Matt Prost

Dr. Calvin Lin

CS 314H

2 September 2015

Image Manipulation

In this assignment, we are challenged to manipulate images with a series of different effects. In addition to the list of required effects, I plan to create some supplemental effects. I am going to implement several of the neighborhood filter ideas listed in the Good Karma section, as well as some ideas that I have involving grayscale.

The understanding several of several concepts was the key to success in this assignment. The first and foremost piece of knowledge is that, assuming that the image is rectangular, images are stored as a matrix of pixels with rgb values. This means that each image is a two dimensional array of ints that store values between 0 and 255 corresponding to the amount of red, green, and blue in a particular section of the image. Many of the assigned effects are dependent on manipulating these rgb values or the matrix itself. For instance, the NoRed effect would require you to manipulate the rgb values of each pixel and set the red value to 0. Other effects however, affect the structure of the matrix more than the pixels themselves. Effects like VerticalReflect require the displacement of pixels within the matrix rather than the manipulation of those pixels' rgb values.

Another aspect of the images is their luminosity, or brightness. The luminosity can be found by adding the total of the red, green, and blue values in a pixel. If all three of the values are 255, then the pixel is the brightest it can be and is displayed as white; if all three of the values are 0, then the pixel is as dark as possible and is displayed as black. In effects like BlackAndWhite and Erode, we have to find the luminosity of pixels in order to determine if they are light or dark.

Finally, many of the effects that I plan to implement involve grayscale. A grayscale image has the same luminosity as a color image, but the pixels are various shades of gray. In order to achieve this, the red, green, and blue values must be equal to one another in each individual pixel. Therefore, one way to make a pixel gray, is to set the red, green, and blue values to the average rgb value in that pixel.

At its final stage, my code implement 24 different effects, including Invert and the 12 that were required in just over 1000 lines of code (including comments.) The extra effects that I implemented were Smooth, Erode, Dilate, GrayScale, Transparency, GrayWithRed, GrayWithNoRed, GrayWithGreen, GrayWithNoGreen, GrayWithBlue, and GrayWithNoBlue.

The Smooth effect turns a pixel into the average rgb value of the pixels in a 3 x 3 neighborhood. I used a second matrix to store the final smooth values, so the original rgb values would not be polluted. I also accounted for the fact that a pixel might not have an entire 3 x 3 neighborhood by checking to see if a pixel existed in the bounds of the matrix and counting the number of pixels in the neighborhood.

The Erode and Dilate effects found the luminosity of the pixels in a 3 x 3 neighborhood and replacing the current pixel with the darkest or lightest, respectively, pixel in the neighborhood. I accounted for similar issues in these effects as in Smooth, utilizing a second matrix and checking to see if a pixel existed within the matrix.

The Transparency effect used a parameter to determine the percentage of transparency to be applied to an image. This increased each pixels' rgb values by the percentage of the difference between those rgb values and 255.

In my GrayScale effect, I averaged the rgb values in each individual pixel in order to remove the color from the pixels but preserve their luminosity.

I combined the concepts behind GrayScale and the NoRed/Green/Blue and Red/Green/BlueOnly effects in my final classes. I used these effects to make some artsy images with the rose picture. I found the dominant color in each pixel, and depending on the effect, I might change that pixel to grayscale or keep it colorful.

In order to test my code, I applied the various effects to multiple images. The key to effectively testing one's code is making it as obvious as possible that the code is operating as intended. Many of the effects like VerticalReflect and Grow, manipulated the matrix of the image without altering the individual rgb values in the pixels. I tested these on some of the provided images like bacon and Dr. Lin's face because it was easy to tell how the images were being altered. Other effects like Dilate and Erode focused on the brightness of certain pixels. I mostly tested these on Dr. Lin's face because it provided contrasting regions of brightness, and it was mildly entertaining.

Most of my tests, however, involved effects that manipulated the colors within the pixels. Many of these would specifically target red, green, or blue, like the NoRed effect or the GrayWithGreen effect. In order to make the results of these effects the most apparent, I used two images from the internet. The first image was a rose. This image showed a red flower with a green stem against a blue sky. Because each of these three colors was so predominant in the image, it was easy to see the effects of the color manipulation effects. The second image was a color spectrum chart. This was supplementary to the rose image, showing off the multiple colors, but it also had the colors organized, so it was even easier to test the results of effects like Threshold.

There are many things to take away from this project but one very important on is the value of transparency in testing. During testing, one of my effects was actually incorrect because of a typing error in just one line of the code. I tested the effect, but it was not apparent with the image that I tested if it was functioning or not. I did not realize until after I tested it on pictures from the internet that there was one huge but easily fixable error in my code.

I also learned how valuable commenting code is. It's not something that I really did in high school, and I avoided it during the project. The comments for many of my effects were added long after I had already finished those effects. However, the comments can be a good way to plan out the code to come and to check that you are implementing things correctly. In my Erode and Dilate effects, I was comparing the pixel value when I should have been comparing the luminosity. This oversight actually led to comparing the amount of red in a pixel rather than its brightness. The effects appeared to operate as expected with the bug, but after writing the comments, I realized my mistake. By using comments, we reduce the errors in our code, and we have a plan for how to implement things.