On the Declarative Immersive Web

@joshcarpenter
UX Lead for WebXR, Google
Mar 2 2017, Khronos Meetup
Is it the web, if...

It’s HTTP, but there’s no DOM?

It’s HTML, CSS and JavaScript, but it’s a native app?

There are hyperlinks, but no URL bar?
Secure
Linkable
Indexable
Composable
Ephemeral
Secure
Linkable
Indexable + Neutral, standards-compliant
Composable
Ephemeral
user agents
What is the unit of content?
The means of production?
Of distribution?
Of discovery?
Of consumption?
WebGL is amazing.
If it didn’t exist we’d have to invent it. It enabled us to kick start WebVR without new standards back in 2014, and gave the web a fighting chance at meeting baseline 3D graphics performance for VR. Today, Web VR = WebGL. And developers are doing amazing things with it.
A-Painter, from MozVR. A WebGL WebVR homage to Tilt Brush. Loads in seconds and is cross platform, shareable and extensible.
Archan Naira: a characteristically beautiful WebGL WebVR collaboration from Marpi
SketchFab: A vast collection of stupendous 3D art, surfable in WebGL WebVR.
But WebGL has some shortcomings.
Most web developers don’t know WebGL, the learning curve is steep, and there is little interoperability with existing web content. WebGL is also missing basic 2D layout features that we take for granted from HTML and CSS. If Wikipedia wanted to create a WebVR version of their site, it would be a massive undertaking.

These factors are keeping the vast majority of today’s web devs and sites on the sidelines of WebVR.
WebGL is also problematic from a usability point of view.

The classic web was wonderfully consistent. You learned how to use one site, and you knew how to use most of them. The content was written in standardized parseable building blocks, and the browser + OS provided most of the interactivity, encapsulating WIMP and then mobile UX conventions—scrollbars, cursors, inertial scroll, etc—"for free". This limited developer freedom until lower-level APIs like WebGL came along, but helped the web scale to take over the world.
WebGL WebVR is a lot more like Flash.

Maximum developer freedom, almost by definition, results in site-to-site UX and performance inconsistency. This is exacerbated by VR, which is 1) a brand new medium with new conventions, and 2) far more fragmented than desktop or mobile.
We’ve made the right choices so far. Maximizing for developer freedom through low-level APIs has helped us collectively discover the jobs we’re going to hire the VR web to do. It’s empowered SketchFab, the Washington Post, and a community of WebGL creatives to explore new ideas without waiting for permission from new standards.

But on its own, I don’t think it will scale to desktop or mobile web volumes.

The ideal recipe for the web is: guaranteed minimum experience + freedom to innovate. We’ve got the freedom. Now it’s time to add the guaranteed minimum experience. And empower everyone on the sidelines to join the party.
I’m interested in the potential of HTML and CSS.

Developers should be able to create compelling VR experiences, or adapt existing sites, without having to use WebGL. HTML and CSS are the languages we have. They may not seem obvious candidates to be great VR design tools, but we believe they will be, with relatively little effort. From experience.
You're browsing on your mobile device...
You drop your phone into a VR headset.
Your site just works, “for free”. But the site is stuck inside a small window. Developers should be able to fill the entire 360 environment.
What if developers could, with CSS: apply a 360 background, make the window transparent, and re-position elements in 3D space?
Basic concept prototype, made with A-Frame. 3D repositioning not used in this one.
Concept prototype, made with A-Frame. 3D repositioning not used in this one.
CSS 3D transforms in true 3D space
CSS 3D transforms in true 3D space
We’ll need new permissions. We can imagine the user agent cropping elements by default, until a new fullscreen permission analogue is granted.
Once that fullscreen permission is granted, DOM elements can expand into the full 360 environment.
Adding models should be as easy as adding images. A `<model>` element with glTF support would trigger an explosion of creative possibilities.

This mockup depicts a site with an embedded model as the user agent shifts between display modes: from flattened 2D, to 3D “magic window” (still cropped to a window, but allowed to extend infinitely on the z-axis away from the user).
The web is already fluid on the X and Y axes. We can imagine it being fluid on the Z as well. Expanding or flattening at any time, depending on the viewing mode (e.g., embedded in a native app, versus open in a VR browser like Chrome or Carmel).
Longer term, imagine CSS Physically Based Materials.
CSS Physically Based Materials: Imagine a web that becomes more beautiful over time, as compute power increases and browsers begin to compete based on ability to render materials like glass or metal. Browsers progressively enhancing visual richness to match client compute power, ala game engines.

Minecraft with “Sonic Ether’s Unbelievable Shaders”: Same underlying textures, but far greater visual richness, and compute requirements.
With declarative, parseable content, we can also build amazing new browser features. It will be important to preserve user comfort on the wide open VR web. We can imagine creating personal safety zones, within which elements are occluded or pushed away. Adjustable by the user, with sites needing to request permission to enter the user’s personal space.
With declarative, parseable content, we can also build amazing new browser features. It will be important to preserve user comfort on the wide open VR web. We can imagine creating personal safety zones, within which elements are occluded or pushed away. Adjustable by the user, with sites needing to request permission to enter the user’s personal space.

https://www.daydream.org wants to: Render inside your personal space

Block  Allow
Today we can drag individual images to our desktop, or long press a link to open a contextual menu. The equivalents in VR are amazing to imagine. Plucking elements out of the 3D world to examine them more closely, through motion controller direct manipulation. Or completely re-presenting a scene for a future reader mode analogues, live developer tools, and so on. This would be impossible to do at scale on imperative black box content.
Some new potential building blocks...

VR-specific media features (enabling VR media queries)
360 backgrounds
True 3D positioning of DOM elements
Native 360 stereo `<video>`
`<model> + glTF`
Physical units and true camera model
Mixed WebGL + DOM (via shared depth buffer?)
DOM-to-texture? Would help WebGL devs, and could help unlock new polyfills for the above.
And a default interaction model, probably based on a common denominator laser-pointer metaphor. Compatible with classic content, while also working for new immersive content. Boris Smus’s new Ray.js is a great step in this direction.
We built a prototype at Mozilla. It was surprisingly compelling.

In 2015 the MozVR team built a DOM-only VR web browser. It presented 3D transformed DOM elements in stereo, and worked with the Oculus DK2. It was buggy, but worked. I used divs to build scenes in a fraction the time as Three.js, and got all of the DOM’s good parts (great text handling, 2D layout tools, iframes, etc) for free. We made browser interfaces, video viewers, sky boxes, and more. My biggest surprise was just how fun it was. Making an iframe Vimeo player took minutes. The experiences I created weren’t going to compete with games for sheer awe and wonder, but they were content rich, instantly consumable, and easy to create with basic skills.
CSS VR browser prototype, May 2015. HTML elements, 3D transforms and keyframe animations. Zero WebGL.
“MS-DOS Neuromancer running in Firefox via Emscripten DOSBox, presented in WebVR”. One of my more popular tweets :) It took less than ten minutes to make an iframe, target a DOSBox ASM.js copy of Neuromancer on archive.org, blow the iframe up to IMAX size and position, and experience it in VR with a DK2.
We built models in TriDiv, a modeling web app that generates textured "meshes" from divs and css.
Three.js’ CSS Renderer is already pretty fast today. Three is known primarily for WebGL, but its CSS Renderer is fast and cross-platform. Periodic Table example.
Via the extensible web. Ideas gestating in JS frameworks to start. We learn from experimentation in the WebGL WebVR ecosystem, then pave the cowpaths with new standards and built-in browser features. This process is already underway, with experiences built on tools like Three.js, A-Frame, ReactVR, Vizor, PlayCanvas, etc.
Thank you!