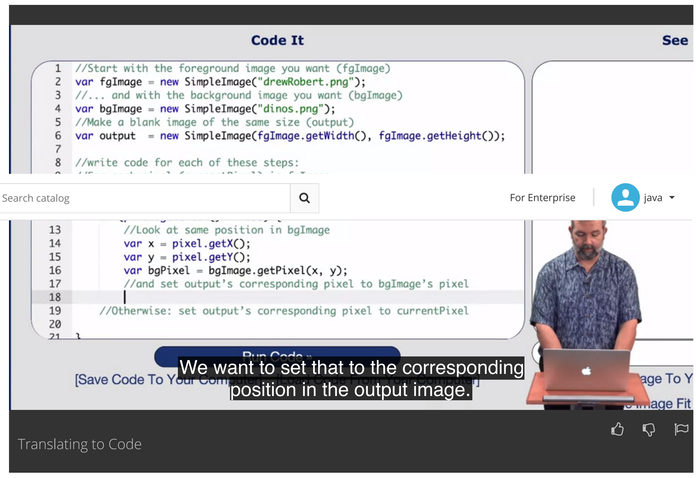

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:03
Okay, now it's time to turn our pseudocode into JavaScript code. For this video, we've already started by filling in some of the code, because it is so much like our previous examples and we figure that you've seen it before, so we don't need to go over it again. So that's creating the fgImage,the bgImage and the output image that is the same size as the fgImage.

0:29
So we're going to get down to actually writing the code which says to process each pixel in the fgImage. So we've written our standard For loop that goes through each pixel of fgImage.values. So now we're at the first line of pseudocode that is non-standard, that's specific to this problem that we haven't written before. And that is to look at the current pixel and determine if it's green.

0:59
It turns out that we didn't really figure out what green is in our planning, so we have to kind of figure that out now as we're coding. We intuitively understood it when we were working out our plan, but how do we explain it to a computer? How does a computer know what green is?

1:22
Right now we're not sure, so we're going to start very simply. And we're just going to say, if the green component of the current pixel.

1:36
Equals 255. That's an example that we've seen before, that we understand means that something has a lot of green in it. In fact, it means it has all the green in it. And so, that's a simple place to start. And we'll see how it works out as we go. Starting simply is often a good strategy. For, when you're writing code just to make sure that we have something that works.

2:01
With that done, let's start looking at what we're going to write when that condition is true. Then we're going to go inside the conditional and write that code. So, it says to look at the same position in the bgImage, and we recall that every pixel has a coordinate location within an image given by its x and y values. So we're going to go ahead and get those and store them in variables named x and y, reasonably.

2:38
So that is the corresponding location in the fgImage, and now we want to get the pixel

2:48
Of the bgImage at that same spot. So we'll call it getPixel with x and y to get that value.

3:01
So now those three lines represent that one line of our pseudocode. So we need it to do a little bit of work to translate that one line into actual code. Now that we have the corresponding pixel in the BG image We want to set that to the corresponding position in the output image. So we're going to do that by calling setPixel with that x and y. And the bgPixel that we just got.

3:45
And that completes the inside of that if statement.

3:52
Okay, let's look at our last line of pseudocode. It starts with the word otherwise, but what does otherwise mean? Otherwise means only do this last piece if the condition on line 12 was false. Well how, how do we write that code? Well, we could write the opposite or the negative of the condition on line 12 but that can be hard, and so that is error prone, that can lead to problems, so most languages, including JavaScript, include something called an else statement, a shorthand for this situation.

4:33
An else just means, if the condition above on line 12 was false, then run this code instead.

4:43
So what do we do when we say otherwise? We're going to set the output's corresponding pixel to the current pixel.

4:50
So we're going to say, output.setPixel(x, y, pixel) but in this case we want the current pixel so that's just pixel. Now we wrote x and y here

5:09
and those are things that we want to computer, just like we did before and I'm just going to show you a different way to do it, than how we did before. I'm going to write these values in line, whereas in the previous version,

5:27
I wrote them and stored them in a variable so, from a functionality perspective of the code, these work exactly the same. Whether I decide to save it into an instance variable, or whether I pass it directly into the function, doesn't matter from the programming's functionality perspective. So you should use the one that you're most comfortable with. Some people find it more readable to give every value a name, and some people find it more readable if the code is a little bit shorter and they can write them in line. It's entirely up to you what you want to do.

6:06
But now I'm going to reveal my pseudocode and see that I start out and I got for each pixel, that was my standard loop, I'm checking to see the current pixel is green. Then I'm getting the pixel at the corresponding x and y coordinates in the bgImage. I'm setting those to the corresponding position in the output image. If that wasn't the case, then I'm setting the pixel, the corresponding pixel in the output image to the foreground images, or the current pixel. So I feel pretty confident at this point that my pseudocode is correct or that my code corresponding to my pseudocode is correct. So I'm going to run the code.

6:53
It doesn't work. My goodness. I just double checked it and I have exactly what looks like the foreground image. This doesn't look like it applied anything at all from the background image. How could that possibly be? Well, if you recall, when we were starting this video, we sort of felt like our idea of what green was, was a little bit hazy. And this is a very exact measure for what green is. This is saying, the green has to be exactly the maximum value of green which is 255. So I'm going to try to have a more loose definition of what green is and I'm going to try, say, 240 as a value. So this now gives me a variety of green values to work with. And let's see what happens when I run this code. That looks a whole lot better. Now I can see dinosaurs, now I can see space. It seems like it worked a whole lot better to give us this range. However, if you look very closely at the image, you'll see that Drew and I have a halo around us. We have a green halo where we weren't quite correct in our range. So let's try, since that worked so well, let's try upping the range a little bit more and say making it 200. So that gives us even more

8:18
green values to work with to try to get that halo to go away.

8:22
And I ran the code again, and now you can see that the colors, the space colors, are bleeding into Drew's shirt. And so it looks like Drew is fading into Space, and that's not what we want either. Now we could spend some time trying to perfect this range, for this particular image so that we get the halo to go away but space doesn't intrude on the foreground image. But that doesn't seem like the best way to go about solving this problem, because we're really just going to make it work for these two images, for this particular range of colors. So let's think about a different way to define green that might be a little bit more general and robust. So instead, I'm going to define green as a color that has more green in it than both red and blue combined. So I'm going to write that as the value of green is greater than the value of the red and the blue added together, combined.

9:36
So now when I run the code I see that the halos run away and the space went back to its proper place, so I'm very happy with the results of this code that I've written. I've gotten my code to work for these two images, and I feel very good about it, but to be certain of my code, I would certainly want to try it on a wider variety of images, to see if I've really defined green correctly and gotten it to work. But I'm going to leave that as an exercise for you to encourage you to experiment on your own with your own images.

Downloads

Lecture Video mp4

Subtitles (English) WebVTT

Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?