So, I’m sure some of you will remember this. It’s this mobile game 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 down. Some of you may even know that there was an AR 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?

Let’s take an easier example. If any of you have ever watched any sporting event, you might have noticed that sometimes the logos will change. In Phillies games in particular, there’s a logo on the pitcher’s mound, and another logo to the right behind the batter. How is that possible?

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 logos how to warp themselves so they fit on these spaces.

One way to do this is that 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. One of the fundamental structures in linear algebra is called a vector. A vector is 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 (1, 2). We label this vector with the x value on top—1—and the y value on the bottom—2-- 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 else? 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 [1, 2]. All this vector says is that it is 1 times i-hat, ***plus*** 2 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, and then multiply the original vector, [1, 2], by where i-hat and j-hat ended up. Let’s just say i-hat and j-hat end up at [3, 0] and [2, 2] respectively. What we do now is similar to how we got the vector in the first place: we multiply 1 by the transformed i-hat, [3, 0], then add 2 times the transformed j-hat [2, 2]. That’s [3, 0] plus [4, 4], giving us a final vector of [7, 4]. What this means is that for any linear transformation, a vector can be entirely described using just the coordinates of where i-hat and j-hat land.

So this is great, but 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 a logo, say, the Temple logo, that I’d like to project onto the wall behind the home plate. One of the requirements of homography is that we need at least four corresponding points. That is to say, we need a series of points where the images are supposed to line up. In this case, it’s the four corners of the Temple logo, and the four corners of the logo behind the pitcher. What we’re doing is setting up starting and transformed vectors. Now we’re stuck with the same issue as before. We have eight vectors, but we don’t know the transformation matrix. Turns out, we can apply a special matrix **[show matrix A]**, 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. All it boils down to is, for every pair of corresponding coordinates, we plug them into this matrix **[show two-row A matrix subset]**. Repeat this step for every coordinate pair. Once we have this absolutely huge matrix, transpose it. Then multiply this transposed matrix by the original.

In order to continue, I need to explain just one more thing from linear algebra. Given a linear transformation, there are many vectors that are knocked off their span. That is to say, they’re rotated off of the line that runs through the original vector. However, there are some vectors, depending on the transformation, that are *not* knocked off their span. These are called **eigenvectors**, andthe **eigenvalue** is the value by which that vector is scaled.

For example, we have basis vectors i-hat and j-hat. [1, 0] and [0, 1]. Let’s just say we apply a transformation matrix [[3,0], [1,2]], saying that i-hat ended up at [3,0], and j-hat ended up at [1,2]. The first eigenvector of this transformation is actually just i-hat! You’ll see that when we apply the transformation, i-hat is not knocked off of its span, which just so happens to be the x-axis. The corresponding eigen*value* is 3, since this vector was scaled by three. Turns out, there’s another, sneakier eigenvector, [-1, 1]. This eigenvector has an eigenvalue of 2, since the original vector [-1, 1] was scaled by two.

Okay, so, back to the example. We have our matrix that we got earlier, now we need to find the eigenvalues and eigenvectors of this matrix. The reason for why we need to find these values is, again, beyond the scope of this video, but if you’re interested, let me know. All we need to do is find the smallest eigenvalue out of the set, and the corresponding eigenvector *is* our homography matrix!

Commence the celebration, we finally got there.

Just to complete the puzzle, when we apply the homography matrix to the Temple logo, it aligns *perfectly* with the advertisement box behind the batter.

This is not the only example of how to use homography. There are tons more. Things like taking panoramas, scanning documents using your phone, the list goes on. I’ll leave it to you to work out the process for each of these examples, but it is very similar. Simply find corresponding points, run them through the big matrix calculations, and find the smallest eigenvalue’s eigenvector.

Thank you all for joining me on this journey. Until next time.