Event-Driven Programming

Green Screen Web Page

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

Interactive Transcript

Search Transcript English

0:03
Hi. Let's look at changing a simple image programmatically using the same concepts you used in the previous module. In the foundational programming lessons, you modified pixels in an image. For example, you changed different parts of an image to create stripes, so that part of the image was red, part was blue, and more. You also saw how to modify all pixels of an image as part of creating a new image that was a green screen composite of two images. We'll use these same concepts of modifying pixels to create interactive web pages that use this simple image library.

0:40
Let's look at creating a grayscale version of an image. Grayscale is a common filter. We'll start with a code pen webpage that already has functionality to upload an image. We'll fork it, and we'll add a button to create the grayscale webpage that takes an image, like the waterfall on the left, and converts it into a gray scale version as shown on the right.

1:06
Extending a working web page continues what we did earlier with the prototype webpage before we introduced the HTML file input type.

1:16
The button that we create will call a JavaScript function named makeGray, using the standard on click event handler. We'll need to access the uploaded image in this function makeGray. This will require a new programming concept, global variables.

1:36
A global variable can be accessed in all functions, because it's defined outside of any of these, that's what makes it global. We'll see that a global variable for an uploaded image can be accessed in the function upload, and also in the function makeGray. The global variable set or assigned a value and upload and accessed or modified in makeGray to create the gray scale version of the image.

2:03
The first four steps in writing a program are to devise an algorithm. We know that what makes a color a shade of gray, is that each of the red, green, and blue values is equal to the others. And we'd like for our gray scale image to retain variations in brightness of the original image, so it won't just be black and white. One way to do this, is to average the RGB values and to set the new RGB values to this average.

2:32
Let's look at the first step. Working an example by hand with some sample pixels. To do any math on this green pixel, we need to know the RGB values.

2:44
For green that's 0 red, 255 green, and 0 blue.

2:51
Averaging these values means performing the operation (0 + 255 + 0), all divided by 3, which is 85. That means the new gray scale pixel will have R and G and B values each of 85.

3:11
Which looks like this.

3:14
Try this for magenta.

3:17
We can see the RGB values. Did you get 170, a gray that looks like this?

3:25
If you'd like more practice by hand, try the gray scale algorithm on maroon, sky blue, and orange with these RGB values.

3:35
Did you get these answers?

3:40
Now we've done steps one and two, working an example by hand. You wrote down what you did. We can now generalize these steps. Start with the image we want. And then for each pixel in the image, obtain the R and G and B values. Calculate the average value. And then set each of the RGB values to this average.

4:06
Finally, display the final image.

4:11
Using the Duke learn to program environment, you wrote code to process all pixels in an image to change their color. We'll use that same idea of looping over pixels to create a gray scale version of an image in our interactive web page. Let's look at translating this algorithm into code.

4:28
Recall that we're adding a button Make Grayscale, who's onclick event handler is the function makeGray.

4:38
Let's look closely at this function makeGray.

4:42
Here's the JavaScript source code for the makeGray function. We'll look at each line and see how the code works to convert an image to gray scale.

4:51
Let's assume that the first step of the algorithm is already complete. The variable image was previously defined.

4:58
To do steps two of the algorithm, we loop over all pixels in an image by creating a variable to represent each pixel, and then using the image method .values to access all the pixels. We calculate the average of the RGB values by accessing each of them, adding them together, and calculating the average. Then we set the pixel's red value to the average, then the green, and then the blue to the same average.

5:27
Finally, to perform the first step of the algorithm, to display the gray scale image, we access the HTML canvas element by calling document.getElementById with the id of the canvas. Here's the C A N or can. We use this canvas by passing it to the image's drawTo method so the image draws itself in the canvas and we see it on the webpage.

5:51
We do need to address the first step of the algorithm. Let's take a look.

5:57
We need to start with the image we want. Where is it? In our version of the webpage that only uploaded a file, changing the file selector by clicking on it, called the function upload,

6:10
where we created an image by calling new simple image.

6:15
But now, we need this image in the function makeGray. How does makeGray access a variable initialized in another function? Let's solve this problem.

6:27
Since upload is where the variable image is defined with the var, V-A-R keyword, That means a function like makeGray

6:36
cannot see variables defined inside the function upload.

6:40
And image.values will not work.

6:44
We need global variables for this so that many functions can access the same variable. We can still use the variable image in the upload function, but without the word var, the keyword V-A-R. A global variable can be accessed in all functions. Let's see how to use global variables in code.

7:06
A global variable is defined outside of all the functions, using the same JavaScript keyword V-A-R, var, to create the variable.

7:15
The global variable can be used in each function without using var. Remember that using var creates a new variable. Here, the variable image is assigned the value of calling new SimpleImage. It's a sign that inside the function upload.

7:32
Here the variable image is accessed in the function makeGray to loop over all pixels in the image. Note that the variable pixel defined with var but the variable image is not. And here, image is used again so that it can be displayed in the canvas by calling the image.drawTo method.

7:54
Be careful when using global variables. If you had defined the variable image using var inside the function upload, you wouldn't be assigned to the global variable, but to one defined for the first time in upload. This is a hard bug to find sometimes.

8:11
It's possible to have more than one global variable. For example, suppose we want to display the original image without the gray scape alteration, without uploading the image again. We could create a new image in function makeGray, rather than modifying the uploaded image. We could store both the regular image and the gray scale version in two different global variables. You'll have the opportunity to try out this alternative where you don't alter the original image.

8:42
Try to minimize the use of global variables. Relying on too many can make it hard to understand the code you're developing and how the functions interact with each other.

8:53
Happy coding.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt
Lecture Slides pdf

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