This talk shares our exploration on the api, implementation, and practical use of immersive web apis.
Cross-platform team of ByteDance.

Among the various roles in the community, the (small) green dot is us!
We gain knowledge from the community, but lean towards real world businesses.

We are:
- Practitioner in Web community: w/ exploration and experimentation on new territories.
- Enabler for business teams: w/ latest Web standards and Web engine expertise

We gain ideas from the community, sometimes come up with ours, in return we tell our stories!

21.8 is the time we start working on XR (I’m sure I missed some important dates!), our solutions are available on PICO Neo 3 and PICO 4.
The big question
— empowering XR with Web

<table>
<thead>
<tr>
<th>Platform</th>
<th># of Native Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta Quest</td>
<td>600+</td>
</tr>
<tr>
<td>App Store Vision Pro</td>
<td>1000+</td>
</tr>
<tr>
<td>Google Play</td>
<td>2.43 million</td>
</tr>
<tr>
<td>App Store iOS</td>
<td>2.28 million</td>
</tr>
</tbody>
</table>

• XR ecosystem demands:
  - Richer content
  - Lower development costs
  - Easier distribution
  - Cross-platform frameworks

• Web promises:
  - Myriad apps and practice
  - Battle-tested infrastructures/tools
  - Inherently dynamic
  - Standardization

XR ecosystem is developing, and web can help!

Quest 600+ https://www.meta.com/experiences/section/1888816384764129/
iOS 2.28m https://www.businessofapps.com/data/app-stores/
Vision Pro 1000+ https://www.roadtovr.com/apple-vision-pro-1000-apps/
Web on XR currently has 2 paths:
- WebXR which renders the whole scene (exclusive)
- Progressively adding 3D-capabilities onto HTML/CSS, also progressively adding Web content onto existing 3D XR content.
Basic overview of WebXR, explaining how it renders to the eye buffers and cannot coexist with other content.

Web ecosystem has good abstractions (frameworks) above WebXR, making it easy to adopt.
Basic overview of the progressive path. Web engines does not render 3D content by itself. Instead it delegates to a lower level runtime, which composite everything in the 3D space.

Example: Magic Leap 1 and Safari on Vision Pro has declarative apis for Web to render in 3D, each on top of there system level engine (Lumin/RealityKit).
Our solution.

Standalone: basically for WebXR. Packages an Web app into an android app.
Embedded: integrate with 3D Engine or another runtime.
Applying WebXR
WebXR needs to start from 2D page. When packaged with our solution (or, on PICO, manifested as an “immersive PWA”), can request immersive session with code.

Open another WebXR page also start from 2D. WebXR Navigation implemented.
Rendering 2D content in WebXR is not good developer experience. Instead we extend WebXR DOM Overlay to 3D.
We use latest Web capabilities to address developer needs. We implement WebXR Layers and WebXR on Workers for perf.
The status quo and the big transition

- In the long run, Web shines in the general use cases ...
- But it can also shine today! In use cases that:
  - Have tight schedule
  - Change frequently
  - Special events, A/B tests …

The progressive path can shine in:
- multitasking use cases
- dynamic part in immersive apps

Screenshot shows a 2D event page in a 3D scene, which can be improved with more 3D content.
Turning Hybrid
We implemented what is called Detached CSS (originally by Magic Leap).

Video shows a page comprising detached planes.
Model Element

- Places a 3D model into space
- Can combine with detached CSS
- Basic interaction support
- Supports GLTF/OBJ/Engine-Specific formats
- Draft spec
- Prior art (Magic Leap, Google)
- Recent interest (Apple)

Also implemented model element.

Video shows 3 models in the scene: rocket, girl, and skybox.
Each scene can contain multiple Web pages, with any one of them running WebXR immersive session.

Description data of planes and models are sent to the lower level engine, and with WebXR frame buffer they are composited.

Screenshot shows usage of our package in Unity: a prefab (WebView) that can be configured in 'Unity style'.
We modified the compositing step so that each plane or model is an individual render target, keeping its own transform in 3D.

Main render loop is made up of a couple of callbacks: (sending frame information to web engine) -> (receiving render targets from web engine) -> (render those targets along with other stuff of the host app)

XR rendering needs to run in high frame rate and must not block on Web rendering. So Web updates are async: an update can last for more than one frame, and takes effect by swapping the active frame and pending frame.
**Entering new territory**

- Level of Detail and Anisotropic Filtering (performance & sharpness)
- Coordinate conversion

These are **same-sized in CSS!**

```javascript
// Convert page coords to world coords.
DOMMatrix pageToWorld();

// Convert world coords to page coords.
DOMMatrix worldToPage();

// Decompose a matrix to use in CSS Transform Functions.
TRS decompose(DOMMatrix m);
```

LOD and more 3D techniques are used to improve render perf and quality.

Util functions are provided to convert coords.
Scrolling is hard in 3D. Design UX carefully.
Sync planes with model animation: make slots on model and associate planes with slots at run time with declarative syntax.
We believe the progressive path has huge potentials. And it is friendly in terms of accessibility, discoverability, and interoperability with existing Web.
Thank you!