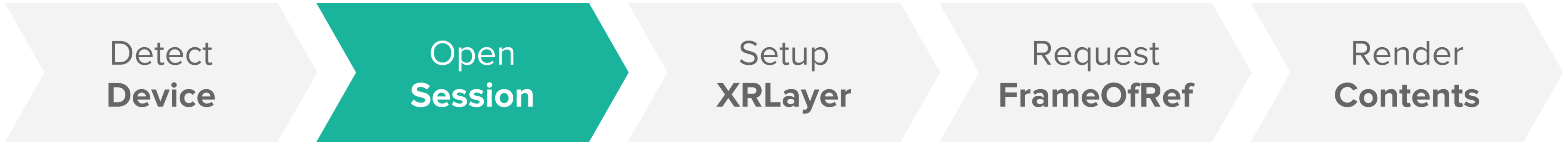


Request Device

```
// Detect default XRDevice
const device = await navigator.xr.requestDevice();
```


Not Found
Available XRDevice
Available XRDevice

STEP2: Open Session

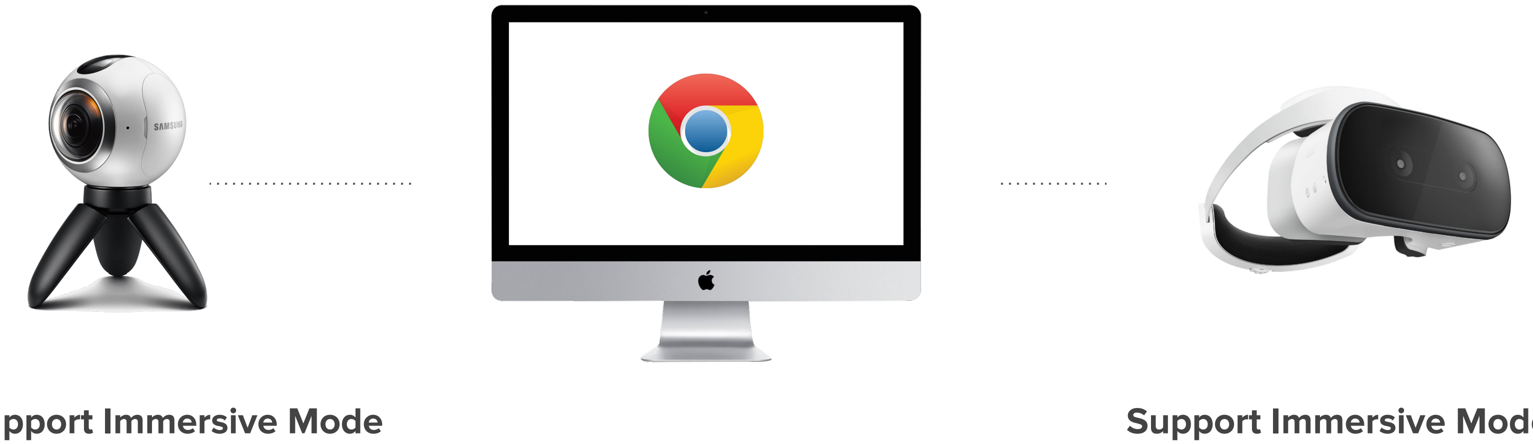


Supports Session Check

```
await device.supportsSession({ immersive: true });
```

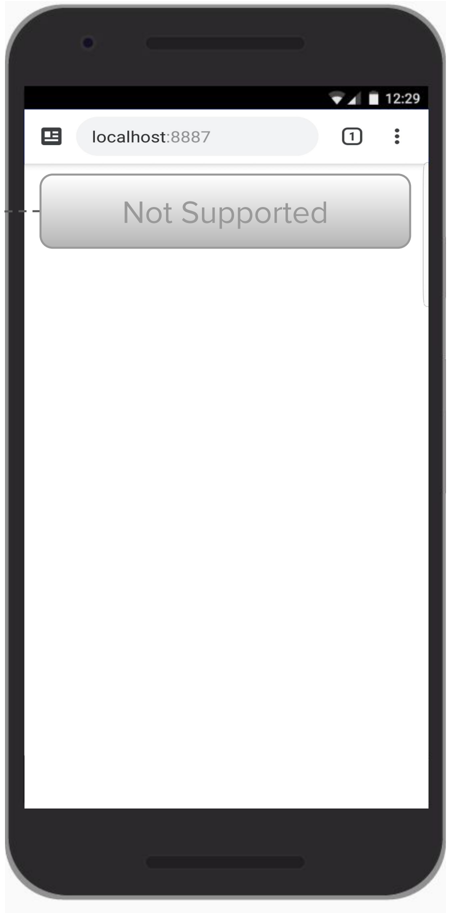
Supports Session Check

```
try {
  await device.supportsSession({ immersive: true });
} catch(err) {
  // Not support session
}
```


Not Support Immersive Mode
Support Immersive Mode

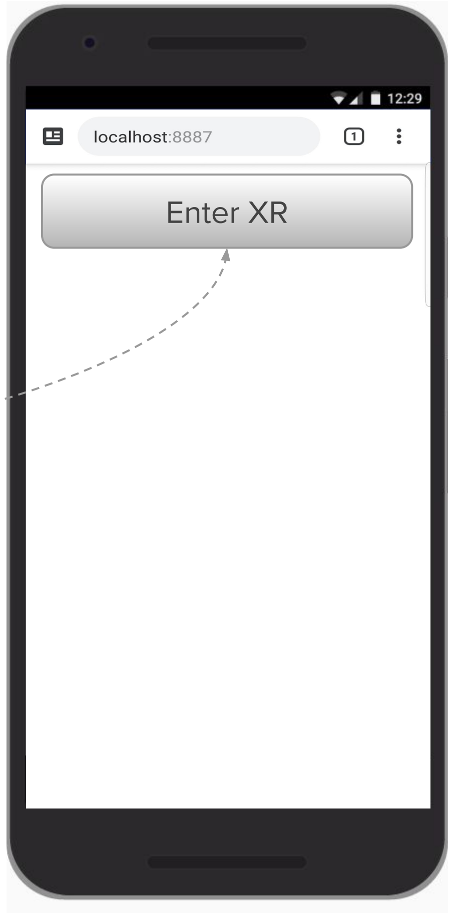
Enable “Enter XR” Button

```
const xrButton = document.querySelector(‘button’);
try {
  await device.supportsSession({ immersive: true });
} catch(error) {
  // Not support session
}
```



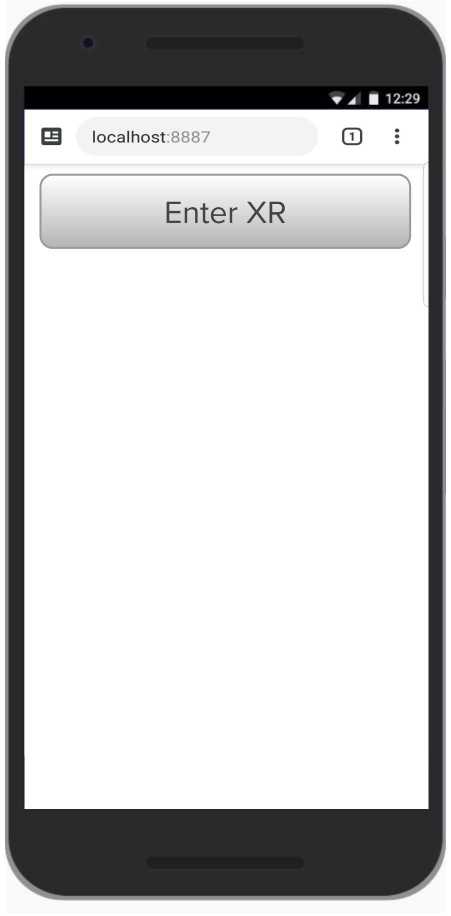
Enable “Enter XR” Button

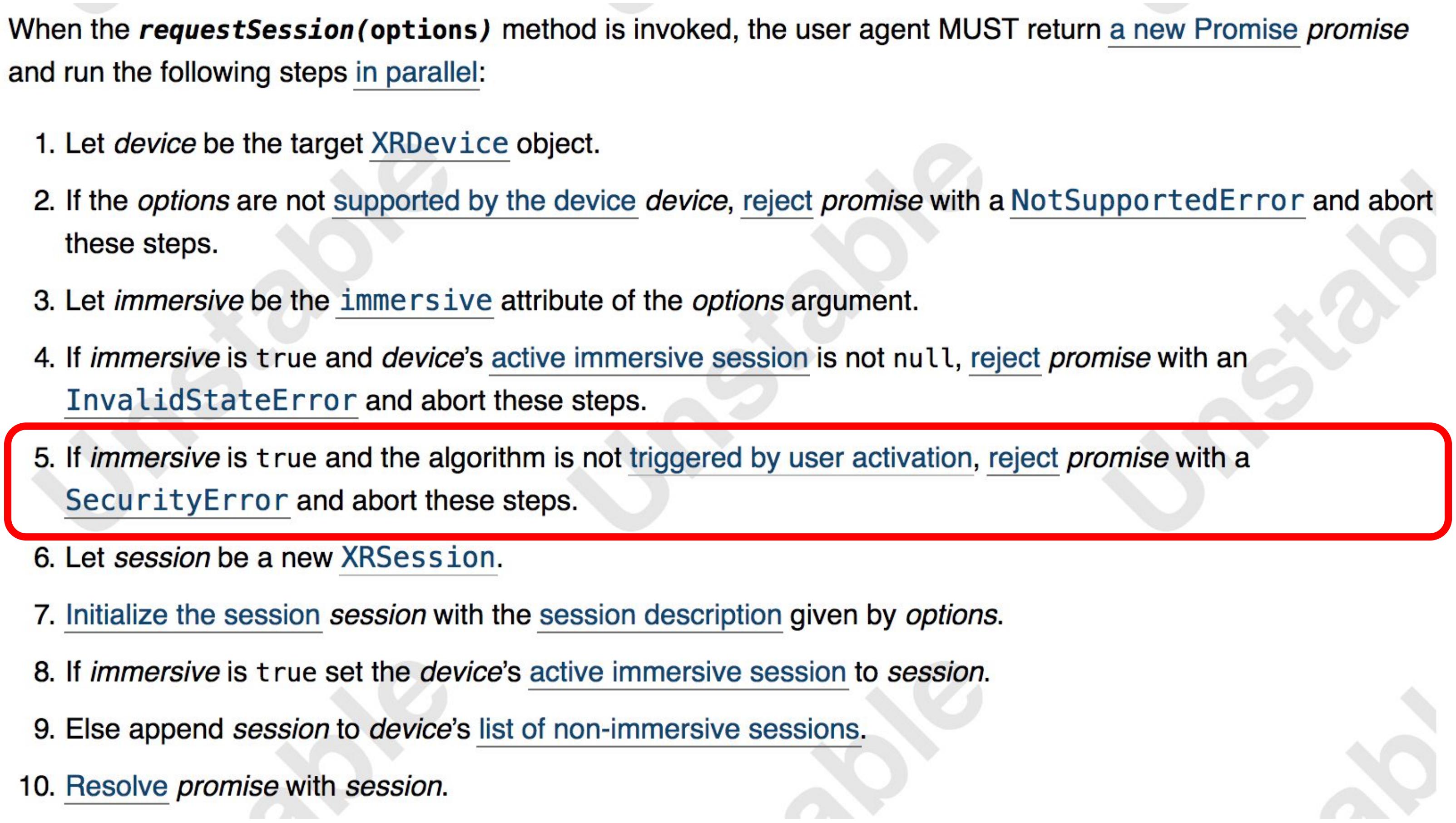
```
const xrButton = document.querySelector(‘button’);
try {
  await device.supportsSession({ immersive: true });
  xrButton.innerText = ‘Enter XR’;
  xrButton.disabled = false;
} catch(error) {
  // Not support session
}
```



User Activation

```
const xrButton = document.querySelector(‘button’);
try {
  await device.supportsSession({ immersive: true });
  xrButton.innerText = ‘Enter VR’;
  xrButton.disabled = false;
  xrButton.addEventListener('click', onEnterXR);
} catch(error) {
  // Not support session
}
```



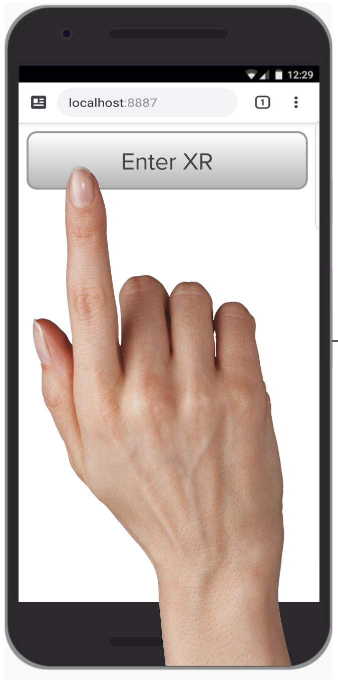


When the ***`requestSession(options)`*** method is invoked, the user agent MUST return a new Promise *promise* and run the following steps in parallel:

1. Let *device* be the target `XRDevice` object.
2. If the *options* are not supported by the device *device*, reject *promise* with a `NotSupportedError` and abort these steps.
3. Let *immersive* be the `immersive` attribute of the *options* argument.
4. If *immersive* is `true` and *device*'s active immersive session is not `null`, reject *promise* with an `InvalidStateError` and abort these steps.
5. If *immersive* is `true` and the algorithm is not triggered by user activation, reject *promise* with a `SecurityError` and abort these steps.
6. Let *session* be a new `XRSession`.
7. Initialize the session *session* with the session description given by *options*.
8. If *immersive* is `true` set the *device*'s active immersive session to *session*.
9. Else append *session* to *device*'s list of non-immersive sessions.
10. Resolve *promise* with *session*.

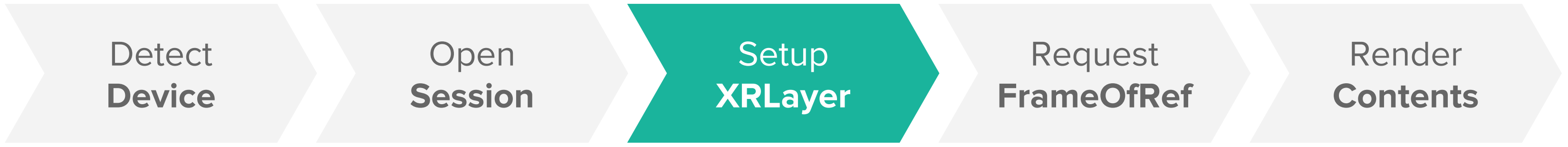
Request Session

```
function onEnterXR() {
  session = await device.requestSession({
    immersive: true
  });
  ...
}
```



```
device.requestSession({
    immersive: true
});
```

STEP3: Setup XRLayer



Setup XRLayer

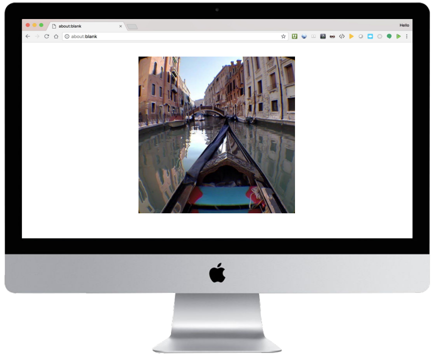
```
function setupXRLayer() {
  const canvas = document.createElement('canvas');
}
```

Setup XRLayer

```
function setupXRLayer() {
  const canvas = document.createElement('canvas');
  gl = canvas.getContext('webgl', {
    compatibleXRDevice: session.device
  });
}
```

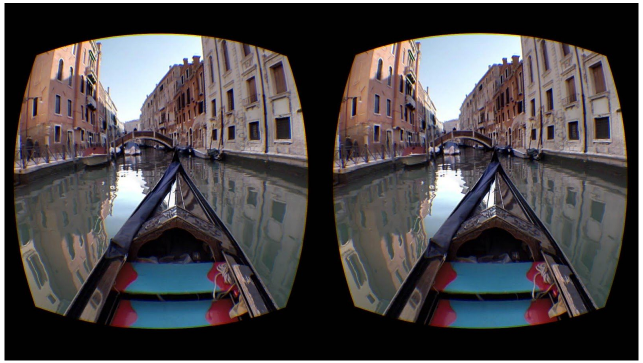
Setup XRLayer

```
function setupXRLayer() {
  const canvas = document.createElement('canvas');
  gl = canvas.getContext('webgl', {
    compatibleXRDevice: session.device
  });
  session.baseLayer = new XRWebGLLayer(session, gl);
}
```

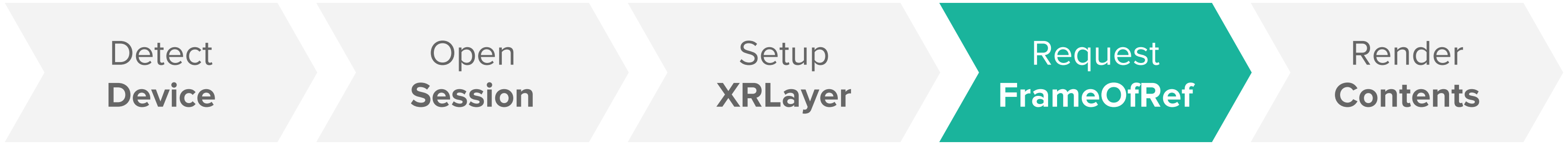


`drawScene(gl);`

XRWebGLLayer



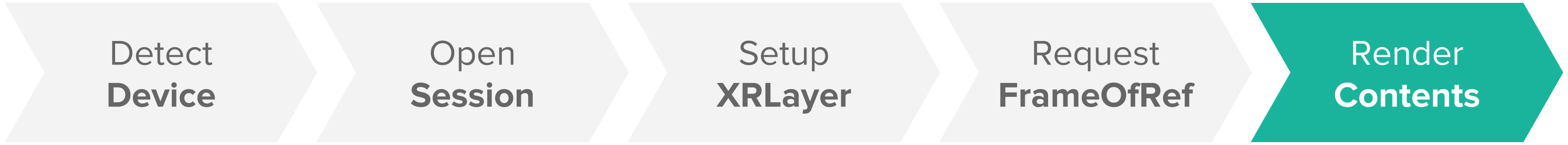
STEP4: Request FrameOfReference



Request FrameOfReference

```
frameOfRef = await session.requestFrameOfReference('stage');
```

STEP5: Render Contents



Rendering Loop

```
function onXRFrame(time, frame) {
}

session.requestAnimationFrame(onXRFrame);
```

Rendering Loop

```
function onXRFrame(time, frame) {
  session.requestAnimationFrame(onXRFrame);
}

session.requestAnimationFrame(onXRFrame);
```

Get Device Pose

```
function onXRFrame(time, frame) {
  session.requestAnimationFrame(onXRFrame);
  const pose = frame.getDevicePose(frameOfRef);
}

session.requestAnimationFrame(onXRFrame);
```

Get Device Pose

```
function onXRFrame(time, frame) {
  session.requestAnimationFrame(onXRFrame);
  const pose = frame.getDevicePose(frameOfRef);
  if (!pose)
    return;
}

session.requestAnimationFrame(onXRFrame);
```

Bind Frame Buffer

```
function onXRFrame(time, frame) {
  session.requestAnimationFrame(onXRFrame);
  const pose = frame.getDevicePose(frameOfRef);
  if (!pose)
    return;
  gl.bindFramebuffer(session.baseLayer.framebuffer);
}

session.requestAnimationFrame(onXRFrame);
```

Rendering

```
function onXRFrame(time, frame) {
  ...
  for(let view of frame.views) {
  }
}
```

Rendering

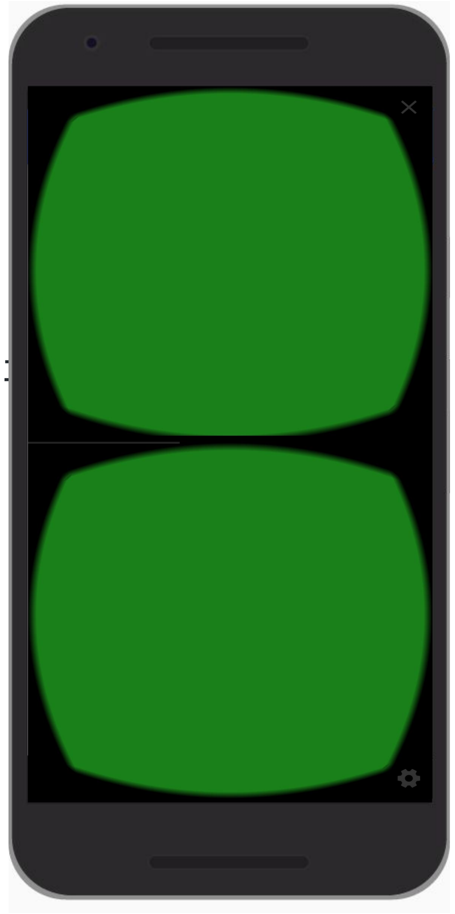
```
function onXRFrame(time, frame) {
  ...
  for(let view of frame.views) {
    const viewport = session.baseLayer.getViewport(view);
  }
}
```

Rendering

```
function onXRFrame(time, frame) {
  ...
  for(let view of frame.views) {
    const viewport = session.baseLayer.getViewport(view);
    gl.viewport(viewport.x, viewport.y,
                viewport.width, viewport.height);
  }
}
```

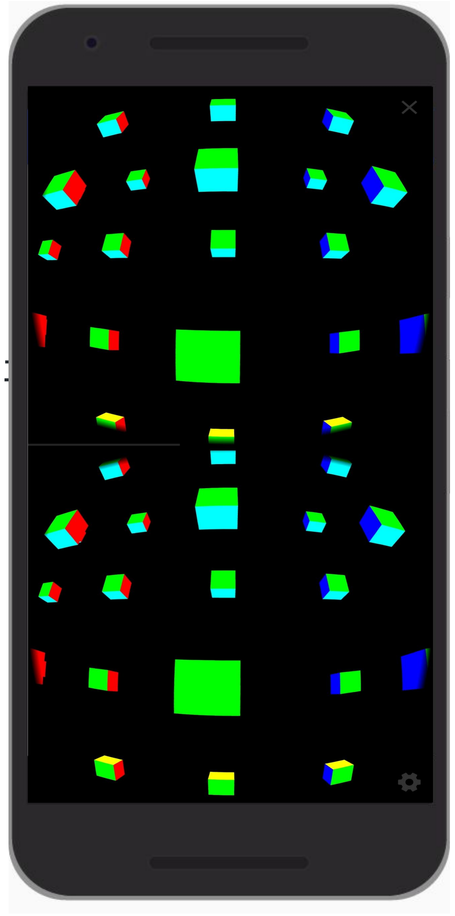
Rendering

```
function onXRFrame(time, frame) {
  ...
  for(let view of frame.views) {
    const viewport = session.baseLayer.getViewport(vi
    gl.viewport(viewport.x, viewport.y,
                viewport.width, viewport.height);
    gl.clearColor(0.1, 0.5, 0.1, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
  }
}
```



Rendering with Three.js

```
function onXRFrame(time, frame) {
  ...
  for(let view of frame.views) {
    const viewport = session.baseLayer.getViewport(vi
    gl.viewport(viewport.x, viewport.y,
                viewport.width, viewport.height);
    draw(view.projectionMatrix,
         pose.getViewMatrix(view), gl);
  }
}
```



MAGIC WINDOW

Magic Window

```
const device = await navigator.xr.requestDevice();
```

Magic Window

```
const device = await navigator.xr.requestDevice();
const canvas = document.createElement('canvas');
```

Magic Window

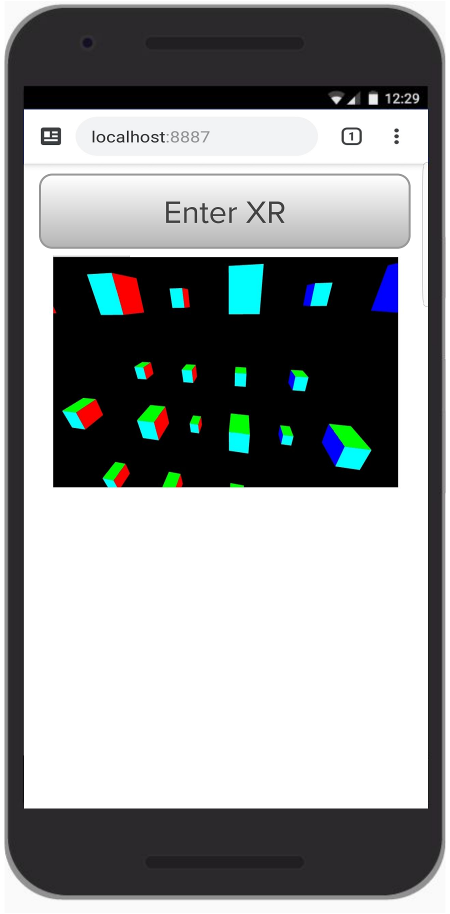
```
const device = await navigator.xr.requestDevice();
const canvas = document.createElement('canvas');
const context = canvas.getContext('xrpresent');
```

Magic Window

```
const device = await navigator.xr.requestDevice();
const canvas = document.createElement('canvas');
const context = canvas.getContext('xrpresent');
const session = await device.requestSession({
  outputContext: context
});
```

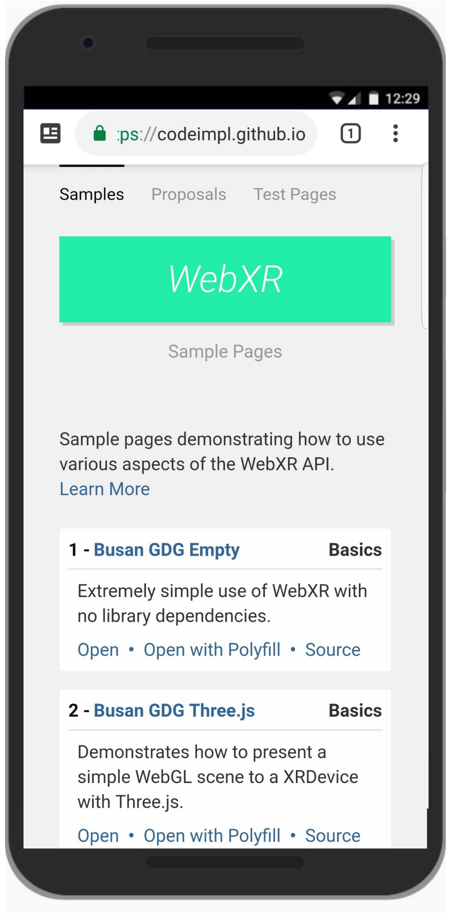
Magic Window

```
const device = await navigator.xr.requestDevice();
const canvas = document.createElement('canvas');
const context = canvas.getContext('xrpresent');
const session = await device.requestSession({
  outputContext: context
});
document.body.appendChild(canvas);
```



Demo & Source Code

```
https://codeimpl.github.io/webxr-samples/
```



WebXR

Sample Pages

Sample pages demonstrating how to use various aspects of the WebXR API.
Learn More

1 - Busan GDG Empty — Basics

Extremely simple use of WebXR with no library dependencies.

Open • Open with Polyfill • Source

2 - Busan GDG Three.js — Basics

Demonstrates how to present a simple WebGL scene to a XRDevice with Three.js.

Open • Open with Polyfill • Source

DEMO

Q & A

\o/

Thank you