Seeing the Web from a VR Perspective

Caution: This article is written for WebVR 1.1, not the WebXR Device API, which is still in development and subject to change.

There are several things about creating WebVR content that are not like regular web development. This means that you need different techniques and a different way of thinking.

If you've done game development - particularly web game development - then some of this may be familiar to you. But even for seasoned HTML5 game pros, there are some WebVR specific things that you need to think about.

Performance is everything

If you create animations on the web, you should already want them to be silky smooth at 60 frames per seconds. That said, if sometimes you don't quite make it because you are processing something in the background it isn't always a huge deal.

With WebVR, janky animation could very literally make your users feel ill. You need to make sure that you aren't just hitting your frame timing most of the time, but that you have enough headroom on every frame that you can handle any variance that the device throws at you.

Everyone wants to pack as much awesome as they can into their VR apps, but your complete experience just may not be possible to do at full speed on all of your target devices. Make sure that you allow the content to scale with the performance of the device.

You can use the high-resolution timestamp from `performance.now()` to measure the time between successive frames, and if you find that you aren't keeping up you can dynamically scale back things like number of objects, draw distance, lighting quality, etc.

The display is not the window

If you are testing on a desktop VR headset then it is pretty easy to see that the VR content is being displayed on a completely separate device. However, if you are using a mobile VR viewer like Daydream, it can be hard to remember the distinction.

Use the correct frame rate

On any screen, the hardware will update every pixel at a fixed frequency, called the frame rate. For many modern displays this will be 60 times per second.

No matter how quickly or slowly your code draws to your canvas, the actual pixels the user sees will be only be updated at exactly this frequency. So if you draw your scene 100 times per second you will be wasting your time. And if you update 50 times per second you will have some physical frames where the display doesn't actually change, which the user will notice as a stutter.

For many years the web has had an API called `requestAnimationFrame` that helps you to draw to the screen at the correct rate. Once per frame your callback will be called, letting you know that it is time to draw.

With WebVR it is important to remember that `requestAnimationFrame` will be called at the correct frequency for the display that the web page is being shown on - NOT the VR display. Some VR displays have frame rates of 90 or even 120 frames per second. So instead of using the normal `requestAnimationFrame` function you need to call the `requestAnimationFrame` method on the same `VRDisplay` object that you call `requestPresent` on.

```
display.requestAnimationFrame(mainLoop);
```

Even on mobile, where it is the same physical display with the same frame rate whichever method you call, it is still important to use the display's `requestAnimationFrame` method. The browser may throttle or pause `window.requestAnimationFrame` because technically the page is not visible - it is hidden behind the system's VR presentation layer.

Use the correct canvas size

The canvas that you present to the WebVR display should be sized to be correct for VR display. Ideally the size of the canvas should be the same as the size of the display, but for performance reasons you can use a smaller canvas and it will be stretched to fill the whole display.

To avoid aliasing artifacts you probably want to use a canvas that is an integer ratio of the full display. For example, if the display has a size of 2880x1800 you could create a canvas with a 1:2 ratio that was 1440x900.

To find the size of the display, use the `getEyeParameters` method on the `VRDisplay`.

```
const leftEye = display.getEyeParameters('left');
const rightEye = display.getEyeParameters('right');

const width = 2 * Math.max(leftEye.renderWidth, rightEye.renderWidth);
const height = Math.max(leftEye.renderHeight, rightEye.renderHeight);
```

There is no DOM

Many HTML5 games will use DOM elements to render UI so that they can take advantage of the browser's layout engine. With WebVR, you can currently only send a canvas to be presented to the display. Everything that you want the user to be able to see must be drawn onto the canvas.

Don't use click or touch events

If you are using a Google Cardboard viewer and you press the button, this is detectable by the hardware as a touch event. This makes it tempting to listen for touches in your WebVR app as a way to tell when the button is pressed.

However, Daydream View, for example, uses screen touches to detect the position of the mobile device within the viewer to make sure that the presented image lines up with the lenses.

And of course, non-mobile VR headsets will not trigger touch events at all.

Instead, use the Gamepad API. In upcoming versions of Chrome a Google Cardboard viewer will be represented as a single button gamepad, and the state of other VR controllers will need to be read in this way anyway.

Unlike mouse and touch interaction, which are event based, the Gamepad API relies on polling each frame. The `navigator.getGamepads` method gives you a snapshot of the current state of all connected gamepads. You must call this once per frame as part of your frame animation loop.

If you simply want to detect if any gamepad button is currently pressed you could use code like:

```
const gamepads = navigator.getGamepads();
let pressed = false;
```

```
for (let i = 0; i < gamepads.length; i++) {
  for (let j = 0; j < gamepads[i].buttons.length; j++) {
    if (gamepads[i].buttons[j].pressed) {
      pressed = true;
    }
  }
}
```

Last updated April 25, 2018.