**CSC 545/645 Computer Speech, Music and Images**

**Exercise No. 6, Week 9, Due March 14, 2021**

**Accessing the Pixel Array**

**Goal**

Learn to use the image pixel array, rather than the get() and set() methods

**Background**

An image is a two dimensional object comprised of pixels arranged in rows and columns. It’s natural to think of accessing pixels using x and y coordinates in a two-dimensional Euclidean plane. However, if you think about it, you’re probably aware that random access memory is linear; each memory location has a unique address—one number (not two coordinates) starting at 0 and increasing until the highest-addressed location installed in the computer. It makes sense, then, that images are physically stored in an array, rather than a matrix. In Processing, you can operate directly on the pixel array, rather than using the get() and set() methods—and there are times when it makes sense to handle an image as a one-dimensional array of pixels, rather than as a matrix.

**Procedure**

Complete the exercise program to show the distance between the color of each pixel and the color of an average pixel. Use the image pixel array to calculate the average red, green, and blue values over the entire image and make a new color from those averages. Note that, while you are calculating the averages, the values will exceed 255—so you will have to use floats to accumulate the red, green, and blue values. Divide the accumulated values by the total number of pixels (the length of the pixel array) and create a new color from the average red, green, and blue.

Write a function (cdist()) to return the Euclidean distance between any two colors. Write another function (avgDist()) to accept an image and its average color; it will use the image pixel array to calculate the distance between each pixel and the average color. It will return an image that has, in each pixel position, a grayscale value representing the distance between that pixel in the original image and the average color. Scale the distance value to the range 0 to 255. For the distance range you can use the maximum distance in the image or the distance between the colors white and black.

Also allow the user to select part of the image for the average pixel value; this is already written in the mouseReleased() event handler—but it calls your avgDist() function. Be sure you understand how this code uses x and y to calculate the position of a pixel in the one-dimensional array. The resulting image will be grayscale values indicating the distance from each pixel to the average of the selection. Try selecting a dark part of the image then a bright part of the image—how does the distance image change? By the way, note also how this function allows the user to drag the mouse in any direction when making a selection.

You could try some other point processing operations using the pixel array—brighten or darken the image, or stretch the histogram. Because point operations don’t depend on any other pixels, they can be easily done using the pixel array.

**Why would you use the pixel array?**

The Processing documentation claims execution is faster when using the pixel array rather than get() and set(). In most applications, the difference is not noticeable. But there are algorithms that are easier to implement using the one-dimensional pixel array, rather than nested loops with get() and set(). For example, both the k-means and median cut color quantization algorithms are easier using pixel arrays rather than two dimensional coordinates.

**New Processing commands**

PImage.loadPixels() //Provides access to the pixel array: img.pixels[]. It need be done only once for each image; you could do it in setup for any images that are available there.

PImage.updatePixels() //Updates the pixel array for the image; this must be done each time the pixel array is modified for the changes to take effect. For example, if you modify the pixels in a function, update the pixel array before exiting.