Assignment 1
Image Processing
Due: September 16, 2010

Overview:

In this assignment you will implement the basic methods involved in 2D graphics. The focus will be mostly on implementing filters which take an input image, process it, and produces a modified output image.

Getting Started:

You should use the skeleton code (from here for C++, and here for Java) as a starting point for the assignment. There are several files which are provided for you to setup the framework, but you should only change `image.cpp`.

The files which need to be in your project are (C++):

- `main.cpp` – Parses the command line arguments, calls the appropriate image functions
- `EasyBMP.[h/cpp]` – Reads and creates Bitmap for images (.bmp)
- `pixel.[h/cpp]` – Pixel processing
- `image.[h/cpp]` – Image Processing **(This is what you need to edit!)**

For Java you need:

- `pixel.java` – Pixel processing
- `image.java` – Image Processing **(This is what you need to edit!)**

One starter image is attached, but you can find more in your personal collection or on the internet. For the C++ code, the images must be bitmaps (.bmp), for Java other files may be read in with minor modifications to the code though bitmaps will work out of the box.

A sample implementation for Windows is here!

How the Program Works:

The UI in this assignment was kept as simple as possible, so you can concentrate on the image processing. The program runs on the command line and performs operation in the order that they appear in the arguments. For example, to increase the brightness of the image `in.bmp` by 10%, and save the result in the image `out.bmp`, you would type:

```
image -input in.bmp -brightness 1.1 -output out.bmp
```

Notice the `input` parameter must appear first. Remember, everything happens in the order specified. First the input, then the brightness change, then the write back out.

For several of the filters, there is more than one corresponding argument. To see the complete list of options, type:

```
      image –help
```

If you specify more than one option, the options are processed in the order that they are found. For example,

```
image -input in.bmp -contrast 0.8 -scale 0.5 0.5 -output out.bmp
```

would first decrease the contrast of the input image by 20%, and then scale down the result by 50% in both x and y directions. It is also possible to specify `-output` multiple times, to save out intermediate results:

```
image -input in.bmp -blur 5 -output blurred.bmp -edgeDetect -output
edges.bmp -rotate 30 -output whatever.bmp
```

What You Have to Do

The following is a list of features that you may implement (roughly from easiest to hardest). The number in front corresponds to how many points the feature is worth.

- (1) **Brighten**: This filter is done for you and you can use it as a starting point.
- (3) **Random** noise: Add noise to an image.
- (3) **Crop**: Extract a subimage specified by a corner and area.
- (3) **Extract** Channel: Leave specified channel intact and set all others to zero.
- (3) **Contrast**: Change the contrast of an image.
- (3) **Saturation**: Change the saturation of an image.
- (4) **Sharpen**: Sharpen an image by extrapolating from a blurred version.
- (5) **Quantize**: Change the number of bits per channel of an image, using simple rounding.
- (5) **Random dither**: Convert an image to a given number of bits per channel, using a random threshold.
- (10) **Blur**: Blur an image by convolving it with a Gaussian low-pass filter.
- (10) **Edge detect**: Detect edges in an image by convolving it with an edge detection kernel.
- (10) **Ordered dither**: Convert an image to a given number of bits per channel, using a 4x4 ordered dithering matrix.
- (10) **Floyd-Steinberg dither**: Convert an image to a given number of bits per channel, using dithering with error diffusion.
- (10) **Scale**: Scale an image up or down by a real valued factor.
- (10) **Rotate**: Rotate an image by a given angle.
- **(**10) **Fun**: Warp an image using a non-linear mapping of your choice (examples are fisheye, sine, bulge, swirl).
- (up to 10) **Nonphotorealism**: Implement any non-trivial painterly filter. For inspiration, take a look at the effects available in programs like *The Gimp*, *PhotoShop*, and *Image*

Composer (e.g., impressionist, charcoal, stained glass, etc.). The points awarded for this feature will depend on the creativity and difficulty of the filter. At most one such filter will receive points.

For any feature that involves resampling (e.g. scale, rotate, fun), you must provide three sampling methods: point sampling, bilinear sampling, and Gaussian sampling.

Submission

You should crease a webpage with:

- Your modified image.cpp
- At least one result image for each implemented filter, along with the command-line arguments used to make the image
- A write-up
- Submission for the art contest (optional, but this is probably the easiest assignment for this…)

Send me only a link to the webpage, as always no attachments.

Hints

- Do the simplest filters first!
- Look at pixel.[cpp/h,java]. You may find helpful functions for manipulating pixels there (e.g. Interpolation)
- The brighten filter is implemented for you. This should get you started and show you how to access and modify pixels in an image.