

Kobe, Japan & online
10–14 November 2025

WebSpatial API for Spatialized HTML/CSS and Spatialized PWAs on spatial and multimodal AI devices

**Dexter Yang, Ruoya Sheng, Siyaman Sadagopan
(PICO OS, ByteDance)**

Today's Agenda

1. The Problems We're Facing
(and What We Currently Have, Why That's Not Enough)
2. What Solutions or APIs Are Required to Solve These Problems
3. Discussion and To-Dos
4. Real-World Practices

1. The Problems We're Facing (and What We Currently Have, Why That's Not Enough)

3

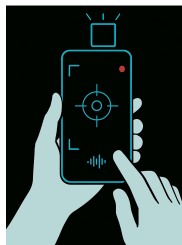
Big Problem 1 of 3: AI and Next-Gen Personal Computing Are Reshaping Apps, While Web Is Falling Behind

What's Happening?

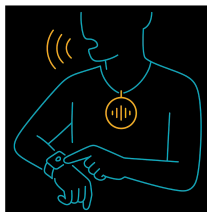
4

What's Happening?

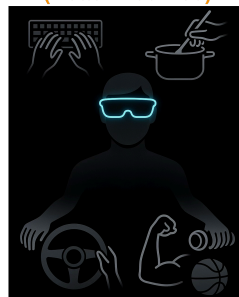
Handheld
Devices



Wearable
Devices



Hands-free Wearable
(Head-mounted)



5

Because AI is in such high demand, the next-gen personal computing devices
and OS need much better multimodal interaction

AI Glasses

What you can see/hear,
AI should also see/hear.

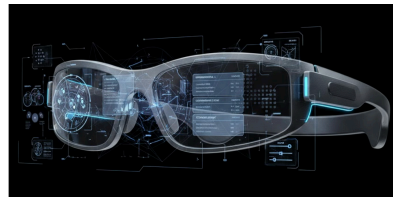


6

Hands-free

AR Glasses

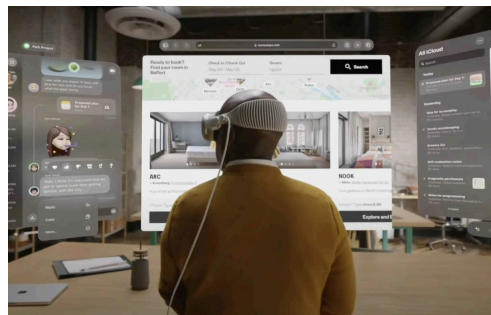
- Natural language fits coarse-grained interaction tasks.
- Fine-grained interactions and intuitive feedback still require a GUI.
- GUI needs to integrate both with the context of AI interactions and with the surrounding physical space.
- GUI is moving beyond screens.



7

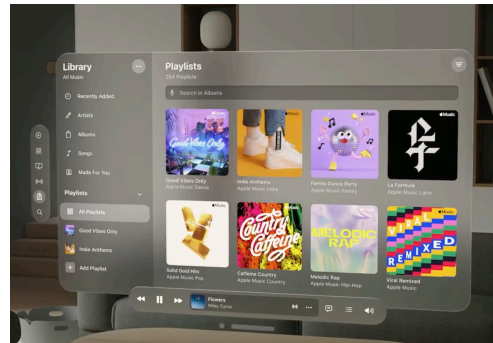
MR Goggles (MR Headsets)

- XR headsets are evolving to align with AR glasses, with more full-featured third-party apps and better spatial integration.
- AR glasses replace phones, MR goggles replace laptops/tablets.

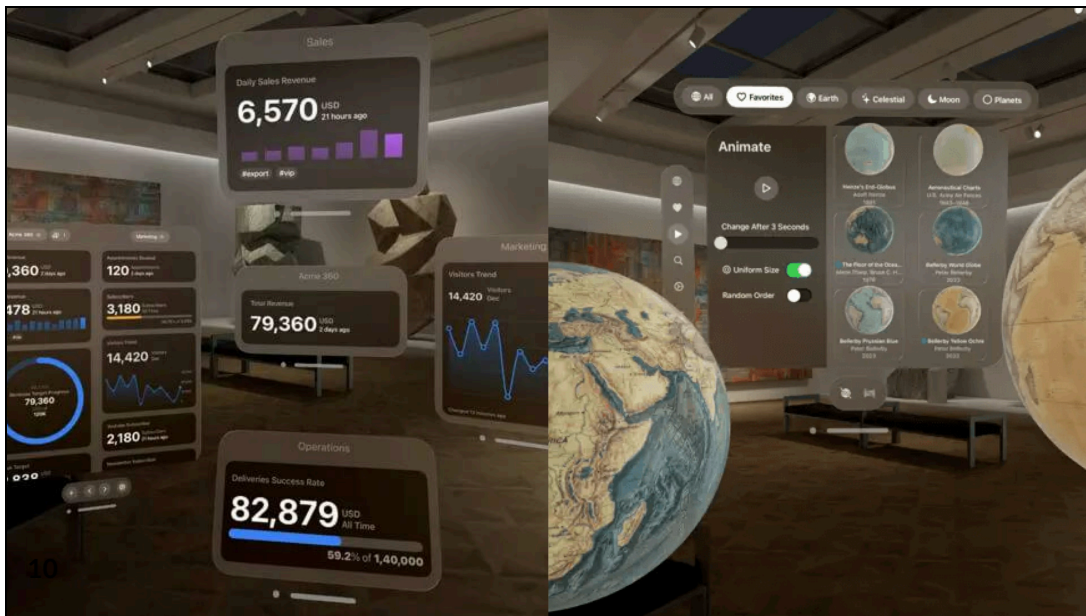


8

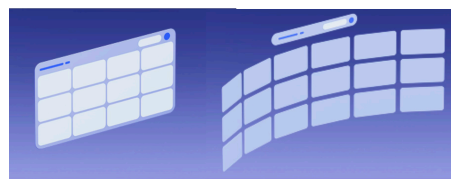
- The 2D app ecosystem needs to be preserved and continued.
- 2D GUIs still fit best in most cases, but parts of their content, details, and how multiple 2D GUIs work together need to break free from the limits of flat screens.



9



Split the Web Page, Free the UI



Elevate HTML Elements, Unlock Depth



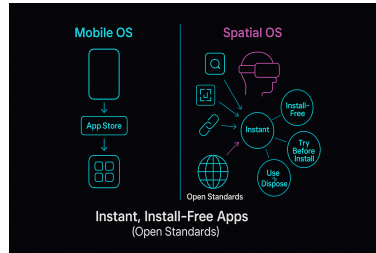
Multiple Scene Containers,
Native Power



Add True 3D Content, Blend Dimensions

11

Next-gen operating systems have a far greater need than mobile OSes for "install-free apps" based on open standards.



12

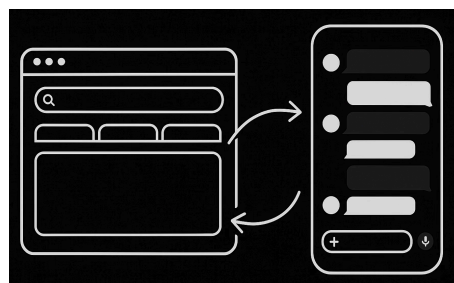
* "Install-free apps" are large-scale and hard to catalog; they launch via a link, run on demand, are disposable by default, and can be upgraded to installed apps when needed

Why

- Client-side AI agents are increasingly choosing "tools" on their own. The kinds of apps for "Tool Use" tend to be vast, unknown, and infrequently used.
 - That means they are unsuitable to be pre-installed, installed on the spot, or retained on the device afterward.
- App discovery and launching from spatial environments work the same way
 - Just like how people in China and Japan scan QR codes to join events or place orders.

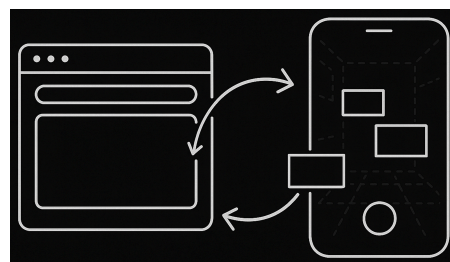
13

The sole "super app" of the desktop era - the browser - is making a comeback, but in new forms



ChatGPT app

Chat boxes are replacing address bars.
Message feeds are replacing tabs.



TikTok/Snapchat camera

XR seethrough views are replacing the address bars.
Window containers with spatial layout are replacing tabs.

14

"Install-free apps" are not inherently bound to the Open Web

We already saw this split happen in China's mobile internet market, where non-standard "mini-app" ecosystems inside super apps like WeChat haven't just taken over most native app needs, but have also pretty much wiped out the Open Web in China.



15

What's the Problem

Apple and Google are extending their native 2D GUI frameworks and platform-specific app ecosystems to support experiences that go beyond screens, blend with spatial environments, and still meet mainstream app needs, whereas the Open Web today cannot satisfy these demands concurrently.

visionOS

- Compatible iPhone and iPad apps -> visionOS apps
- SwiftUI + RealityKit + ARKit

Android XR

- XR compatible large screen app -> XR differentiated app
- Jetpack Compose + SceneCore + ARCore

16

The Web might once again fall way behind native apps

The web fell behind during the paradigm shift from desktop to mobile.

- "Gap Analysis between Web and Native on Mobile"
<https://w3c-webmob.github.io/gap-analysis/>
- "Closing the gap with native"
https://www.w3.org/wiki/Closing_the_gap_with_native

17

What We Currently Have and Why That's Not Enough

If the status quo persists, as AI/AR glasses and visionOS/Android XR devices take off, web devs will be forced to switch stacks - moving to native 2D GUI stacks or to hybrid ecosystems like React Native ([React Native visionOS](#)) / Mini-apps that diverge from the mainstream web.

Native

Development outcomes accumulate in closed, platform-exclusive walled gardens.

Native / React Native

Loss of the web's core advantages, like URLs, no-install, on-demand access.

Mini-apps

It neither inherits from nor integrates with the existing web ecosystem, it starts anew.

18

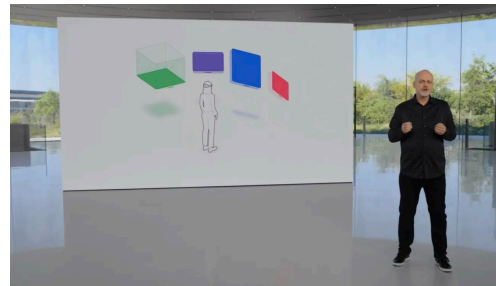
Big Problem 2 of 3: Mainstream Web Stack Lacks New UI Capabilities for Spatial Apps

What's Happening?

19

Paradigm shift in XR OS: From Compositor-based Architecture to Unified Rendering Architecture

Through the visionOS platform, Apple pioneered a Unified Rendering architecture and a Shared Space model for multi-app coexistence, setting a new standard in the industry.



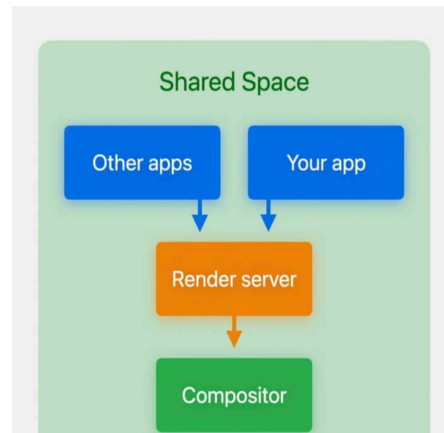
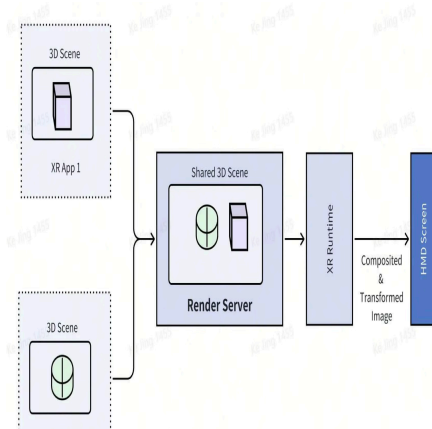
20

Shared Space



21

Unified Rendering



22

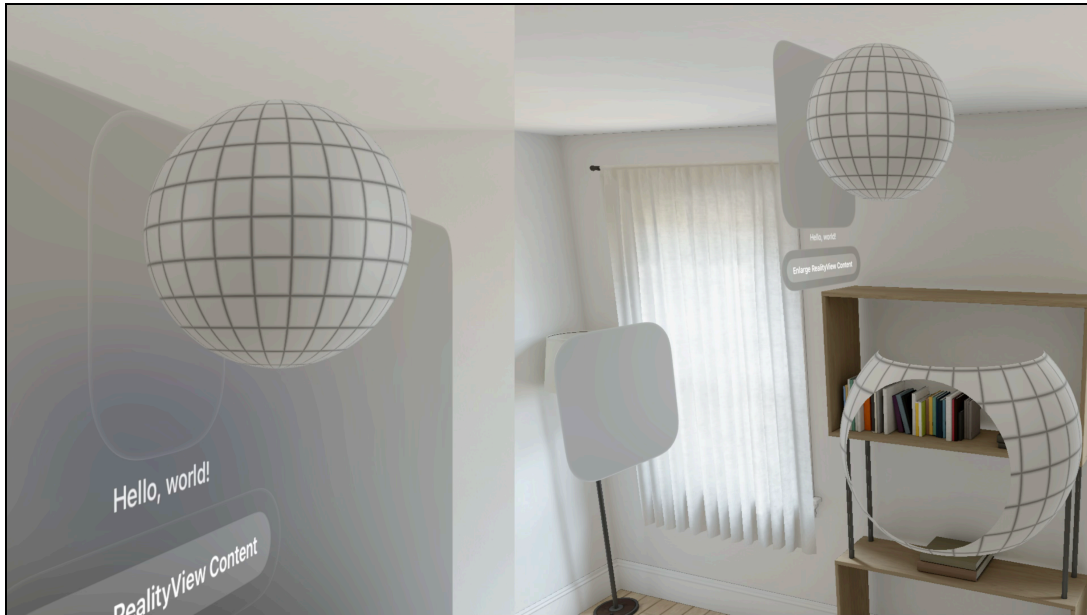
- To meet mainstream needs and general-purpose use cases, the OS should default to multitasking - multiple apps coexisting
 - Each app handles just a portion of the display/space instead of taking over the whole thing.
 - Allow quick switching and combined use.
- To integrate with and fully leverage spatial environments, coexisting apps need to go beyond 2D windows and integrate into one shared space
 - Rather than existing solely as overlays/HUDs
 - 3D/Spatialized content from different apps shares spatial positional relationships, is affected by the same spatial environment (e.g., lighting, frosted-glass background), and enables consistent spatial interaction.
- These coexisting apps aren't 2D apps or immersive apps anymore; they're called "spatial apps".
- They can mix 2D content and content with spatial relationships or 3D volume, and they can still all share the same space.

23

With visionOS, Apple has established industry design patterns for spatially 2D+3D hybrid GUIs:

- The OS is responsible for implementing and managing spatial scene containers - Windows for GUI, Volumetric Windows for simulating objects, or fully immersive spaces.
- Each running app must place its content inside these containers to integrate with the same space.
- Apps can't control these containers directly; they can only provide their desired configurations to the OS during container initialization (like type, default size, resize constraints)
- Windows and Volumes are like bounding boxes, and all 2D/3D content sticks to the box's back face and is treated as 2D frames managed by a 2D layout system, but can move or morph along the Z-axis when needed.
- Some of these 2D frames are 3D content containers, rendering volumetric content in the space in front of and within their bounds.

24



- Spatial apps cannot render themselves in isolation. The OS should render them uniformly
 - which means it needs to understand what the apps contain, instead of compositing pre-rendered, information-losing frames.
- Spatial apps cannot implement arbitrary low-level interactions independently (e.g., rendering the user's hands). The OS should provide unified visual cues (such as hover effects) and hit testing, while apps just receive and handle gesture events.
 - **With visionOS, Apple has established industry design patterns for natural interaction:**
 - Eye-hand-based indirect interaction (selection by gaze and confirmation via finger gestures)
 - Touch-based direct interaction
 - Both support basic spatial gestures like drag, rotate, and zoom.

26

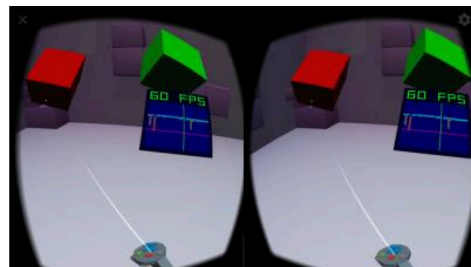


What We Currently Have for Spatial UI and Unified Rendering

- WebXR
- HTML/CSS/DOM
- Spatial Browsing
- PWA

28

The current Immersive Web Working Group focuses on the WebXR API standard.



29

Why We Should Do More

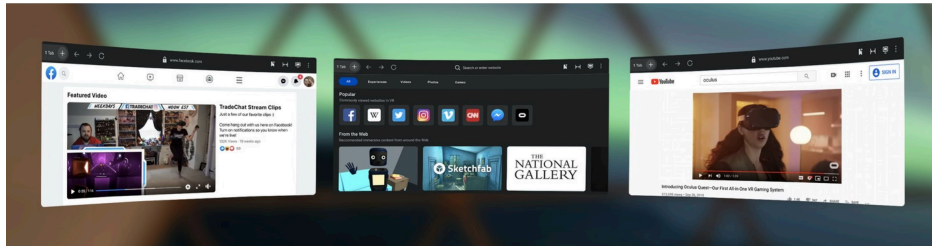
Like OpenXR, WebXR API takes over the XR device's full stereo view and the entire space, renders on its own using low-level 3D graphics APIs (WebGL/WebGPU), submits only final frames to the OS compositor, and requires building natural interaction from scratch.

- WebXR sessions can't coexist with their host page window or other app windows.
- WebXR sessions can't render only part of a Shared Space, and multiple WebXR sessions can't blend into the same Shared Space.
- 3D graphics APIs like WebGL/WebGPU, which work on the final frame pixel by pixel instead of describing spatial content, may fundamentally conflict with the Unified Rendering architecture.

30

What We Currently Have for Spatial UI and Unified Rendering

HTML/CSS/DOM are screen-based by now, even without a screen.



31

Why Screen-based HTML/CSS Isn't Enough

Current layout systems in HTML/CSS/DOM only support the X and Y axes.

- z-index is about stacking order, not a Z-axis API.
- CSS Transform API can have a Z axis, but only affects appearance after 2D projection.

32

Why Screen-based HTML/CSS Isn't Enough

All HTML elements are just flat 2D panels with no volume.

- Web 3D content ultimately projects onto a 2D canvas plane.

33

Why Screen-based HTML/CSS Isn't Enough

Only fixed, solid colors are available, and CSS styles can only be manually authored based on static device states (media queries)

- Element backgrounds and text colors cannot dynamically track the surrounding environment, making them unsuitable for the complexity of the real world (coexisting hues and light/dark conditions) and for next-gen platforms where app backdrops change continuously.

34

Why Screen-based HTML/CSS Isn't Enough

Existing web window-related APIs are insufficient for spatial scene containers.

- Opening a new page window via `<a>` allows no initialization configuration.
- When a new page window is opened via `window.open`, the configuration options control the exact window size and resize permission, which is not the same as the initialization semantics of spatial scene containers.
- If it runs as an installed PWA (a standalone app with its own window, outside the browser), you also can't set any initial options for the first page window.
- Web page windows don't have a type concept, so they can't support new spatial scene container types like Volume.
- Even standalone PWA windows have a fixed, solid background and visible border, preventing UI elements from appearing spatially separated.

35

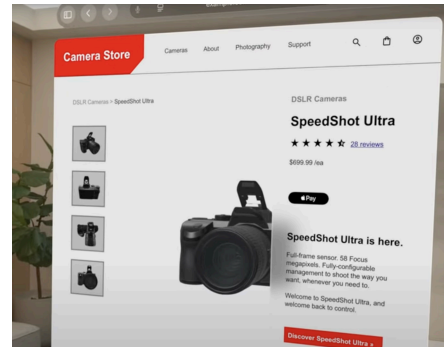
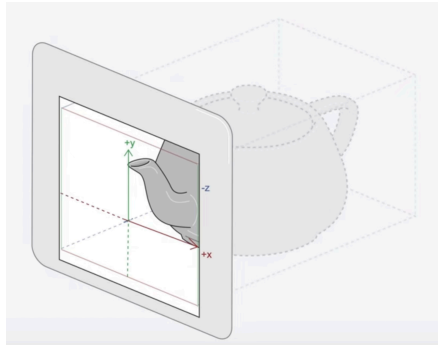
Why Screen-based HTML/CSS Isn't Enough

Supports only low-level JS interaction events based on 2D positions, without natural gestures or spatial position tracking.

- Current JS interaction events (e.g., Pointer Events) are very low-level, and even basic gestures have to be implemented in JS.
- It's hard to build a great natural interaction experience just from low-level events.
- Spatial OSes can't freely expose sensitive underlying interaction data - like eye tracking - to apps because of privacy concerns.
- Spatial OSes need to ensure consistent interaction in the Shared Space by uniformly implementing core natural interaction gestures.

36

The spatial web features added to Safari on visionOS



37

The spatial web features added to Safari on visionOS

HTML

```
<picture>
  <source srcset="images/spatial/bikerack1.heic" type="image/heic">
  
</picture>
```

Javascript

```
fullscreenTarget.addEventListener('click', event=>{
  try {
    fullscreenTarget.requestFullscreen();
  } catch(error) {
    //handle error gracefully
  }
});
```



38

Why We Should Do More

Model element

- The `<model>` element can only render volumetric 3D content inside a "hole" in the page plane. It can't appear in the space in front of the page like a native SwiftUI app's Model3D view.
- The 3D content in the `<model>` element can only come from pre-made 3D model files; you can't program it dynamically (no mainstream Web 3D engine features)

39

Why We Should Do More

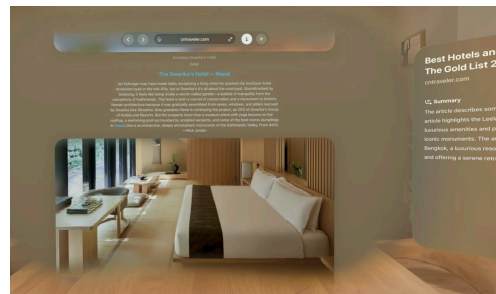
Immersive Media

- Like a WebXR session, you have to call the Fullscreen API to switch into a special mode to view spatial photos and videos, instead of viewing that spatial content right in the webpage window.

40

Spatial Browsing

- The Spatial Browsing capability introduced in Safari for visionOS 26 doesn't introduce or rely on new Web APIs, so it's merely an app-level, reader-mode-like feature available only on a limited small set of qualifying, article-centric pages.



41

Why We Should Do More

Spatial Browsing-like auto-conversion of 2D pages into spatial UIs

- Limitation: Without new Web APIs for expressing spatial intent, we're stuck recognizing a few known patterns in 2D pages, and only in matched cases can the UI be spatialized automatically.
 - Even with more powerful generative AI, it still wouldn't be a general-purpose solution.

42

Why We Should Do More

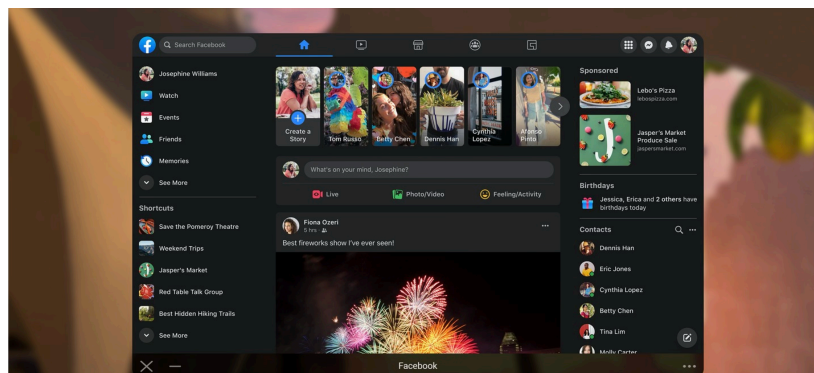
Spatial features in traditional browser UIs, like Spatial Browsing:

- Limitation: Whether the spatial UI is auto-generated or precisely built by developers, the end result conflicts with the existing browser app's Window/Tab UI and can't coexist.
 - For example, the address bar and tab bar are tightly coupled to the browser window's frame and solid-color background.
 - If we heavily rework the browser UI to suit a borderless, transparent-background spatial web UI, it could make browsing traditional 2D pages less efficient.
 - Even Apple needs users to trigger a special dedicated mode for Spatial Browsing in current Safari, just like with WebXR.

43

A Current Technology Bypasses Traditional Browser UI Limits

PWA



44

Why We Should Do More

PWA

- Once installed, PWAs can run in standalone windows without browser UI such as the address/tab bars, so it's a natural fit for spatial web UIs.
- But uninstalled PWAs can only run as browser tabs, still constrained by the browser UI.
- If a web app with a spatial UI has to be installed first to enable the spatial experience, it's basically no different from a native app.
 - Still, there are losses of the web's core advantages, like URLs, no-install, on-demand access.
 - Doesn't meet AI/MR OS's needs.

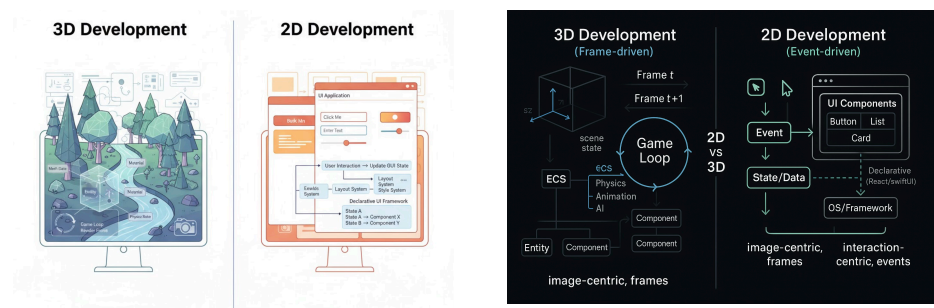
45

Big Problem 3 of 3: Web 3D is Still Hard and Lacks Developers for General Computing Needs

What's Happening?

46

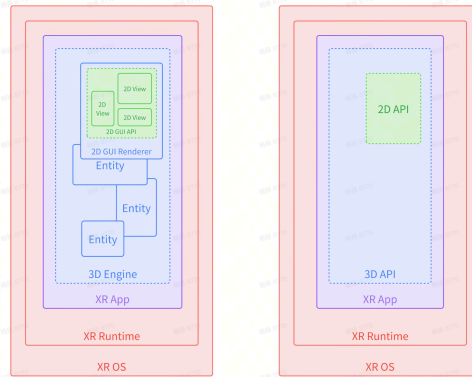
Paradigm shift in XR Development: From 3D containing 2D to 2D containing 3D



47

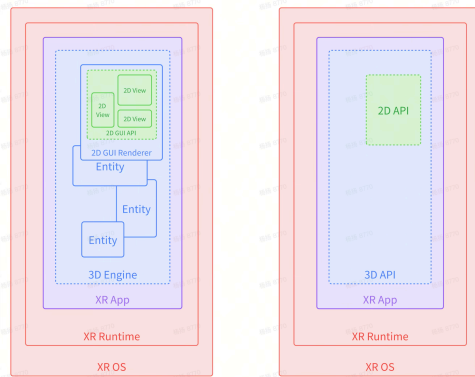
Traditional 3D development: "3D containing 2D"

- Everything defaults to 3D, and the app is built with a 3D engine.
- Some parts can still be plain 2D (like a 2D GUI), but those 2D pieces can't exist or be developed separately. You can't just plug in mainstream 2D GUI frameworks or component libraries. Instead, they need to be folded into the 3D development approach, providing dedicated APIs on top of the 3D engine that imitate typical 2D GUI development.



48

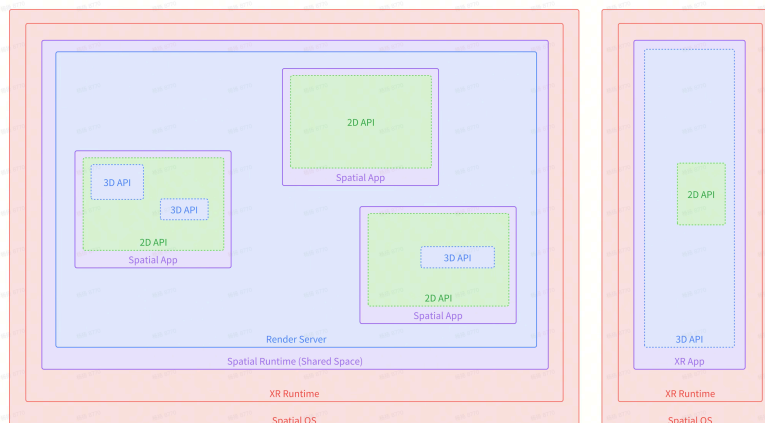
- Unity UGUI / CSS3D in Three.js / Dear ImGui



49

- Most regular web developers find it hard to take part in developing this kind of app.
- Most apps only need 3D for a few parts. Making the whole app with a 3D engine and 3D mindset just for those bits - and relying on the small pool of 3D developers - is inefficient.
- As multimodal AI devices drive apps toward mixed 2D+3D spatial forms, the current number of 3D developers and the efficiency of 3D development can't scale to match.

Spatial Development: "2D containing 3D"



50

Spatial Development: "2D containing 3D"

- The outermost Shared Space is built with a 3D engine, but it's implemented and managed by the OS.
- All spatial app content lives inside spatial scene containers within the Shared Space. Inside these containers, everything follows a 2D worldview - the content is composed of UI components based on a 2D layout system, not 3D graphics or entities rendered in a 3D coordinate space.
 - Everything defaults to 2D views and is developed using a 2D GUI framework.
- All UI components can be laid out and transformed not just along the X and Y axes but also along the Z axis.
 - With a 2D GUI development mindset alone, developers can now create many content/effects that were once possible only with 3D engines.

51

- Some UI components act as 3D content containers. The containers themselves are still used as 2D views, but inside, they follow a 3D worldview, rendering 3D entities and assets in a bounded 3D coordinate space.
 - 3D content containers bridge the outer 2D world and inner 3D worlds, allowing 2D to include 3D.
 - Since this paradigm mainly targets 2D GUI developers, it's best to blend 3D code into the 2D GUI system as much as possible - to keep the 2D mindset, lower the learning curve, and give developers a smoother experience.
 - 3D code no longer runs on its own main loop or controls every frame. Instead, updates to 3D content are triggered by the outer 2D GUI system when appropriate.
 - This kind of 3D code can't rely on low-level graphics APIs or arbitrary 3D engines, because under a Unified Rendering architecture, all content, including content in 3D containers, must be OS-understood and rendered by the OS's single rendering server.

52

- This new development paradigm enables the broad community of 2D GUI developers to join in building these kinds of apps, using familiar tools and mindset, and handling most localized 3D needs on their own without mastering full 3D engine stacks.

With visionOS, Apple was the first in the industry to define this new development paradigm

53

```

import SwiftUI
import RealityKit

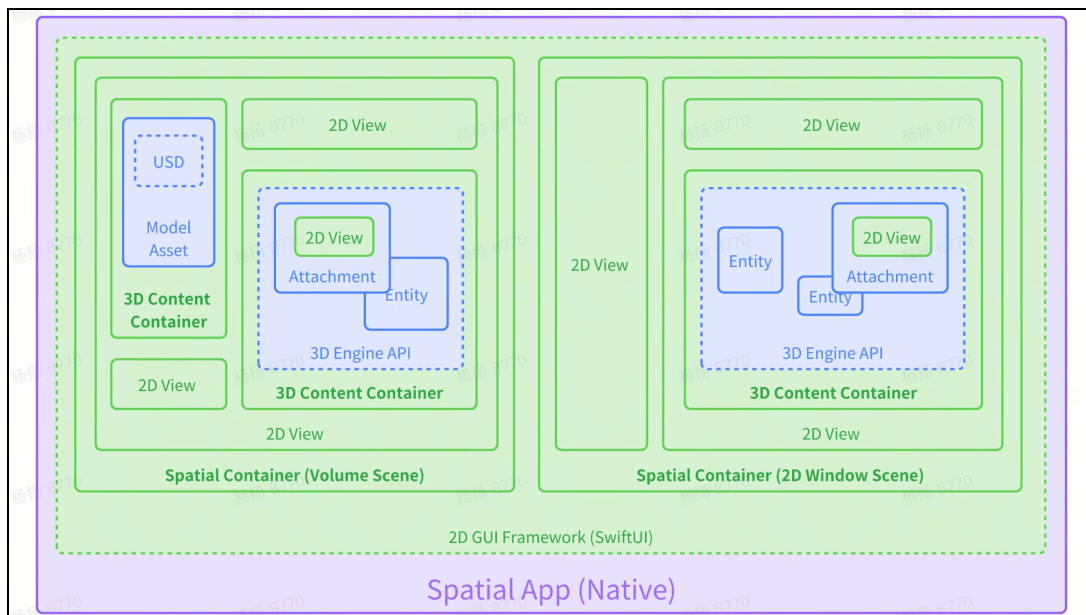
struct ContentView: View {
    @State var enlarge = false

    var body: some View {
        VStack {
            // Add the initial RealityKit content
            if let scene = try? AssetEntity(named: "Scene", in: realityKitContentBundle) {
                content.add(scene)
            }
            // Update the RealityKit content when SwiftUI state changes
            if let scene = content.entities.first {
                let uniformScale: Float = enlarge ? 1.4 : 1.0
                scene.transformFunction = UniformScale(uniformScale, uniformScale)
            }
            let physicalBounds = proxy.frame(in: .local)
            print("Size (update)", proxy.size)
        }
        .gesture(TapGesture().targetedToAnyEntity().onEnded { _ in
            enlarge.toggle()
        })
        .glassBackgroundEffect() // 2D API
        .frame(width: 200, height: 100) // Layout API
    }
}

```

- building spatial apps mainly with a 2D GUI framework (SwiftUI),
- and providing 3D content containers like Model3D View and RealityView.
 - **Model3D View:** uses USD assets for its content
 - **RealityView:** use RealityKit API (3D engine API) to build dynamic 3D content
 - RealityKit code can only run through two hooks - init and update - both of which are controlled by SwiftUI's declarative framework mechanism.
 - The RealityKit APIs available in RealityView are high-level ECS-based APIs. Even for custom material needs, you have to use high-level APIs like MaterialX, not hand-written shaders.

54



What We Currently Have for 3D Development

- `<canvas>` element
- Web 3D Engines (Three.js, R3F, AFrame, ...)
- WebXR

56

Why Canvas Isn't Enough

The existing `<canvas>` element in HTML isn't really part of this new "2D including 3D" paradigm.

- The `<canvas>` element's content is rendered independently using low-level 3D graphics APIs or an arbitrary Web 3D engine, so the OS can't render it in a unified way.
 - When `<canvas>` content is shown concurrently with other encompassing 2D HTML elements, the 3D content in a `<canvas>` is always projected onto a flat surface when rendered, so it can't display real volumetric 3D content in actual space when it is "contained within" a 2D context.
- Developing content for the `<canvas>` element isn't mainly for 2D GUI developers but rather targets developers who are familiar with 3D development mindset, 3D graphics APIs, and 3D engines.

57

Why Web 3D Engines Aren't Enough

Web 3D Engines

- Existing Web 3D engines are built on the `<canvas>` element and low-level 3D graphics APIs.
 - Inside the `<canvas>`, existing Web 3D engines follow a traditional "3D containing 2D" paradigm.
- Like Unity, most Web 3D engines can't directly support Unified Rendering and are also hard for regular web developers.
 - AFrame and R3F are much better since they both try to keep the declarative UI component style familiar from 2D development

58

Why WebXR Isn't Enough

WebXR

- WebXR follows a traditional "3D containing 2D" paradigm.
- In WebXR, 3D content is built on the `<canvas>` element and low-level 3D graphics APIs.
 - Since the `<canvas>` content for WebXR sessions doesn't coexist with the surrounding HTML elements that contain it, it can display true volumetric 3D content. However, it still has two `<canvas>` issues: it doesn't support Unified Rendering, and it's hard for regular 2D developers to use.

59

2. What Solutions or APIs Are Required to Solve These Problems

60

Break WebXR's limits in [Spatial UI](#) and [Spatial Development](#):

Solution Pillar 1 of 3:

A New Dev Paradigm Enabling Unified Rendering and Developer Friendliness

We need a new XR development system, distinct from WebXR - one that completely avoids custom rendering, lets the OS understand the content, fits into the Unified Rendering architecture and Shared Space, and is designed for regular 2D web developers by building on the familiar mainstream web ecosystem and mindset. It should continue to offer ease of use and high efficiency, while still offering enough spatial and 3D development power.



61

Close the gaps with [native 2D GUI frameworks and native spatial apps](#):

Solution Pillar 2 of 3

Enabling Spatial UI in the Mainstream HTML/CSS Web Ecosystem

The only solution is to introduce these spatial UI capabilities, which are currently exclusive to native 2D GUI frameworks, into the standardized, open, and mainstream HTML/CSS-based Web ecosystem at the earliest opportunity.



62

- Keep as much of the innovation and progress made in developing spatial apps as possible within the open ecosystem.
- It means that popular 2D web UI frameworks like React, which are based on HTML and CSS, would also gain spatial UI features.
- This also means that existing websites and web apps built with HTML/CSS should be able to transition into the new app ecosystem, gaining enhanced UI capabilities in spatial platforms without disrupting their current screen-based interfaces and primary user scenarios (desktop and mobile).
- Modern native 2D GUI frameworks like SwiftUI are actually very similar to HTML/CSS and web-based frameworks like React. They have comparable 2D UI capabilities and face the same challenges. So when bringing in spatial UI features, we can build on what these frameworks have already achieved, reusing their progress and lessons to avoid trial and error and reduce the cost of reaching consensus.

63

Systematically extend [spatial web](#) and [spatial browsing](#) trends toward a truly [2D-includes-3D](#) paradigm:

Solution Pillar 3 of 3

Blend into the 2D HTML/CSS/DOM APIs with Minimal Additions

A minimal but systematic HTML/CSS/JS API set, similar to SwiftUI's new APIs in visionOS, is required to extend existing 2D HTML/CSS/JS APIs specifically for spatial UI requirements. This extension should integrate seamlessly with the original 2D APIs, enabling spatial functionality to be implemented with precision and flexibility while preserving existing content and development experiences with minimal cost and disruption.



64

Proposed API 1

Fix the issue that [HTML/CSS layouts don't really have a Z-axis](#).

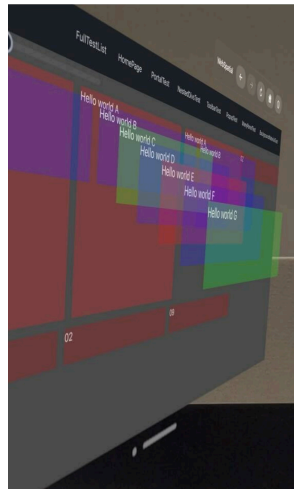
65

Proposed API 1: Depth

Depth Layout

```
p {
  --xr-background-material: transparent;
  position: absolute;
  bottom: -10px;
  left: 0;
  right: 0;
  --xr-back: 20;
}
```

- `back: 0.5m`
- `element.offsetBack === "0.5m"`
- "Box model" layout along the Z-axis? ("Cube model"?)



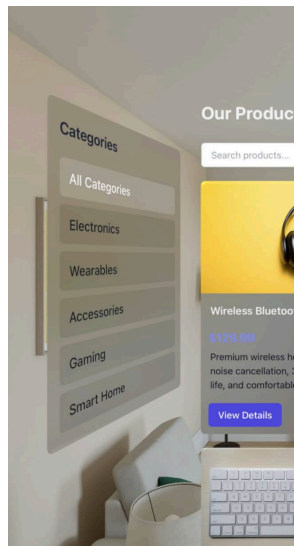
66

Proposed API 1: Depth

Spatial Transform

```
.list-menu {
  position: fixed;
  top: 200px;
  left: 0;
  transform-origin: top left;
  transform: translateZ(320px) rotateY(80deg);
}
```

- `transform-style: spatial`
- `translateZ()`,
`translate3d()`



67

Proposed API 2

Fix the issue where HTML/CSS only support fixed solid colors and you cannot remove the window background and border.

68

Proposed API 2: Material

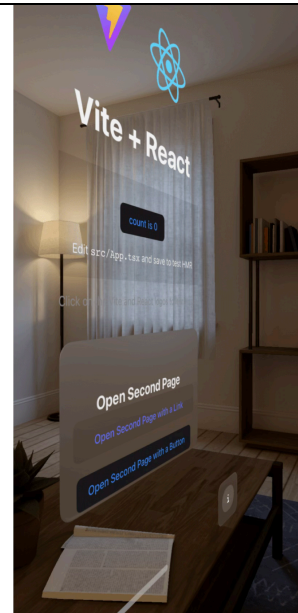
- background-material: translucent
- background-material: transparent
- background-material: regular | assist
- background-material: thick | embed
- background-material: thin | highlight

```
html.is-spatial {
  background-color: transparent;
  --xr-background-material: transparent;

  .count-card {
    --xr-background-material: thick;
    position: relative;
  }

  .link-card {
    --xr-background-material: translucent;
    border-radius: 20px;
    position: relative;
    top: 20px;

    a {
      display: block;
      --xr-background-material: thick;
      border-radius: 10px;
    }
  }
}
```



69

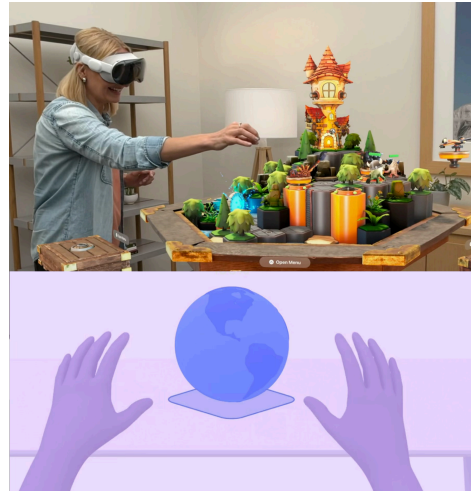
Proposed API 3

Fix the problem that HTML cannot support natural interactions in 3D space.

70

Proposed API 3: Spatial Events

- High-level Spatial Events
 - "spatialclick"
 - "spatialdrag", "spatialdragstart", "spatialdragend"
 - "spatialrotate", "spatialrotatestart", "spatialrotateend"
 - "spatialmagnify", "spatialmagnifystart", "spatialmagnifyend"
- Low-level Spatial Events
 - "spatialevent"
- 71 • Custom Hover Effect



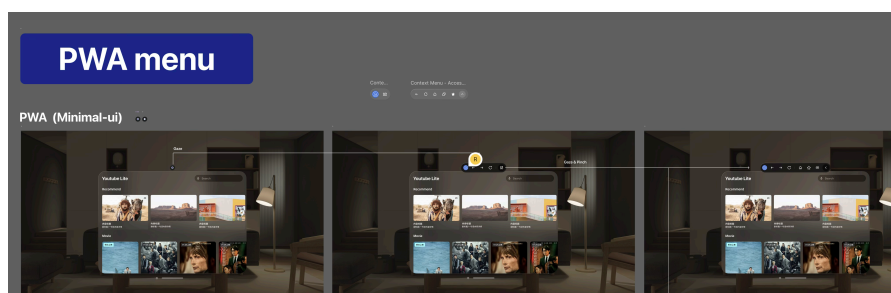
Proposed API 4

Fix the problem where enabling spatial UI in traditional browsers forces you [into a special dedicated mode](#) or [makes you install a PWA first](#)

72

Proposed API 4: Install-free PWA

Reuse and extend the standalone window experience provided by the Web App Manifest and PWAs, without sacrificing the web's advantage in delivering install-free apps.



73

Proposed API 5

Fix the problem where current web window APIs don't support initializing spatial scene containers or handling different scene types.

74

Proposed API 5: Spatial Scenes

Initialize the Start Scene

- Web App Manifest
 - "start_url"
 - "start_scene"
 - "type": "window" | "volume" | "stage"
 - "defaultSize"
 - ...

Create/Manage More Scenes

- `<a href={newSceneUrl} target="_blank">`
- `<a href={newSceneUrl} target="newSceneName">`
- `window.open(newSceneUrl);`
- `window.open(newSceneUrl, "newSceneName");`

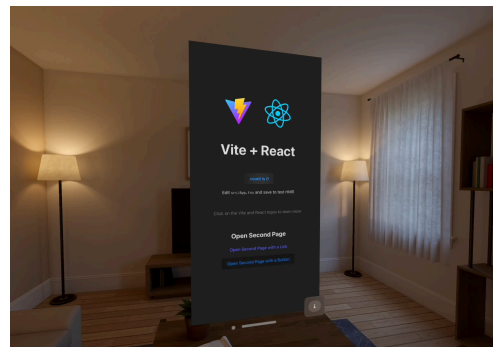
75

Scene Initialization

- type - 'window' | 'volume' | 'stage'
- defaultSize - { width, height, depth }
- resizable
- worldScaling - 'dynamic' | 'dynamic'
- worldAlignment - 'dynamic' | 'gravityAligned'
- ...

Scene Status (Size)

- `window.innerWidth`,
`window.outerDepth`



76

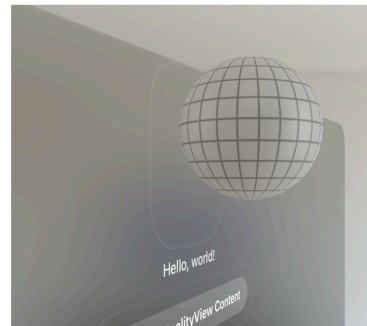
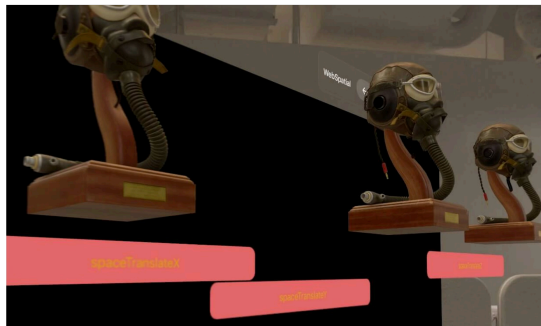
Proposed API 6

Expand what the [existing model element](#) can do, and fix the issue that HTML [only has 2D elements and no real 3D elements](#)

77

Proposed API 6: 3D Containers

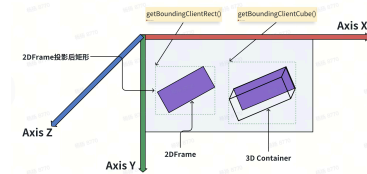
```
<model style="width: 400px; height: 300px" autoplay stagemode="orbit">  
  <source src="assets/example.usdz" type="model/vnd.usd+zip">  
  <source src="assets/example.glb" type="model/gltf-binary">  
</model>
```



78

Proposed API 6: 3D Containers

- depth
- clientDepth
- getBoundingClientRect()
- getBoundingClientCube()



79

Proposed API 7

Fix the issue where the [current canvas element](#) can't serve as the bridge between the 2D and 3D worlds in a "2D containing 3D" development paradigm.

80

Proposed API 7: 3D Engine Elements

```

1 <reality style="width: 100%; height: 50px; --xr-depth: 200px; position: relative; --xr-back: 50px;">
2   <asset id="exampleModel">
3     <source type="model/vnd.usdz+zip" src="/models/example.usdz" />
4     <source type="model/gltf-binary" src="/models/example.glb" />
5   </asset>
6   <asset id="albedoTexture">
7     <source type="image/avif" srcset="/textures/wood/albedo.avif" />
8     <source type="image/webp" srcset="/textures/wood/albedo.webp" />
9   </asset>
10  <material id="exampleMat" type="unlit" texture="albedoTexture" />
11  <div attachment="info-panel"><p>Hello, World!</p></div>
12  <button attachment="my-button">Click me</button>
13  <scenegraph>
14    <entity>
15      <entity model="exampleModel" material="exampleMat" position="0 0 0" rotation="0 0 0" />
16      <box position="0 0 0" rotation="0 0 0">
17        <box position="0 0 0" rotation="0 0 0" />
18        <sphere position="0 0 0" rotation="0 0 0" />
19      </box>
20    </entity>
21    <plane position="0 0 0" rotation="0 0 0" />
22    <cone position="0 0 0" rotation="0 0 0" />
23    <cylinder position="0 0 0" rotation="0 0 0" />
24    <attachment name="info-panel" position="0 0 0" rotation="0 0 0" />

```

81

Proposed API 7: 3D Engine Elements**Asset Declaration**

- `<asset>`
 - `or(<prefab>, <texture>)`
 - `with <source />`
- `<material />`
- `<div attachment="info">`

Entity Properties for Built-in Components

- Transform: `<entity position="0 0 0" rotation="0 0 0" />`
- Enable Physics Engine
- Enable Particle Effects
- ...

Entities with Built-in Components

- Root: `<world>`
- Primitives: `<box>`, `<sphere>`, `<plane>`, `<cone>`, `<cylinder>`
- Attachments: `<attachment name="info">`
- Models: `<entity prefab="exampleModel" />`
- ...

82

More missing APIs

- Ornaments / Orbiter
- Full Space + AR
- ...

Explore cutting-edge APIs

- WebXR Sessions as Dynamic 3D Content Containers
- Spatial DOM Overlays for WebXR

83

3. Discussion and To-Dos

- [?] Whether this systemic issue should be addressed by forming a new W3C group or by leveraging existing groups.
- [?] Set up a GitHub repo.
- [] Gather feedback and practical experience on different parts of the API.

84

4. Real-World Practices

85

<https://webspatial.dev>

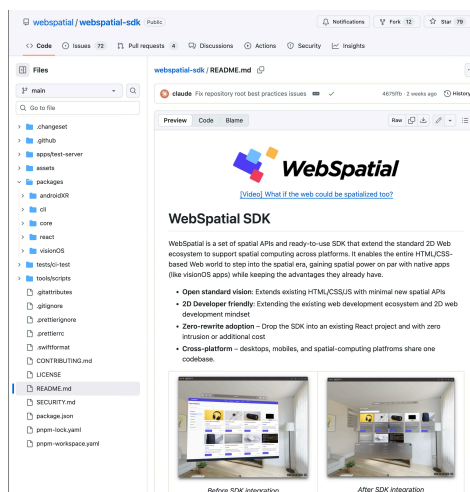
Build an open-source SDK project called WebSpatial SDK based on existing declarative web frameworks and web build tools. This way, developers can start using WebSpatial APIs right away in HTML/CSS-style APIs like JSX and CSS-in-JS, without having to wait for browser engines to support them.

Hybrid technology will be used to implement the WebSpatial Runtime across different spatial app platforms. On platforms where the WebSpatial Runtime can be integrated into the browser, web developers just need to run the site URL as a PWA to enable spatial UI. For platforms where the browser can't be modified, developers can add a packaging step to their workflow (similar to using Electron) to pre-package the site as a PWA, with the WebSpatial Runtime bundled in.

86

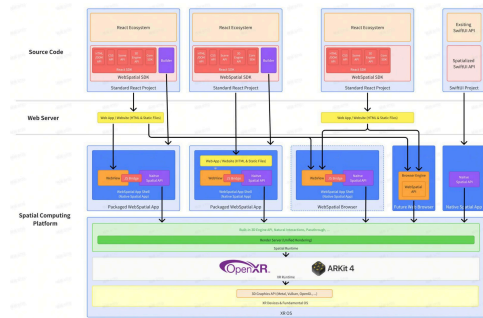
<https://github.com/webspatial/webspatial-sdk>

- Already supports React and various frameworks and web build tools in the React ecosystem
- Already supports visionOS
- Planned support for Android XR and more.
- Depth Layout, Spatial Transform, Material Background, and Spatial Scene are already supported.
- APIs related to 3D content - like Spatial Events, Volume, 3D Container Elements, and the 3D Engine API - have been merged into the main branch and are currently being



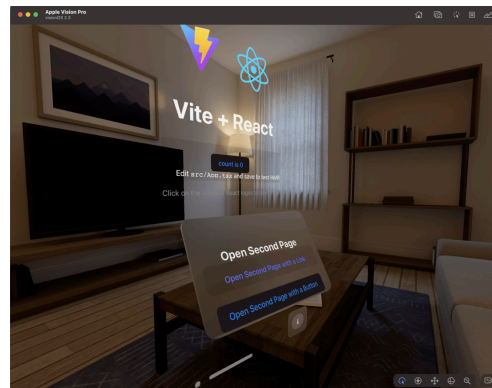
<https://github.com/webspatial/webspatial-sdk>

- High priority on minimizing the integration cost for existing websites
 - Ready-to-use optimization solutions are provided to ensure the original UI and main user scenarios (desktop and mobile) remain unaffected.
 - A new solution will make it even simpler - one website and one URL can support both screen-based and spatial devices without interfering with each other.



88

WebSpatial Builder provides full support for packaging, simulator-based debugging, real device testing, and submission to the visionOS App Store, eliminating the need to interact with Xcode or native app shells throughout the entire process.

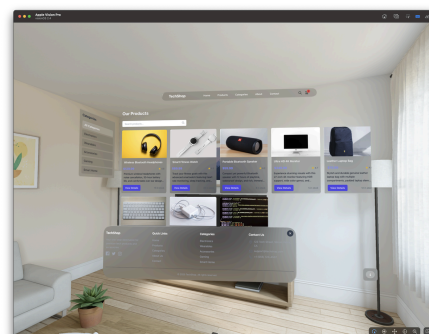
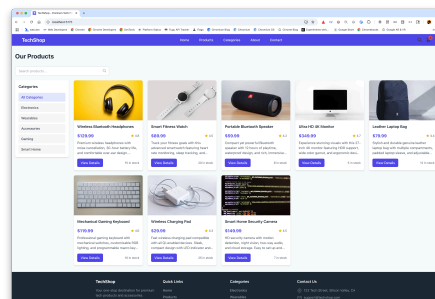


89

Demo - Techshop

<https://github.com/webspatial/sample-techshop>

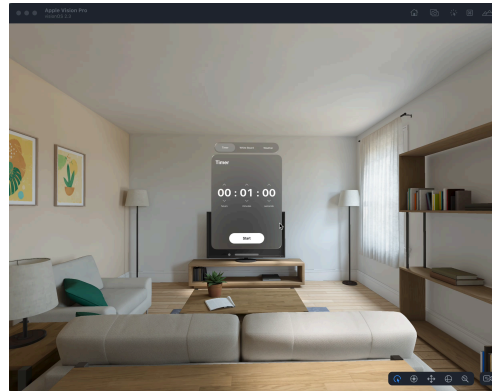
Just by adding some CSS styles and a bit of if-else logic to existing website, you can have a completely different spatialized UI on visionOS.



90

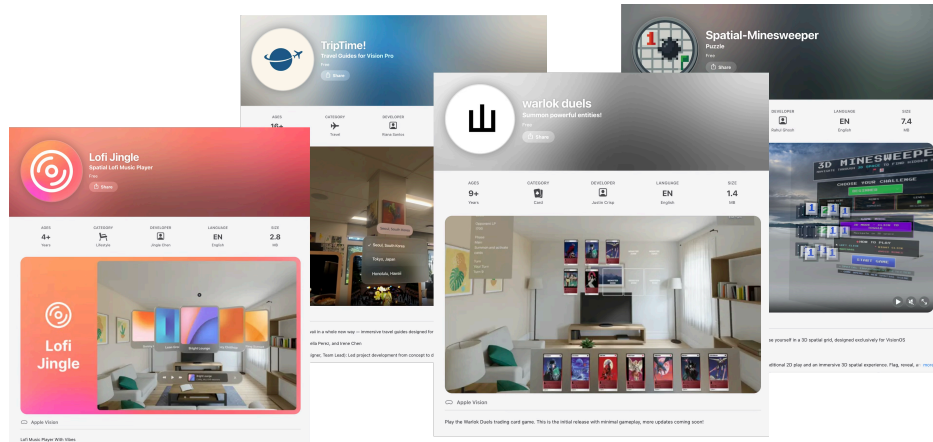
Demo - Widget Generator

<https://github.com/webspacial/widget-generator>



91

Real App Store Apps Built with WebSpatial From the Developer Community



92

Real App Store Apps Built with WebSpatial From the Developer Community

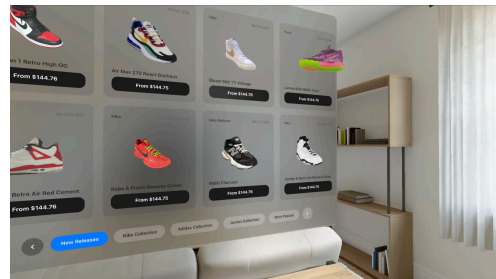
<https://youtu.be/VaoG2SETVKw?si=EVQa1hBmCMIMUwue>



93

Real App Store Apps Built with WebSpatial From the Developer Community

<https://youtu.be/6qTAYwV8B5M?si=vj05bJB8EfF1G4hl>



94

Early Feedback and Data

- Fluid's founder highly praised WebSpatial, believing it can efficiently transform their Next.js + React smart TV interface into a 3D spatial media application.
- Each of the real apps shown earlier was built solo in a little over a week of spare time.
- Tried out a hackathon event called "Web to Spatial"
 - 95% of developers have no issue following the "Quick Example" to start a WebSpatial app
 - Not too many problems with using spatialized 2D HTML APIs.
 - Many requests on 3D APIs (we hadn't yet supported APIs like Spatial Events, <reality>, and <entity> during the event)
 - The main blocker was the complex Apple Store submission process, needing detailed docs.

95

