Project 5: Pixel Magic

CPE 101: Fundamentals of Computer Science
Winter 2019 - Cal Poly, San Luis Obispo

Purpose

To implement a program that accepts command-line arguments as well as tie together core concepts learned throughout the quarter.

Description

For this project, you will write functions that transform an image in the following ways:

1. Adjusting colors to reveal a hidden picture
2. Darkening colors to produce a fading effect
3. Averaging neighboring colors to blur an image

Download the image files from Polylearn.

Image File Format

The P3 (.ppm) format is a text-based format (meaning that it is readable text) that defines an image as a sequence of pixels beginning with the top-left pixel and stored in row order (i.e. every pixel in a row is stored in left-to-right order and before any pixel in the next row).

A file conforming to the P3 format begins with header information. The header consists of the characters `P3`, the integer width of the image (in pixels), the integer height of the image, and the maximum value for a color component (we will use `255` for this value). Immediately following the header is the color information for each pixel. A pixel's color is represented by three integers denoting the red, green, and blue components (in that order). The following example shows how a file would look if it contained an image initially with one blue pixel, one red pixel, and one green pixel.

```
P3
3 1
255
0 0 255
255 0 0
0 255 0
...
```

The top-left pixel is said to be at location `<0, 0>` and the bottom-right at location at

`<width - 1, height - 1>`.

Testing

You are required to write at least 3 tests for each function that returns a value (i.e. is not an I/O function). Since we are emphasizing test-driven development, you should write tests for each function first. In doing so, you will have a better understanding as to what the functions take as input and produce as output, which makes writing the function definitions easier.

Implementation

In the functions below, pixels refers to a list of lists of integers, in which each inner list (representing a single pixel) contains 3 integers, one for each color value.

`main()`

`find_image(pixels)`

Return decoded pixels.

`fade_image(pixels, width, row, col, radius)`

Return faded pixels.

`blur_image(pixels, width, reach)`

Return blurred pixels.

Error and Usage Messages

Regardless of the filter mode, print one of the following error messages when its corresponding condition is true.

- If either the filter mode or image path is not provided, your program must print:

  ```
  Usage: python pixelmagic.py <mode> <image>
  ```

- If the image argument is provided but cannot be opened (e.g. does not exist), your program must print (replacing the blank with the name of the file given):

  ```
  Error: Unable to Open ____
  ```

- If the mode argument is not decode, fade, or blur, your program must print:

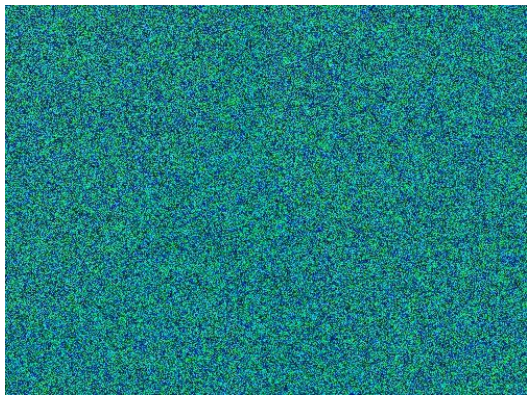
  ```
  Error: Invalid Mode
  ```

For all other messages to display, see the corresponding sections below.

Hidden Image

This program will get you started with the core functionality of processing a P3 image file. Your program must take a single command-line argument that specifies the name of the input image file. It will output the decoded image to a file named `decoded.ppm`. The output must be a valid P3 image file; do not forget to write the required header information to it.

The puzzle image hides a real image behind a mess of random pixels. In reality, the image is hidden in the red components of the pixels. Decode the image by increasing the value of the red component by multiplying it by `10` without allowing the resulting value to pass the maximum value of `255`. In addition, set the green and blue components equal to the new red value. Shown below is the hidden image; it will be obvious once you have properly decoded the puzzle image.



Fade

This program is a relatively minor modification of the previous program. In addition to the filter mode and file name, it must take three additional command-line integer arguments:

1. The row (y-coordinate) position of the fade center
2. The column (x-coordinate) position of the fade center
3. The fade radius

These three arguments may be assumed to be integers. If any these arguments are not provided, an appropriate usage message must be printed to the terminal and the program should terminate. This message is as follows:

```
Usage: python pixelmagic.py fade <image> <row> <col> <radius>
```

Your program will output the faded image to a file named `faded.ppm`, which must be a valid P3 image file; do not forget to write the required header information to it.

This program will transform pixel values based on their distance from a specified point (this point may fall outside of the image). The row and column coordinates specified on the command-line give the point and the radius is used to control the fading.

For each pixel, compute the distance from the pixel location to the specified point. Scale (multiply) the color components of the pixel by:

```
(radius - distance) / radius
```

Do not use a scale value below `0.2`, which prevents very dark borders around the image).

	
original image	row = 230, col = 255, radius = 255
	
original image	row = 160, col = 200, radius = 250

Blur

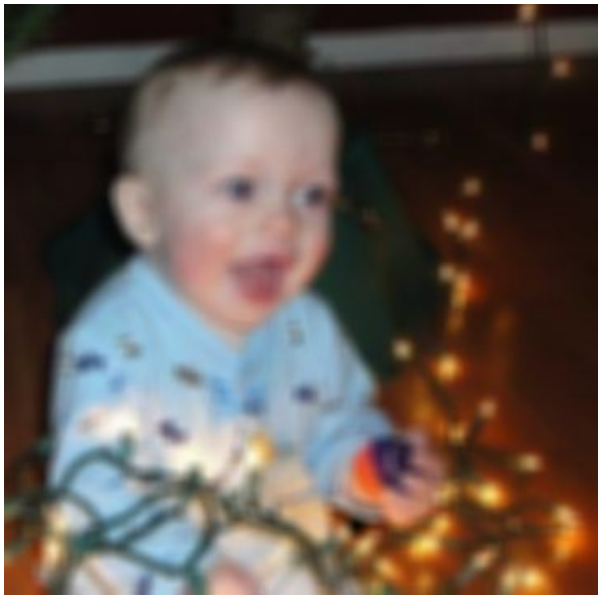
Your program must take two command-line arguments (in the following order):

1. The name of the input image file
2. The "neighbor reach" to use in the blurring calculation (this argument is optional and will be used to determine which neighbor pixels to consider in the averaging calculation, as discussed below)

If a second command-line argument is not present (i.e. the blur factor is not specified), then use a default value of `4`. Your program will output the blurred image to a file named blurred.ppm. The output must be a valid P3 ppm image file; do not forget to write the required header information to it.

Your program will blur the image by computing, for each pixel, the average of nearby pixels (more precisely, the averages over each color component of the nearby pixels). A pixel's blurred color will be determined by the colors of the pixels within a specified "neighbor reach". The default "neighbor reach" is `4` (but this can be modified by a command-line argument as discussed previously). Thus, a pixel's blurred color will be determined by the colors of the pixels within four pixels to the left or right and within four pixels above or below (this will form a square around the pixel). The following diagram is meant to help illustrate.

This diagram shows how to compute the color for the pixel in the center (the red element). Its neighbors (within a reach of 4) are all of the green elements. The pixels outside of this 9x9 square are not considered in the blurring calculation for this pixel. To compute the color for the center pixel, average the red, green, and blue components (independently) of every pixel in the 9x9 square (including the pixel itself; the red element in this diagram). Some pixels will not have the full complement of neighbors (such as those on or near the edge of the image). In these cases, just average the existing neighbors (i.e. those within the bounds of the image).

original image	reach = 4
	
original image	reach = 4

Submission

Zip `pixelmagic.py` and `tests.py` into one file and submit it to polylearn.