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

**Exercise No. 3, Week 7, due Sunday, Feb 28, 2021**

**Pixel Gradients**

**Goal**

Learn to difference pixels in an image

**Background**

So far we have been talking about *point processing*—the new value of each pixel depends only on the current value of the pixel. In *neighbourhood processing*, or *region processing*, pixels are assigned new values based on one or more surrounding pixels. This exercise is a simple example of neighbourhood processing.

**Procedure**

Write a Processing program to replace each pixel by the difference between it and its immediate vertical or horizontal neighbour; this is sometimes called a *gradient*. Carry out the following operations on the canvas – this will enable you to try multiple differencing (or, later, averaging) operations to see the cumulative effect when you carry it out multiple times. Be sure the image is displayed, though, when you want to start the operation for the first time. The truth is that it rarely makes sense to difference pixels multiple times but it often makes sense to average multiple times, which you will implement later.

Write a function vDiff() to difference pixels vertically. Start with row 0 and go to row height-2, replacing each pixel with a grayscale value that represents the difference between it and the pixel directly below it. The bottom row (row height-1) won’t be modified. Difference the red, green, and blue channels separately (use absolute values) and add those differences to get the new value of the target pixel. Remember that the total difference must not exceed 255—use constrain() (or an if statement) to clip the difference at 255. Alternatively, you could divide the total difference by 3 to keep it in range.

Write a similar function hDiff(), replacing each pixel with the difference between it and the pixel to its immediate right. Start with column 0 and go to column width-2. Set the hot keys as specified in the skeleton code.

What can you say about the different effects you get from the vertical gradient and the horizontal gradient? The difference can be most clearly seen in an image that has strong vertical and horizontal lines. If you want to preserve the edge colors, rather than make the edges grayscale, you could create new RGB colors from the channel differences and set the pixels to those colors.

Now try adding two functions: hAvg() and vAvg(). They will be just like the differencing functions but they will average each pixel with the one to the right, for hAvg, or the one below, for vAvg (average RGB separately and create a new color). I don’t know what keys to use – maybe capital ‘H’ and ‘V’? What effect do you get with averaging? See what happens if you repeat the averaging some number of times.

If you’re convinced your program is working correctly, try some other images. Load them into the data folder. To keep it simple, you can replace the file names in the fnames array; alternatively, you can add to the fnames array and create new key handlers for the new images.

Handling borders in neighbourhood processing always involves choices. One option is to leave a border of unprocessed pixels, as you did when not modifying the bottom row and rightmost column. Modify your program to modify the bottom row and rightmost and column but make the assumption that the image "wraps around" itself. In other words, x[width] uses x[0] and y[height] uses y[0] - so the last row is differenced with the first row and the last column is differenced with the first column. This is easier than you might think at first (think about how modulo works).

**Deliverables**

Submit your .pde file on Blackboard by the due date