COSC 101: Introduction to Computing I
Image Processing
Fall 2014

This document is a master document for all image manipulation stuff.

1 An overview of digital images

Digital images are rectangular grids of color. The dimensions of digital images are measured in units of pixels. The term pixel is short for **pic**ture **el**ements. Pixels are essentially tiny areas that are colored in some way. The composition of the grid of pixels, and the various colors of those pixels, is what causes a digital image to have a specific appearance.

Each pixel in the image is located using a simple two-dimensional coordinate system: the origin of the coordinate system, $(0, 0)$, is in the upper-left corner of the image (which is somewhat different than you might have expected) and numbers increase right and down. If the width of an image is `w` and its height is `h`, the lower-right coordinate is `(w-1,h-1)`. Figure 1 depicts the image coordinate system graphically.

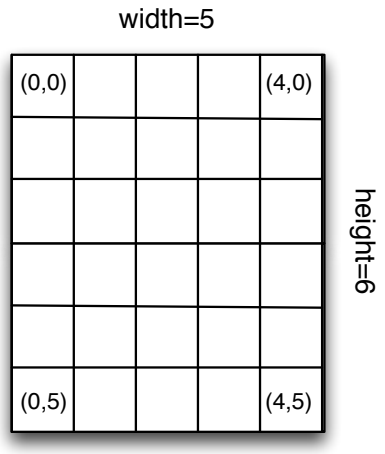

Figure 1: Example showing how the image coordinate system works. The coordinate of each corner pixel is shown along with the width and height of the image.

The most common way to specify color on the computer is with red, green, and blue (RGB) values (the additive color model used by light). An RGB value sets the amount of red, green and blue light in a single pixel in the screen. The intensity of each color component is what gives a pixel its overall color. Color intensities typically range from 0–255 with 0 the minimum and 255 the maximum. Examples:

- a black pixel has (R, G, B) = $(0, 0, 0)$ (no intensity for any color, no light is emitted equivalent to the screen off)

- a white pixel has (R, G, B) = (255, 255, 255) (full intensities in each component mix to white light).
- an entirely green pixel has (R, G, B) = (0, 255, 0).
- a pixel with (R, G, B) = (255, 0, 255) has maximum intensity red *and* maximum blue and has a magenta color.
- shades of gray between black and white can be obtained by setting all three colors to the same number for some number between 0 and 255.

With this fairly simple system, one can create a variety of 2^{24} different colors (which is a ridiculously large number of colors!) If you're interested, you can read more about pixels at `http://en.wikipedia.org/wiki/Pixel`.

2 Programming with images

We provide an `image` module that facilitates the process of manipulating images. The `image` module provides functions for creating and saving `Image` objects. The `image` module is distinct from an `Image` object, much like the `turtle` module is distinct from a `Turtle` object.

2.1 Creating, saving, and displaying images

There are three ways to make an Image object.

1. Load an image from a file: `img = image.load_from_file('crayons.png')`. You can open any file in PNG format. If you have a favorite JPG image you want to work with, a google search for "convert jpg to png" will yield links to a number of free online conversion tools. You may also be able to use a program on your computer. (In Apple's Preview program, you can open a JPG file and do "Save As..." to save into a different format.)
2. Make a new image: `img = image.new_image(400, 300)` will make an all black image that is 400 pixels wide by 300 pixels high. You can use the methods on Image objects to change the pixel colors, as explained below.
3. Make a copy of an existing image: suppose `img` is an Image object, then `img_copy = image.copy_image(img)` will create an identical copy of the given Image object.

You can save an Image object named `img` by doing this: `image.save_to_file(img, 'my_img.png')`. The image will be saved to a file named `my_img.png`. The file will be placed in the same folder as the program that you are running.

If you have several images, such as `img`, `img2`, and `img3`, you can display them in a pop-up window by doing this: `image.display_images(img, img2, img3)`. The `display_images` function takes one or more image objects (separated by commas) and displays them. *This statement should be the last statement in your program* because it will halt your program until the user clicks the "Quit" button on the window.

2.2 Working with Image objects

Once you have an Image object, you can call methods on it. First, let's make an an Image object:

```
img = image.load_from_file("crayons.png")
```

Methods `width()` and `height()` return the width and height respectively.

To find out how much red is in an individual pixel, you can use the `get_red` method. For example,

```
r = img.get_red(100, 200)
```

will assign to `r` the amount of red in the pixel at (x, y) = (100, 200). The value will be a number between 0 and 255. Similar methods exist for green and blue.

To change the value of red in an individual pixel, use the `set_red` method. This method takes three arguments. For example,

```
img.set_red(100, 200, 0)
```

will remove all red from the pixel at (x, y) = (100, 200). The third argument to the function must be a number between 0 and 255. Similar methods exist for green and blue.

Sometimes it is convenient to get all three colors at once. For example,

```
r, g, b = img.get_rgb(55, 40)
```

The `get_rgb` method returns *three* numbers – the red, green and blue values – which are in turn assigned to the variables `r`, `g`, `b`. This is special kind of assignment statement called **tuple assignment**. We will learn about this in more detail later in the course. For now, all you need to know is that for this method, you can put three comma-separated variables on the left hand side of the assignment operator. For example,

```
rutabaga, garlic, bok_choy = img.get_rgb(55, 40)
```

will assign the red, green, and blue values to the variables `rutabaga`, `garlic`, `bok_choy` respectively.

Similarly, it is convenient to set all three colors at once. You can use the `set_rgb` method which takes *five* arguments:

```
img.set_rgb(55, 40, 255, 255, 0)
```

the above statement sets the pixel at (x, y) = (55, 40) to be yellow – i.e., the color you get with maximum red and green but no blue.

2.3 Example program #1: red filter

This example program takes the image of the crayons and applies a red filter to it, meaning that all green and blue is removed from the picture and the only light that remains is the light coming from the red channel. As you might expect, crayons that are dark green or dark blue will appear to look almost black in the filtered image.

The program starts by making a copy of the image. This is so that we maintain a copy of the original image so in the end we can display "before" and "after" images side by side. The nested loops iterate over every possible (x, y) coordinate. For each coordinate, we obtain the pixel and remove green and blue by setting them to zero.

```
import image

img = image.load_from_file('crayons.png')
red_only_img = image.copy_image(img)

w = img.width()
h = img.height()

# loop over every (x,y) pair
for x in range(w):
    for y in range(h):
        # filter out green and blue
        red_only_img.set_green(x, y, 0)
        red_only_img.set_blue(x, y, 0)

# save the new image to a file
image.save_to_file(red_only_img, 'red_crayons.png')
```

```
# create window that displays both the original and modified images
image.display_images(img, red_only_img)
```

When run, the program causes this window to be displayed:

2.4 Example program #2: copy top

This example is more complicated than the previous example. In this example, we make a new image of the crayons in which the top half is duplicated and copied over the bottom half. The result looks like this:

Here is the program. Notice the **for** loop for the `y` coordinate only loops over half of the y values. Also notice that `px_bottom` is a pixel from the bottom half of `img_copy` since it is located at coordinate `(x, y + h/2)`.

```
import image

img = image.load_from_file('crayons.png')
img_copy = image.copy_image(img)

w = img.width()
h = img.height()
```

```
# loop over every x value...
for x in range(w):
    # ... but only loop over y values in top half
    for y in range(h/2):
        # get rgb from top half of original image
        r, g, b = img.get_rgb(x, y)

        # copy over rgb from top half to bottom half
        img_copy.set_rgb(x, y + h/2, r, g, b)


# save the new image to a file
image.save_to_file(img_copy, 'copy_top.png')

# create window that displays both the original and modified images
image.display_images(img, img_copy)
```