Image Editor Project

Introduction

Image manipulation programs like PhotoShop include various filters or transformations that are applied to images to produce different effects. In this lab you will write a Java program that can apply four different transformations on a loaded image:

- Invert colors
- Convert to grayscale
- Generate an "embossed" image
- Add a motion-blur effect

The first two transformations, invert and grayscale, modify each pixel independent of the values of the pixels around it. Emboss and motion-blur, on the other hand, modify each pixel based on the values of the other pixels around it. More detailed descriptions of each transformation are given below.

The PPM File Format

All of the image formats you're familiar with (.jpg, .png, .gif, etc) compress pixel information which makes it hard to edit the pixels directly. So for this lab we will be using the .ppm (portable pixelmap) file format which stores uncompressed pixel information as text.

The format of a PPM file is as follows (see this link):

1. Each PPM file begins with "P3"
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value, again in ASCII decimal. For this lab, this value will always be 255
8. A single whitespace character (usually a newline).
9. Sets of 3 numbers from 0-255, each representing an RGB color value, the first representing the red value, the second representing the green value, and the third representing the blue value. Each color value is separated by whitespace. A set of 3 color values represents a pixel.

Notes:

- The color values for a pixel (referenced in #9) are not necessarily all on the same line
- Whitespace includes newlines, spaces, tabs, comments, etc.
- There are exactly width * height pixels in the file. That is, there are in total 3*width* height color values.

- The pixels are in row-major order. That is the first n values (where n = width) represent the first row of the image, the 2nd n values represent the 2nd row, and so forth.
- At any point in the file, there may appear comments that begin with a pound sign (#) and end with a new line (\n). These should be ignored.

An alternative BNF-style PPM format specification using Java regex-like expressions is:

PPM_File	::= **Header Pixels Separator***
Header	::= **MagicNumber Separator Width Separator Height Separator MaxColorValue** \s
MagicNumber	::= P3
Separator	::= \s+ **Comment**? \s* | **Comment** \s+
Comment	::= #[^\n]*\n
Width	::= \d+
Height	::= \d+
MaxColorValue	::= 255
Pixels	::= (**Pixel** (**Separator Pixel**)*)?
Pixel	::= **RedColorValue Separator GreenColorValue Separator BlueColorValue**
RedColorValue	::= **Number**
GreenColorValue	::= **Number**
BlueColorValue	::= **Number**
Number	::= [01](\d\d?)? | 2[0-4]\d | 25[0-5] | [3-9]\d?

Hint: Characters normally stand for themselves. For example P3 is the character P followed by the character 3. Other such characters in the definition above are 6, #, 2, and 5. A special character used above is \n meaning end-of-line character. The character class [***xy***] means the character ***x*** or ***y***. The character class [***x-y***] means any character from ***x*** to ***y***. \s (defined as [\n\t\r\f\x0B]) means any whitespace character and \d (defined as [0-9]) means any digit. A sequence of characters, character classes, or groups can be grouped together by surrounding them (and). A ? following a character, character class, or group means optional. A * following a character, character class, or group means 0 or more. A + following a character, character class, or group means 1 or more.

Creating Your Own Data Structure

You will need to load the PPM to be edited into some kind of internal data structure. Remember, each pixel is made up of 3 integer color values that correspond to red, green, and blue in that order.

Tip: If you want to convert an image to .ppm you can use GIMP (Gnu Image Manipulation Program) or PhotoShop. GIMP is free online and is included in the CS Linux labs.

Command Line Syntax

java ImageEditor *inputFileName* outputFileName {grayscale|invert|emboss|motionblur *blurLength*}

- The main class should be named ImageEditor.
- The *inputfileName* is the name of a file containing a ppm formatted picture.
- The *outputFileName* is the name of a file that will contain the transformed picture. It need not exist before executing the transformation. If the file exists and is writable it will be overwritten.
- The *blurLength* is a non-negative integer. It is usually less than 25.

As an example of running the program the following command would load bike.ppm, invert the colors, and save it as bike-inverted.ppm:

```
java ImageEditor bike.ppm bike-inverted.ppm invert
```

The last command line option (representing a transformation) may be any of the following 4 strings.

- "invert"
- "grayscale"
- "emboss"
- "motionblur" which also requires a number after it to specify how much to blur

The following command would load bike.ppm, apply a motion blur of 10 pixels, and save it as blurred.ppm:

```
java ImageEditor bike.ppm blurred.ppm motionblur 10
```

If the arguments are incorrect (either the wrong number of arguments, no number after "motionblur", etc) a usage statement should be printed to the screen and the program should terminate. A usage statement informs the user what they should type to correctly run the program. A possible usage statement for this program might be the string:

```
USAGE: java ImageEditor  in-file out-file (grayscale|invert|emboss|motionblur motion-blur-length)
```

Transformation Descriptions

All transformations require an operation to be applied to each pixel in the image.

Invert

Every color value for every pixel is changed to its inverse value. For example, 0 becomes 255, 240 becomes 15, and 127 becomes 128. Remember that the minimum color value is 0 and the maximum is 255.

Grayscale

To convert an image to grayscale, each pixel's color value is changed to the average of the pixel's red, green, and blue value. For example:

Original pixel color values:
    Red: 25 Green: 230 Blue: 122

Grayscale conversion: (25 + 230 + 122) / 3 = 125 (using integer division)
    Red: 125 Green: 125 Blue: 125

Emboss

Assume an image is stored in a structure called **image**, with “height” x and “width” y. Assume also that **image**[0][0] refers to the top-left of the image. For every pixel p at position [x][y] (p = **image**[x][y]), set its red, green, and blue values to the same value by doing the following:

Calculate the differences between red, green, and blue values for the pixel and the pixel to its upper left.

```
redDiff    = p.redValue - image[x-1][y-1].redValue
greenDiff  = p.greenValue - image[x-1][y-1].greenValue
blueDiff   = p.blueValue - image[x-1][y-1].blueValue
```

Find the largest difference (positive or negative). We will call this difference maxDifference. For example, if redDiff is -10 and blueDiff is 5, maxDifference will be -10 (not 5 or 10). We then add 128 to maxDifference. If there are multiple equal differences with differing signs (e.g. -3 and 3), favor the red difference first, then green, then blue.

```
v = 128 + maxDifference
```

If needed, we then scale v to be between 0 and 255 by doing the following:

```
If v < 0, then we set v to 0.
If v > 255, then we set v to 255.
```

The pixel’s red, green, and blue values are all set to v.

Be sure to account for the situation where r-1 or c-1 is less than 0. V should be 128 in this case.

Warning: Your code may not exactly match this pseudo-code. Many students face problems at this point because they either don’t look to the upper left pixel (it may not be image[x-1],[y-1] for your code) or they base the differences on already embossed/altered pixels.

Motion blur

A number will be provided in the command line arguments if the command is “motionblur.” We will call this number *n*. n must be greater than 0.

The value of each color of each pixel is the average of that color value for *n* pixels (from the current pixel to *n*-1) horizontally.

Example: if we store the pixels in a 2d array, the motion blur would average each color from pixel[x][y] to pixel[x+n-1][y]

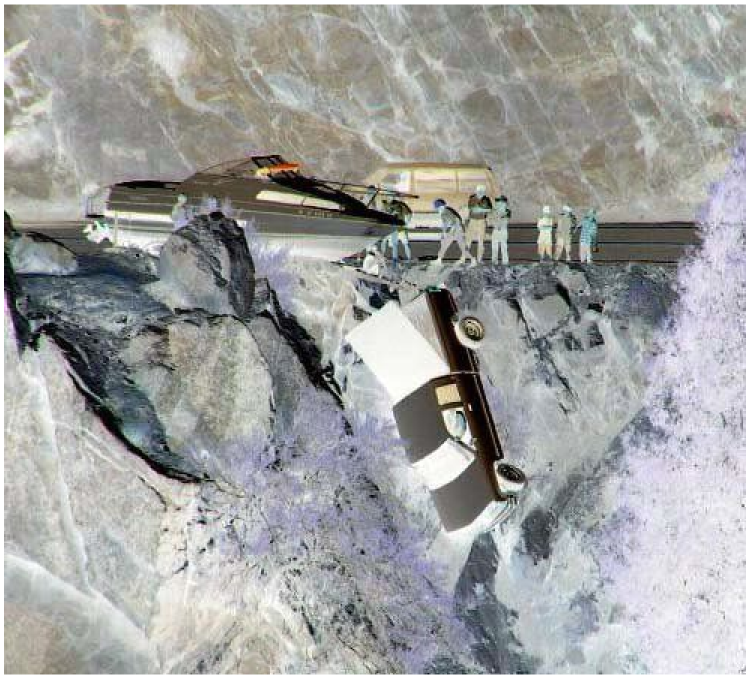
Be sure to account for the situations where one or more of the values used in computing the average do not exist. For example, if an image has width w and we are considering the pixel on row r, column c, if c + n >= w, then we only average the pixels up to w.

Examples: see below

```
java ImageEditor crash_boat_jeep.ppm inverted_crash_boat_jeep.ppm invert
```

The crash_boat_jeep.ppm looks like:

The inverted image looks like this:



```
java ImageEditor policedonut.ppm grayscale_policedonut.ppm grayscale
```

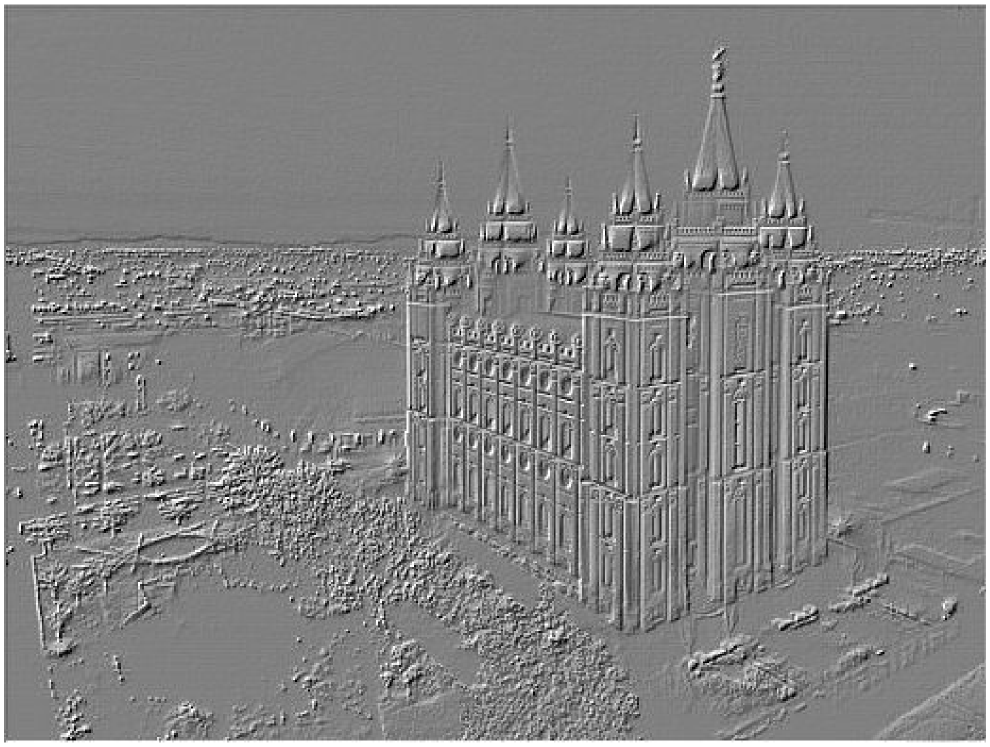
policedonut.ppm looks like:

The grayscale image looks like this:

```
java ImageEditor temple.ppm embossed_temple.ppm emboss
```

The temple.ppm looks like:

The embossed image looks like this:



```
java ImageEditor funny_cat.ppm motionblur_funny_cat.ppm motionblur 20
```

The funny_cat.ppm looks like this:

The motionblur 20 image looks like this: