Augmented reality for the web

By Joseph Medley
Technical Writer

In Chrome 67, we announced the WebXR Device API for both augmented reality (AR) and virtual reality (VR), though only the VR features were enabled. VR is an experience based purely on what's in a computing device. AR on the other hand allows you to render virtual objects in the real world. To allow placement and tracking of those objects, we just added the WebXR Hit Test API to Chrome Canary, a new method that helps immersive web code place objects in the real world.

Where can I get it?

This API is intended to stay in Canary for the immediate future. We want a protracted testing period because this is a very new API proposal and we want to make sure it's both robust and right for developers.

Aside from Chrome Canary, you'll also need:

- A compatible smartphone running Android O or later.
- To install ARCore.
- Two Chrome flags (chrome://flags): WebXRDevice API (#webxr) and WebXR Hit Test (#webxr-hit-test)

With these, you can dive into the demos or try out our codelab.

Note: Some of the Immersive Web Community Group's existing demos, specifically the ones using magic windows, do not work with the WebXR Hit Test turned on. Please excuse our construction debris.

- Chacmool
- Immersive Web Sample
- Codelab

It's just the web

At Google IO this year, we demonstrated augmented reality with an early build of Chrome. I said something repeatedly to developers and non-developers alike during those three days that I wish I had known to put in my immersive web article: "It's just the web."

"What Chrome extension do I need to install?" "There's no extension. It's just the web."

"Do I need a special browser?" "It's just the web."

"What app do I need to install?" "There is no special app. It's just the web."

This may be obvious to you since you're reading this on a website devoted to the web. If you build demonstrations with this new API, prepare for this question. You'll get it a lot.

Speaking of IO, if you want to hear more about the immersive web in general, where it is, where it's going check out this video.

What's it useful for?

Augmented reality will be a valuable addition to a lot of existing web pages. For example, it can help people learn on education sites, and allow potential buyers to visualize objects in their home while shopping.

Our demos illustrates this. They allow users to place a life-size representation of an object as if in reality. Once placed, the image stays on the selected surface, appears the size it would be if the actual item were on that surface, and allows the user to move around it as well as closer to it or farther from it. This gives viewers a deeper understanding of the object than is possible with a two dimensional image.

If you're not sure what I mean by all of that, it will become clear when you use the demos. If you don't have a device that can run the demo, check out the video link at the top of this article.

One thing that demo and video doesn't show is how AR can convey the size of a real object. The video here shows an educational demo that we built called Chacmool. A companion article describes this demo in detail. The important thing for this discussion is that when you place the Chacmool statue in augmented reality, you're seeing its size as though it were actually in the room with you.

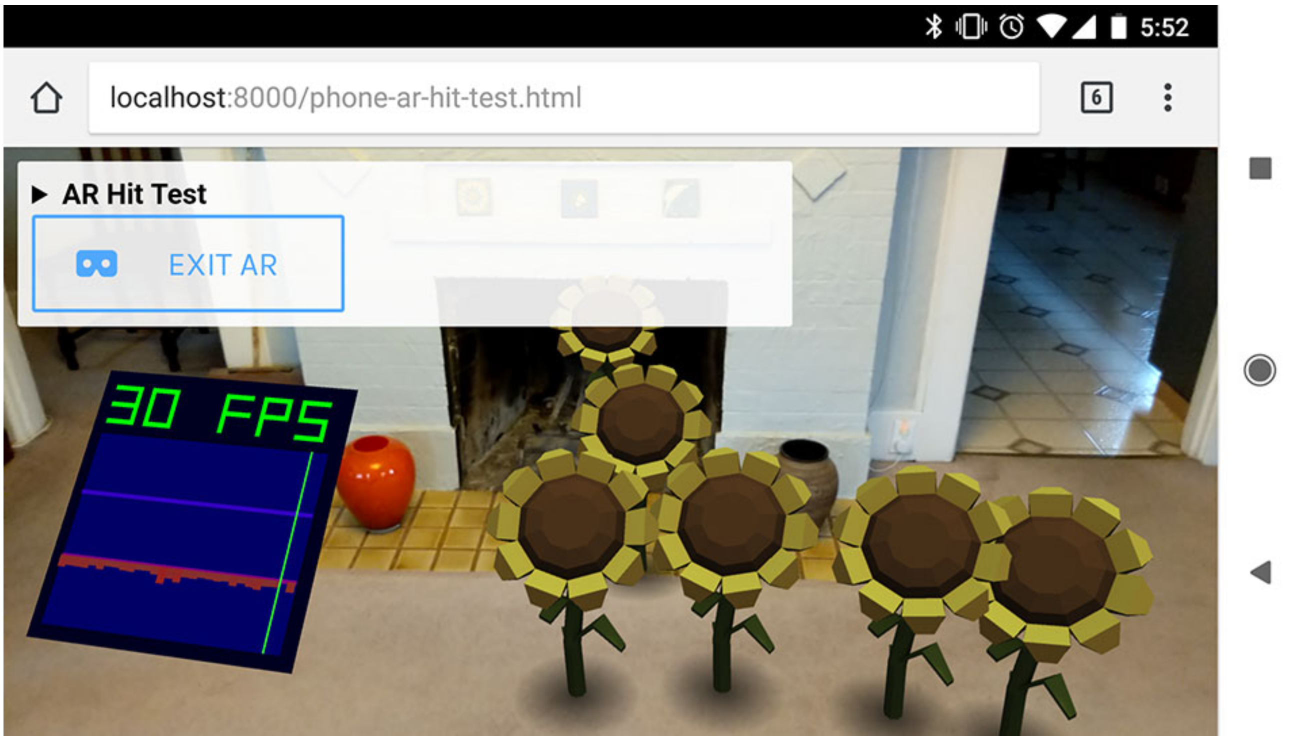
The Chacmool example is educational but it could just as easily be commercial. Imagine a furniture shopping site that lets you place a couch in your living room. The AR application tells you whether the couch fits your space and how it will look next to your other furniture.

Ray casts, hit tests, and reticles

A key problem to solve when implementing augmented reality is how to place objects in a real-world view. The method for doing this is called *ray casting*. Ray casting means calculating the intersection between the pointer ray and a surface in the real world. That intersection is called a *hit* and determining whether a hit has occurred is a *hit test*.

This is a good time to try out the new code sample in Chrome Canary. Before doing anything, double-check that you have the correct flags enabled. Now load the sample and click "Start AR".

Notice a few things. First, the speed meter which you may recognize from the other immersive samples shows 30 frame per second instead of 60. This is the rate at which the web page receives images from the camera.



The AR Hit Test demo

The other thing you should notice is the sunflower image. It moves as you move and snaps to surfaces such as floors and table tops. If you tap the screen, a sunflower will be placed on a surface and a new sunflower will move with your device.

The image that moves with your device, and that attempts to lock to surfaces is called a *reticle*. A reticle is a temporary image that aids in placing an object in augmented reality. In this demo, the reticle is a copy of the image to be placed. But it doesn't need to be. In the Chacmool demo, for example, it's a rectangular box roughly the same shape as the base of the object being placed.

Down to the code

The Chacmool demo shows what AR might look like in a production app. Fortunately, there is a much simpler demo in the WebXR samples repo. My sample code comes from the AR Hit Test demo in that repository. FYI, I like to simplify code examples for the sake of helping you understand what's going on.

The basics of entering an AR session and running a render loop are the same for AR as they are for VR. You can read my previous article if you're unfamiliar. To be more specific, entering and running an AR session looks almost exactly like entering a VR magic window session. As with a magic window, the session type must be non-immersive and the frame of reference type must be `'eye-level'`.

The new API

Now I'll show you how to use the new API. Recall that in AR, the reticle attempts to find a surface before an item is placed. This is done with a hit test. To do a hit test, call **`XRSession.requestHitTest()`**. It looks like this:

```
xrSession.requestHitTest(origin, direction, frameOfReference)
.then(xrHitResult => {
  //
});
```

The three arguments to this method represent a ray cast. The ray cast is defined by two points on the ray (**`origin`** and **`direction`**) and where those points are calculated from (**`frameOfReference`**). The origin and direction are both 3D vectors. Regardless of what value you submit, they will be normalized (converted) to a length of 1.

Moving the reticle

As you move your device the reticle needs to move with it as it tries to find a location where an object can be placed. This means that the reticle must be redrawn in every frame.

Start with the **`requestAnimationFrame()`** callback. As with VR, you need a session and a pose.

```
function onXRFrame(t, frame) {
  let xrSession = frame.session;
  // The frame of reference, which was set elsewhere, is 'eye-level'.
  // See onSessionStarted() ins the sample code for details.
  let xrPose = frame.getDevicePose(xrFrameOfRef);
  if (xrPose && xrPose.poseModelMatrix) {
    // Do the hit test and draw the reticle.
  }
}
```

Once you have the session and the pose, determine where the ray is casting. The sample code uses the gl-matrix math library. But gl-matrix is not a requirement. The important thing is knowing what you're calculating with it and that it is based on the position of the device. Retrieve the device position from **`XRPose.poseModalMatrix`**. With your ray cast in hand, call **`requestHitTest()`**.

```
function onXRFrame(t, frame) {
  let xrSession = frame.session;
  // The frame of reference, which was set elsewhere, is 'eye-level'.
  // See onSessionStarted() ins the sample code for details.
  let xrPose = frame.getDevicePose(xrFrameOfRef);
  if (xrPose && xrPose.poseModelMatrix) {
    // Calculate the origin and direction for the raycast.
    xrSession.requestHitTest(rayOrigin, rayDirection, xrFrameOfRef)
    .then((results) => {
      if (results.length) {
        // Draw for each view.
      }
    });
  }
  session.requestAnimationFrame(onXRFrame);
}
```

Though not as obvious in the hit test sample, you still need to loop through the views to draw the scene. Drawing is done using WebGL APIs. You can do that if you're really ambitious. Though, we recommend using a framework. The immersive web samples use one created just for the demos called Cottontail, and Three.js has supported WebXR since May.

Placing an object

An object is placed in AR when the user taps the screen. For that you use the **`select`** event. The important thing in this step is knowing where to place it. Since the moving reticle gives you a constant source of hit tests, the simplest way to place an object is to draw it at the location of the reticle at the last hit test. If you need to, say you have a legitimate reason not to show a reticle, you can call **`requestHitTest()`** in the select event as shown in the sample.

Conclusion

The best way to get a handle on this is to step through the sample code or try out the codelab. I hope I've given you enough background to make sense of both.

We're not done building immersive web APIs, not by a long shot. We'll publish new articles here as we make progress.

Last updated July 17, 2018.