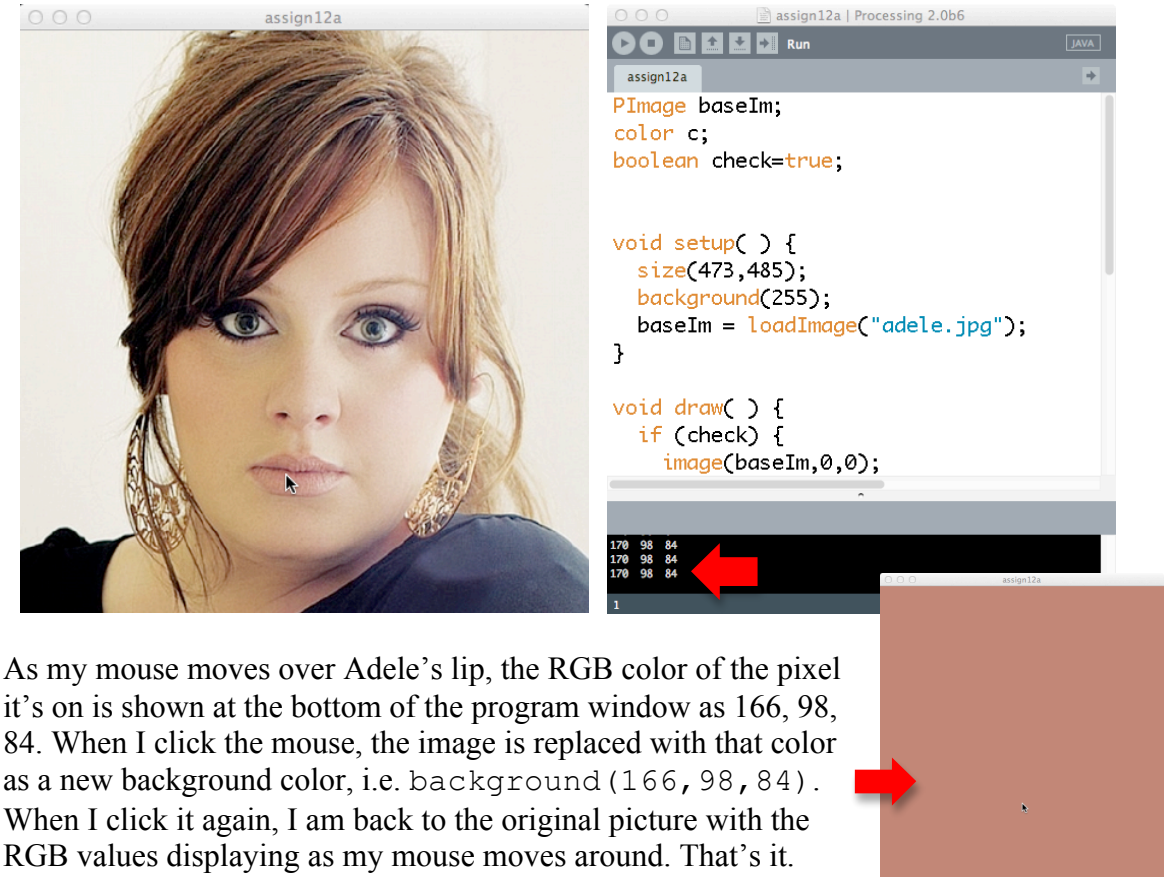
Homework 12: Color Checker

Goal: To understand the basics of processing images in Processing. This assignment teaches new material, so you will need to consult the Processing Reference for full understanding.

Overview

Our plan is to display a .jpg image, and then program the following tiny app, Color Checker: The user moves his or her mouse over the photo, and the RGB colors of the pixel that the mouse is on are displayed. For example, my photo and the running program are shown here:



As my mouse moves over Adele's lip, the RGB color of the pixel it's on is shown at the bottom of the program window as 166, 98, 84. When I click the mouse, the image is replaced with that color as a new background color, i.e. `background(166,98,84)`. When I click it again, I am back to the original picture with the RGB values displaying as my mouse moves around. That's it.

Displaying A Photo

Recall that when we were working with letters, we needed to go through a series of steps to actually use the letters. Images, like .jpg photos, work in a very similar way. Here are the steps to get a photo onto your canvas:

1) Put the photo in the same folder as the .pde file; suppose it's called `photo.jpg`
2) Find the width and height of the photo (Windows, at the bottom of the file list when the file is selected; Mac, in the Info display (use control+CLICK > Get Info).)
3) Declare an image variable of datatype PImage, as in `PImage baseIm;`

4) Define the `size( )` to be equal to the photo's width, height; this is generally not necessary, but it simplifies matters for us today.
5) Next load the photo into your program and give it a name by adding to `setup( )` the statement, `baseIm = loadImage("photo.jpg");`
6) Finally, in `draw( )` position the image on the canvas starting in the upper left corner using the statement `image(baseIm, 0, 0);`
7) Finally, follow the step (6) `image( )` call with `updatePixels( );` call to get the screen to change whenever you change it.

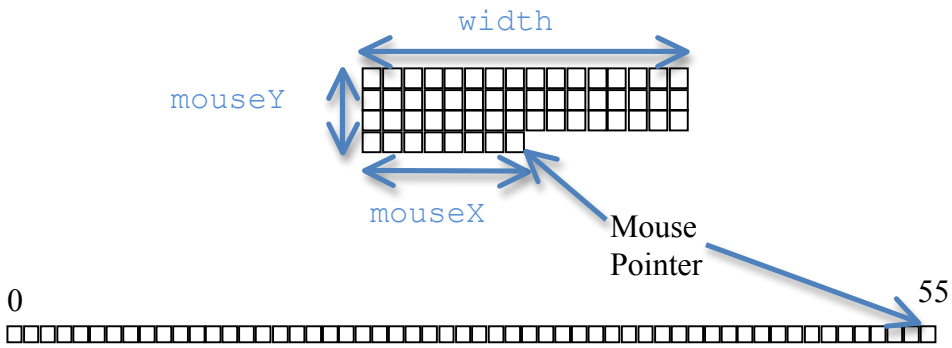
If you have any questions about these operations, check the Processing Reference. At this point running your program should display the image. Try it!

Linearizing A Picture

Next we need to grab the RGB values from the pixels. The first step is to ask Processing to fill an array, called `pixels[ ]`, with the image's pixels. You do this with the statement, `pixels( )`; After doing so, the `pixels[ ]` array is filled. The i^{th} pixel is referenced as `pixels[i]`, and of course, they start at 0.

The curious thing about the array of pixels is that it is one-dimensional despite the fact that the picture is two-dimensional. So, the pixel in the upper left corner is `pixels[0]` and the pixel in the lower right corner is `pixels[width*height-1]`. (Recall that width and height are the number you used in the `size( )` function in `setup( )`.

The fact that the pixels are stretched out in a line isn't much of a problem, but it does mean that we need to covert from 2-D to 1-D. For example, if `mouseX` and `mouseY` are the position of the mouse, then its pixel is found at `pixels[mouseY*width + mouseX]`. Why? If we think of the image as a 2-D array of pixels



each row contributes `width` number of pixels, and there are `mouseY` of them up to the row where the Mouse Pointer is. And, the mouse pointer is `mouseX` pixels over to the right. So, in the example, `width` is 16, `mouseY` is 3 and `mouseX` is 7, because we always start counting at 0, i.e. 0-origin counting, which makes the `pixels[3*16+7] == pixels[55]`. Make sense?

Extracting RGB

At the top of the program declare a variable `c` of datatype `color`, and in the `draw( )` function assign to it the pixel under the mouse pointer, as just explained. The RGB colors of this pixel can be extracted by the functions `red(c)`, `green(c)` and `blue(c)`. Again, check the Processing Reference for details. They actually have float datatype, but we can make them integers with the function `int( )`, as in `int(red(c))`. So, after assigning the `c` variable the pixel pointed at by the mouse, write it to the console with

```
println(int(red(c)+"    "+int(green(c)+"    "+int(blue(c));
```

Here, the color values are converted to a string and spaces are positioned in between using the concatenate operation (+). You should now be able to move the mouse around and see the pixels printed at the bottom of your working window. Try it!

Checking Color

Now, to finish the app. You will need a Boolean variable that starts out `true`. Let's call it `check`. Your entire revised `draw( )` program will be an if-statement testing this Boolean variable. If it is true, then your app does pretty much what we've done so far: `image( )`, `loadPixes( )`, `updatePixels( )`, assign, `println( )`. Otherwise it does nothing. And, in addition, you need a `mousePressed( )` function. It specifies a new background using the extracted colors (float version), and flips the truth of the Boolean variable.

That's all there is to it. Try it out!

Wrap Up You have been introduced to the techniques of displaying .jpg images (.gif and .png work the same way, but with different file extensions, of course). You have also found out how to load pixels into the working array `pixels[ ]`, and extract the colors from each pixel.

Turn In Turn in your commented program – your grade will include a component for the quality of your comments – and submit it to the dropbox.