WONKAVISION

Could we make the web more immersive using a simple optical illusion?

In the history of mind-blowing tech demos there is one that stands out, and that is Johnny Lee's Wii Remote hack to create VR displays:

That video is from 2007, and yet the technology being shown is so impressive that it feels like it could have been made yesterday.

The response to Johnny's demo was universal acclaim. He presented it at a TED talk a year later, along with many other amazing hacks, and he went on to work on Microsoft's Kinect and Google's Project Tango.

Sadly, however, his VR display technique was never used by any Wii games or other commercial products that we know of.

Rewatching his demo earlier this year we wondered why his technique had not taken off. Everyone wanted to try it, but nobody ever did anything big with it.

We were also curious about whether we could implement his technique in the browser using only a webcam. We thought that if that was possible, it would make his technique accessible to everyone and it would open up a whole new world of interactive experiences on the web.

Imagine, for example, opening your favorite brand's website and being presented with a miniature virtual storefront. You could look at their most popular products as if you were standing on a sidewalk peering into their shop.

That was enough to get us excited, so we got to work on solving this problem.

Exploration

3D eye tracking with a webcam

The first thing that Johnny Lee's technique requires is a way to calculate where your eyes are in world space with respect to the camera. In other words, you need a way to accurately say things like "your right eye is 20 centimeters to the right of the camera, 10 centimeters above it, and 50 centimeters in front of it."

In Johnny's case, he used the Wii Remote's infrared camera and the LEDs on the Wii's Sensor Bar to accurately determine the position of his head. In our case, we wanted to do the same but with a simple webcam.

After researching this problem for a while we came across Google's MediaPipe Iris library, which is described as follows:

> *MediaPipe Iris is a ML solution for accurate iris estimation, able to track landmarks involving the iris, pupil and the eye contours using a single RGB camera, in real-time, without the need for specialized hardware. Through use of iris landmarks, the solution is also able to determine the metric distance between the subject and the camera with relative error less than 10%.*
>
> *[…]*
>
> *This is done by relying on the fact that the horizontal iris diameter of the human eye remains roughly constant at 11.7±0.5 mm across a wide population, along with some simple geometric arguments.*

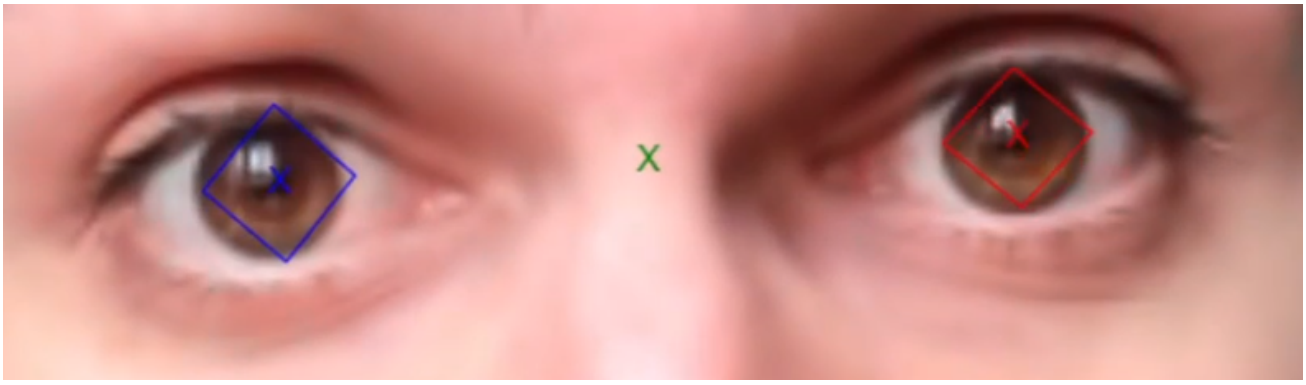
That's precisely what we needed, but there was one problem, however: MediaPipe Iris isn't exposed to JavaScript yet, and we wanted our demo to run in the browser.

	Android	iOS	C++	Python	JS	Coral
Face Detection	✅	✅	✅	✅	✅	✅
Face Mesh	✅	✅	✅	✅	✅	
Iris	✅	✅	✅			

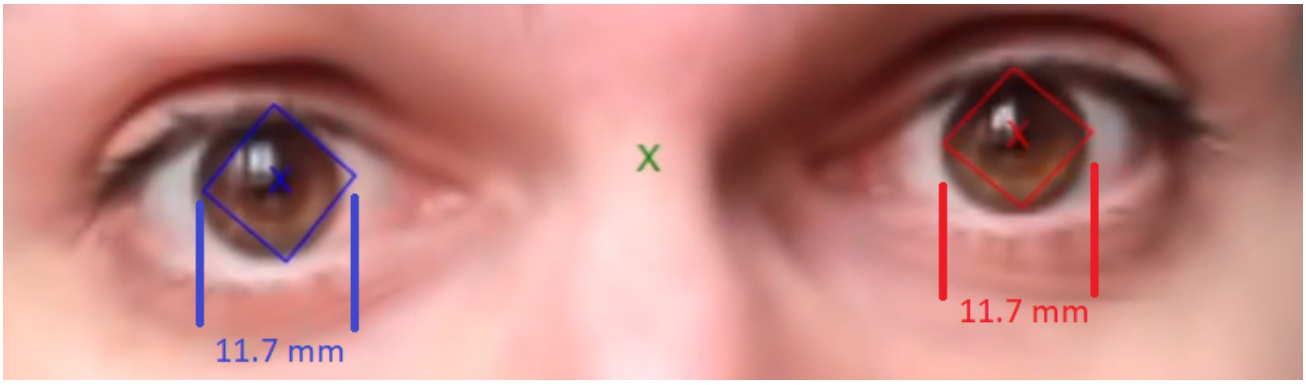
We were really disappointed by this, but then we came across this blog post that explains how Google's MediaPipe Face Mesh library can also be used to measure the distance between the camera and the eyes.

Here's a summary of how it's done:

- MediaPipe Face Mesh detects four points for each iris: left, right, top and bottom.



- We know the distance in pixels between the left and right points, and we also know that the world-space distance between them must be roughly equal to 11.7 mm (the standard diameter of a human iris).

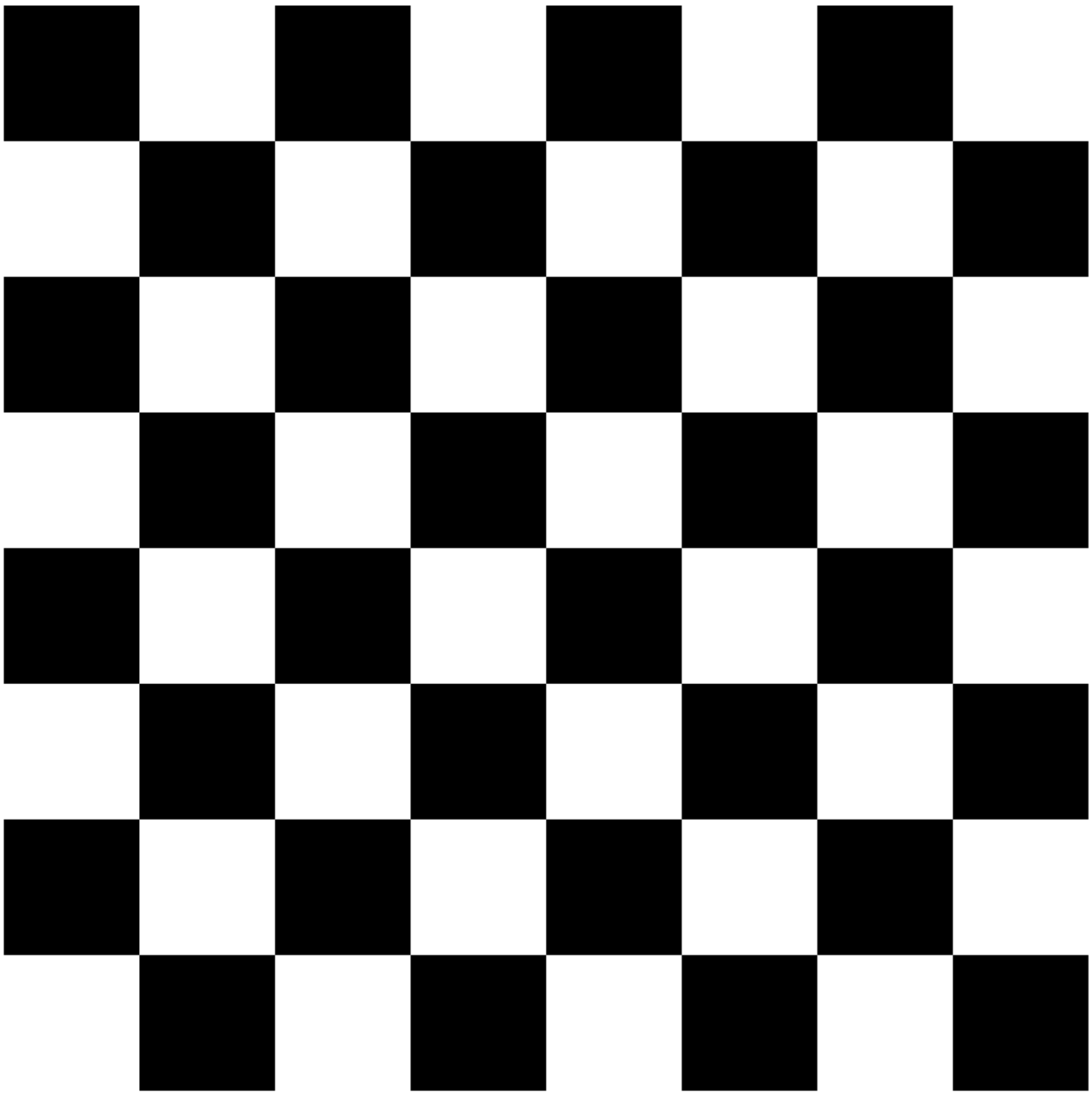


- Using those values and the intrinsic parameters of our webcam (the effective focal length in X and Y and the position of the principle point in X and Y) we can solve some simple pinhole camera model equations to determine the depth of the eyes in world space, and once we know their depth we can also calculate their X and Y positions.

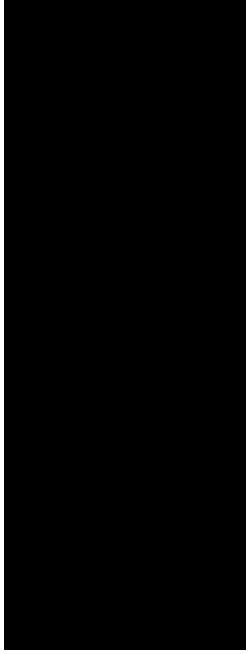
It sounds difficult, but it's surprisingly easy to do.

The only problem then became how to calculate the intrinsic parameters of our webcam. The previous blog post also answered that for us by pointing us to this blog post, which explains it as follows:

- First you print out this checkerboard pattern and you tape it to a hard surface like a piece of cardboard:

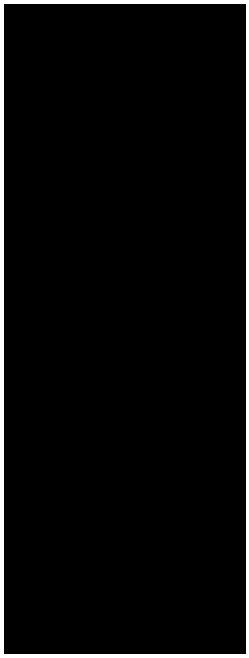


- Then you take around 50 photos of it at different angles using your webcam.



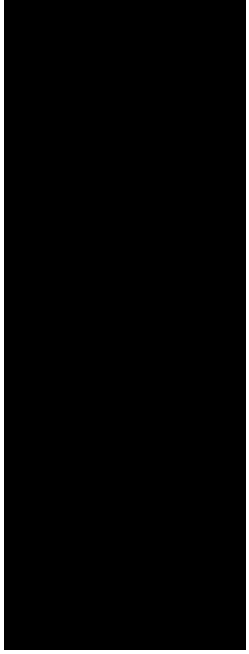
0:00

- Finally, you feed the photos to a Python script that uses OpenCV to detect the corners of the checkerboard pattern, which then allows it to calculate the intrinsic parameters of your webcam.



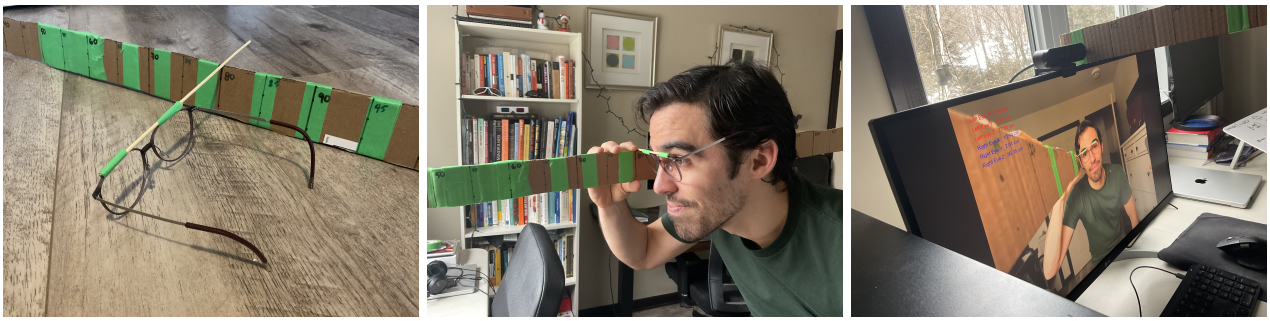
0:00

Here you can see the eye tracking in action, with the world space positions of the eyes being displayed in the top left corner:



0:00

To validate that the depth values we were computing were correct we used MediaPipe's DIY approach of taping a stick to a pair of glasses and then sliding our eyes along a big ruler towards the webcam:



The results are accurate enough for our purposes, but they would definitely be better if we could use MediaPipe Iris instead of MediaPipe Face Mesh.

Turning the screen into a virtual window

With 3D eye tracking out of the way we then focused on implementing the actual optical illusion.

The easiest way to explain it is to start by looking at the screen that will be displaying the effect:

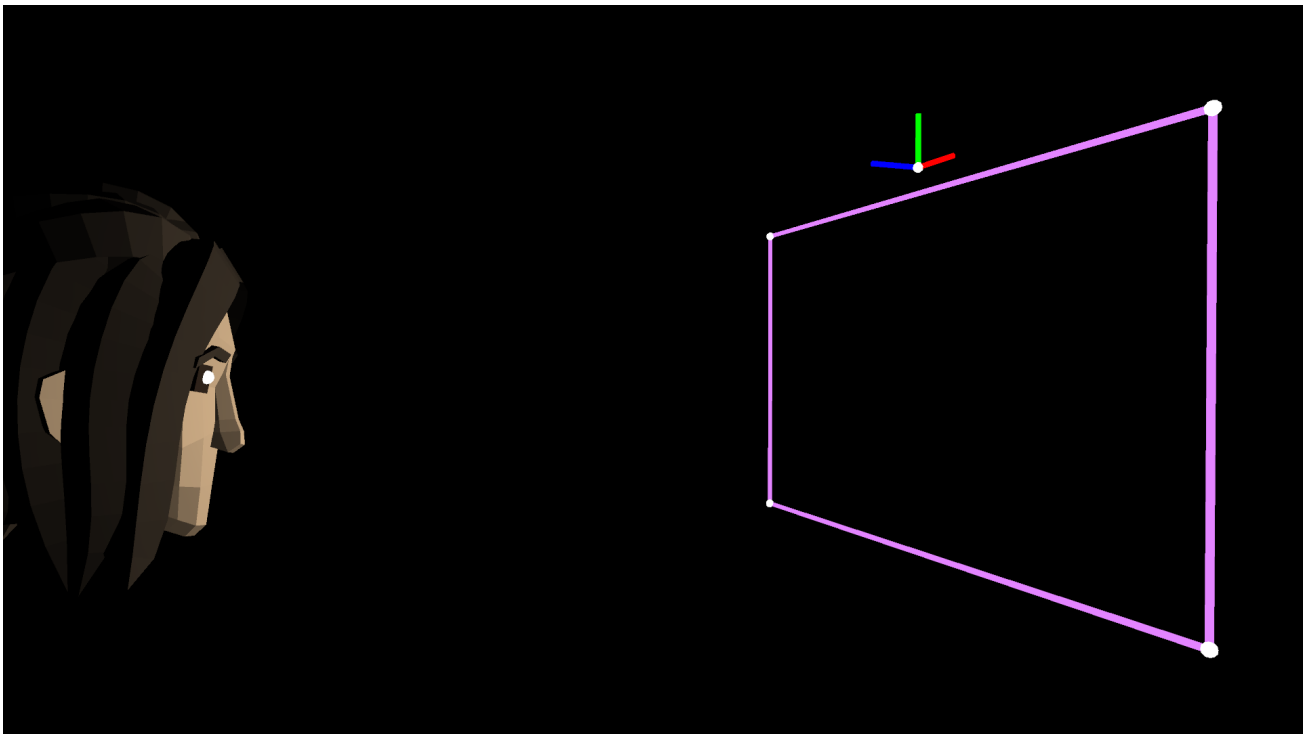
We want the screen to behave like a window: what it displays should change depending on the angle at which we view it, and on the distance between it and our eyes.

To make that happen we first need to recreate the real world in our game engine.

Since the positions of our eyes are calculated relative to the webcam, we can assume that the origin of our virtual world is the webcam.

If we do that, this is what the virtual world looks like overlaid on top of the real one:

And this is what it looks like in our game engine:



The pink rectangle defines the edges of the screen. It has the exact same dimensions as the real screen in centimeters. Even the distance between the origin of the world (the

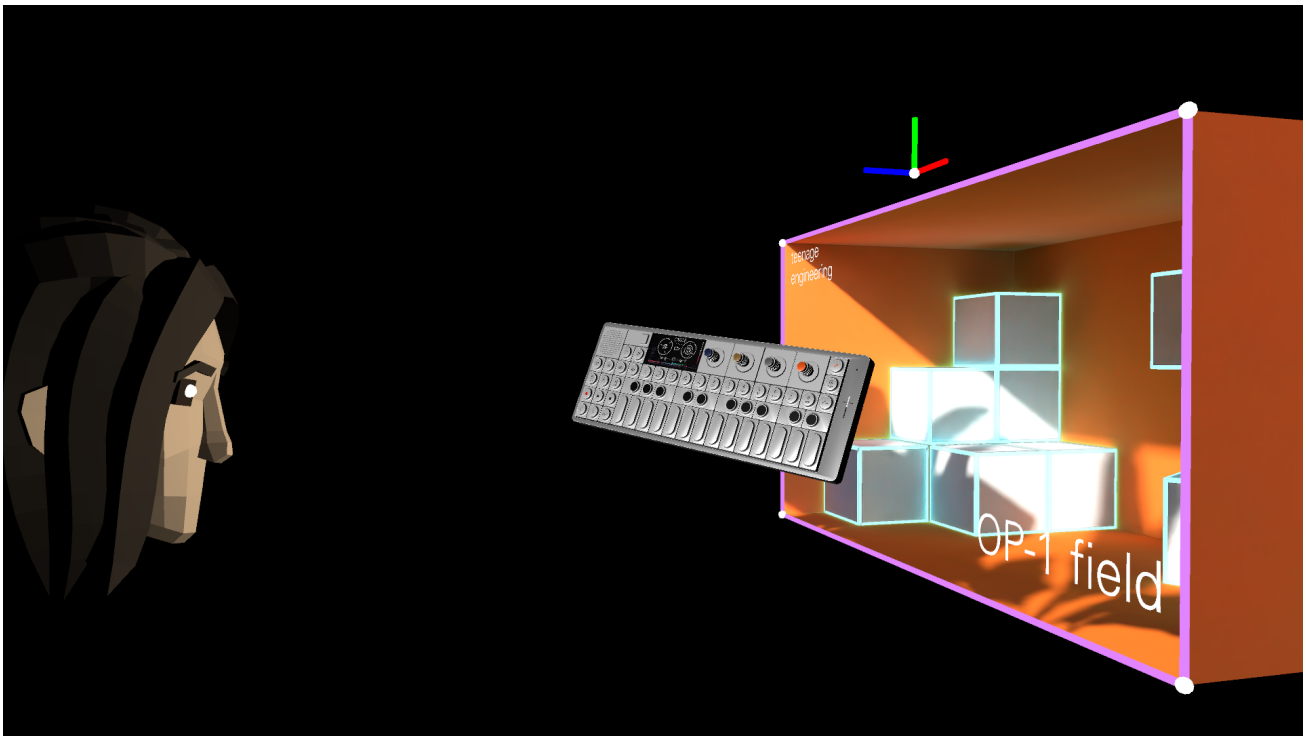
webcam) and the top edge of the screen is measured and accounted for.

You are probably wondering why it's necessary to use the dimensions of the real world. Think about this way:

- Let's say that MediaPipe calculates that your right eye is 20 centimeters to the right of the webcam, 10 centimeters above it, and 50 centimeters in front of it.
- In our game engine we can then place a virtual camera at the exact position of your right eye (later on we will explain why we use the right eye or the left eye but not the midpoint between the eyes).
- To have the screen behave like a window we need to pretend that it exists in the virtual world so that when the virtual camera looks at it, the angle and distance to it match those of the real world.

This is difficult to put into words, but hopefully it makes sense. We are going after an optical illusion, and we need great precision to achieve it.

Now anything we put behind the virtual window will appear to have depth, and anything we put in front of it will appear to pop out of the screen.

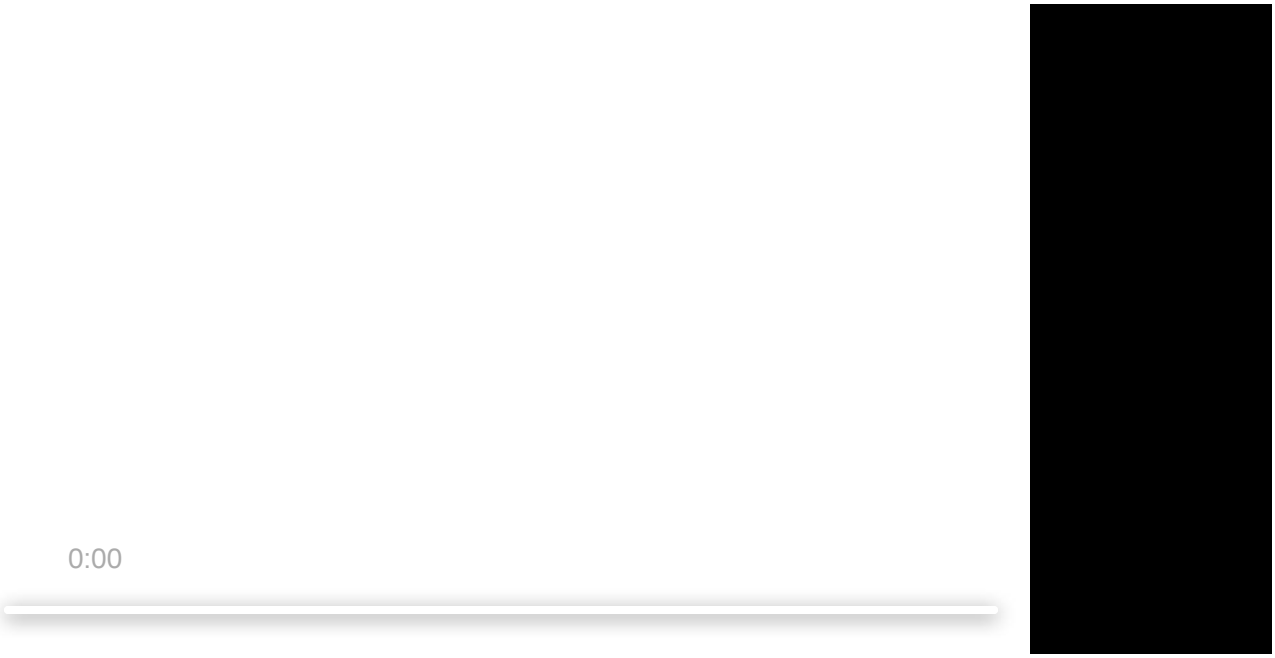


On-axis VS. off-axis perspective projection

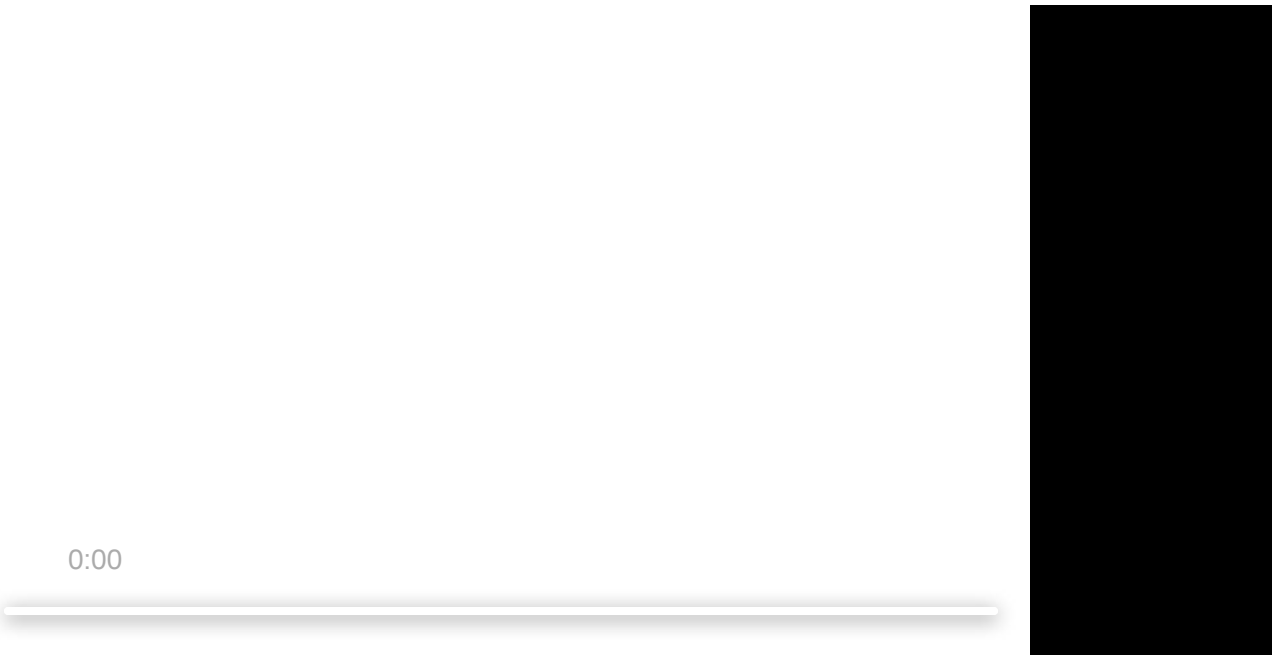
At this point we have recreated the real world in our game engine, and we are using eye tracking to place a virtual camera at the position of our right eye.

There is only one major gotcha left to talk about, and that is the difference between on-axis and off-axis perspective projections.

If our virtual camera used a normal perspective projection (also known as an on-axis perspective projection), this is what the view frustum would look like:

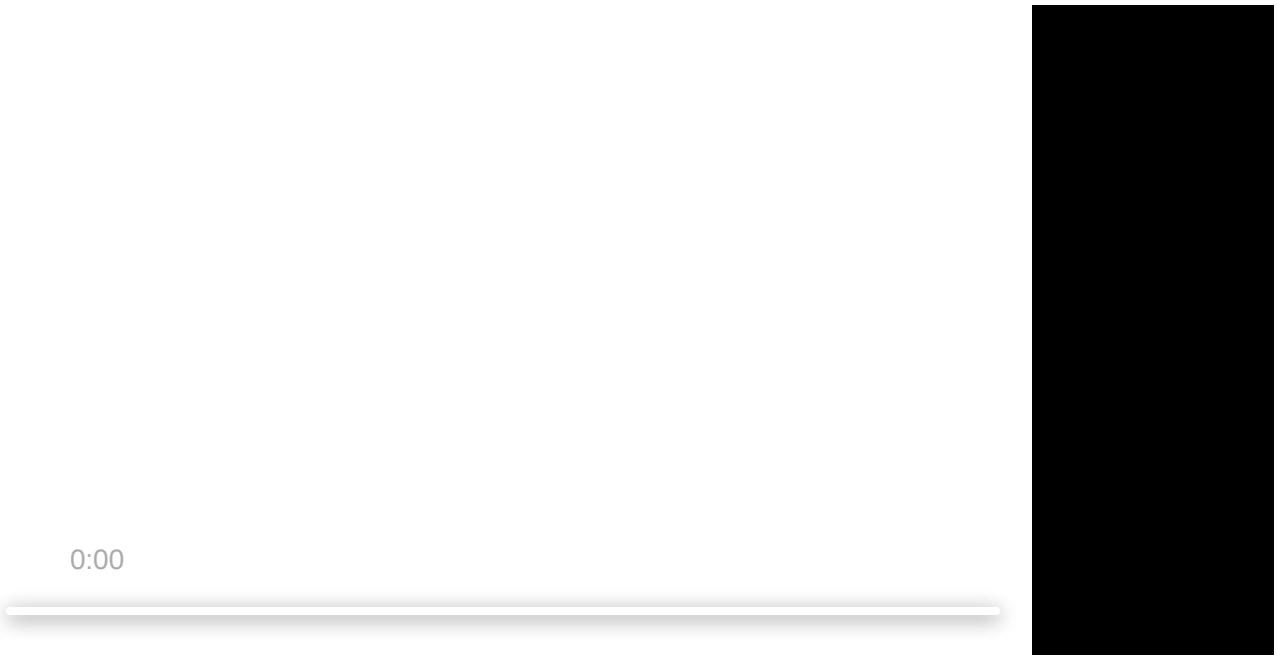


As you can see, it's a symmetric pyramid whose shape never changes. This is problematic because with this type of perspective projection, this is what we see on the computer screen when we move our head around:



As you can see, we are able to see beyond the edges of our window, and that's not what we want at all.

Now take a look at what our camera's view frustum would look like if we used an off-axis perspective projection matrix:



As you can see, it's an asymmetric pyramid whose base is glued to the virtual window. Now this is what we see on the computer screen when we move our head around:

This is precisely the effect we are after. What we see changes based on our viewing angle and distance, but we never see beyond the edges of the virtual window.

If you are interested in learning more about off-axis perspective projections, this paper explains them better than we ever could.

Why are you placing the virtual camera at the position of the right eye or the left eye? Why not place it at the midpoint between the eyes?

This is an important limitation of this technique that a lot of people don't know about: for the optical illusion to work, you must place the virtual camera at the position of one of your eyes and close the other one. Otherwise, things don't pop out of the screen or have the same feeling of depth.

To understand why that's the case, think about how VR headsets work:

- They render the scene twice, once for each eye.
- They present the separate renders to their corresponding eyes on separate screens.

In our case, we only have one screen, so we need to choose one eye and render things from its perspective.

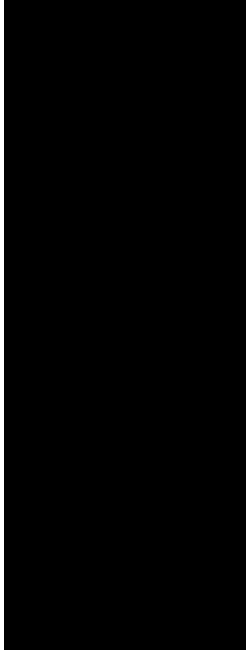
Placing the virtual camera at the midpoint between the eyes causes things to not look quite right for either eye, and this makes the optical illusion vanish.

Note, however, that it's still a really fun effect if you use the midpoint between the eyes and keep both eyes open. It just doesn't pop the same way.

Final prototype

In the end we achieved our goal of implementing Johnny Lee's technique in the browser using only a webcam.

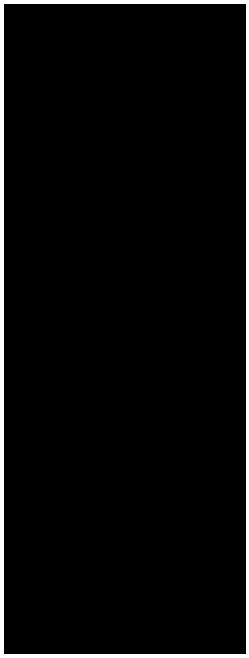
Here you can see our version of Johnny's famous targets demo:



0:00

To record that video we held an IPad with the camera pressed as closely as possible to our right eye. It's amazing how much the targets appear to pop out of the screen.

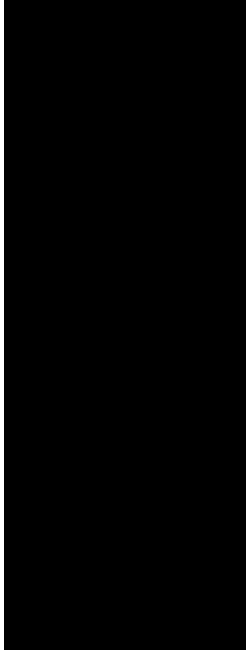
We also implemented a demo that's more focused on showcasing depth and camera movement. You can see it there:



0:00

We thought it would be funny to have the zombie try to break out of the screen and get angry when he couldn't. Imagine all the things that could be done with this technique in video games!

Finally, we implemented the virtual storefront demo that we proposed in the beginning of this post. You can see it here:



We decided to use Teenage Engineering's OP-1 field for this demo because it's such a cool product.

It's unbelievable how much it feels like the OP-1 field is floating out of the screen. It's almost as if you could reach out and grab it.

That feeling of "reach out and grab it" is the reason why we started referring to this project internally as "WonkaVision", as a homage to this famous scene from the original Willy Wonka & the Chocolate Factory movie in which Charlie reaches into a TV and pulls out a chocolate bar that has been sent through the air.

Conclusions

In the beginning of this post we asked ourselves:

> *Why did Johnny Lee's technique never take off? Why wasn't it implemented everywhere if everyone loved it?*

After jumping through all the technical hurdles required to make it work, we believe it never took off because:

- You need special hardware or the intrinsic parameters of your webcam, which are difficult to calculate correctly.
- You need the physical dimensions of your screen, which can't be queried even in modern web browsers.
- You need to close one eye to go from "fun effect" to "stunning optical illusion."

Sadly, Johnny's technique is simply not easily portable. We have it running in the browser as a website, but it only works with one particular screen and one particular webcam.

Unless it becomes possible to easily get the intrinsic parameters of any webcam, and the physical dimensions of any screen, this wonderful optical illusion will never go mainstream.

And even if all that becomes possible, will users be willing to close one eye to enjoy immersive experiences? That's a question we can't answer.

Despite our failure to ship this to a massive audience, we still think it's important to take a step back and marvel at what's currently possible in modern web browsers. Thanks to WebAssembly we can now load and run complex machine learning models by just clicking on a link. We can track eyes, faces, hands and so much more. We can even render beautiful scenes at smooth frame rates.

Maybe we can't bring the magic of XR to people who don't own headsets with Johnny's technique, but there are so many other avenues left to explore.

It's hard not to be excited about the future of the web right now.

February 28, 2023