So, I’m sure some of you will remember this game. It’s called Stack, where the object is to stack these thin blocks on top of each other, as high as you can go. If your block is just a little bit off, that part of it falls off. Some of you may even know that there was an AR, augmented reality, version of this game. The premise is the same, with the gimmick that the stack could now live in the real world with you. Cool way to waste some time, I guess. If you like games like that. But, have you ever wondered how this actually works? How does your phone know where the table is? How do the stacks warp and contort to make it look like it’s on your desk? Same thing the snapchat dancing hotdog filter. How does the app know how big or small to make Mr. Hotdog?

Well, it has to do with something called **Homography.** Basically, a homography is a matrix that transforms images, such that it conforms to a specific perspective. In this case, it tells the stack (or Mr. Hotdog) how to warp itself to look like it’s sitting on your desk.

Let’s take an easier example. Say I have two pictures of my coffee machine. One of them is looking at it a little to the left, the other a little to the right. How would I transform the first image to look like the second?

Well, I could open up GIMP and mess around with the perspective transform tool until I got the desired effect. That’s a more visual way to do it. But that isn’t how we’re going to do it.

In order to understand how homography works, we first need to understand what vectors are in the world of linear algebra.

Take a coordinate grid. I’m sure we’ve all seen these before. Vectors are a ray that always goes from the origin to a point. That point where the vector ends up is how we label the vector. So, for example, we have a vector that points to the location (2, 3). We label this vector with the x value on top—2—and the y value on the bottom—3-- surrounded by square brackets. That’s it! That’s what a vector is.

Now, this vector by itself isn’t all that interesting. What happens when we want it to point somewhere other than where it currently is? How would I, for instance, tell the software that I am using to animate this video how to move the vector to a new location?

Well, we use something called a **Linear Transformation**.

This is quite simple in practice. In order to get the new vector, all we need to do is multiply that vector by where i-hat and j-hat end up *after* a transformation. Consider the vector from earlier, vector [2, 3]. All this vector says is that it is 2 times i-hat, ***plus*** 3 times j-hat. Assuming i-hat is a vector [1, 0] and j-hat is a vector [0, 1], all we need to do is apply the transformation we’d like to apply, and then multiply the original vector, [2, 3], by where i-hat and j-hat ended up. Let’s just say i-hat and j-hat end up at [1, -1] and [2, -1] respectively. What we do now is similar to how we got the vector in the first place: we multiply 2 by the transformed i-hat, [1, -1], then add 3 times the transformed j-hat [2, -1]. That’s [2, -2] plus [6, -3], giving us a final vector of [8, -5]. What this means is that for any linear transformation, it can be entirely described using just the coordinates of where i-hat and j-hat land.

To sum up: we take the transformation matrix and multiply it by the original vector to find out where that vector is after the transformation.

Well, what about the opposite? So far we’ve only covered what happens to vectors before and after a transformation. But, what happens when we don’t *know* the transformation? In other words, what happens when we don’t know where i-hat and j-hat end up?

Well, that’s tricky to explain. For what is essentially a primer to linear algebra, finding a transformation matrix is well into the weeds. However, in the context of image manipulation, this is what **homography** aims to solve.

Let’s go back to our example from the beginning. I have those two images of my coffee machine, taken from different perspectives. If we take a common point between these images, say, the control knob, what we’re doing is setting up a starting vector, and a transformed vector. Now we’re stuck with the same issue as before. We have two vectors, but we don’t know the transformation matrix. Turns out, we can apply a special formula, shown on the screen, to generate the matrix needed to do the homography calculation. The reasoning behind why *this* matrix in particular is needed is long and complicated, but if you’re interested, let me know and I’ll point you in the right direction.

One of the requirements of homography is that we need at least four of these points.