Assignment 1 – Image Processing
Due: October 1st (4:00pm), Check-in: September 24th

Overview:

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

Getting Started:

You should use the skeleton code on the course webpage as a starting point for the assignment. There are several files that are provided for you to setup the framework and read/write image files, but you should only change `image.cpp`.

The files that need to be in your project are:

- `main.cpp` – Parses the command line arguments, calls the appropriate image functions
- `stb_image.h` – Reads images files (.bmp, .jpg/.jpeg, .png, .tga)
- `stb_image_write.h` – Write images files (.bmp, .jpg/.jpeg, .png, .tga)
- `pixel.[h/cpp]` – Pixel processing
- `image.[h/cpp]` – Image Processing **(The only file you need to edit!)**

A small number of starter images are included with the code, but you can find more in your personal collection or online (https://www.google.com/imghp). The images you use must be one of the supported formats (.bmp, .jpg/.jpeg, .png, .tga).

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 writing to the specified output file.

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 encountered. 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 allCombined.bmp
```

What You Have To Do:
The following is a list of features that you should implement (roughly by difficulty). 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 randomized 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) **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 in size 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).

(10 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.

Check-in. You must turn in 10 points worth of features by Sept 24.

Extra Credit –
In addition the above required filters you have some options for extra credit. These are somewhat harder to implement than the above features but very rewarding to complete. These are independent and optional; feel free to do all, some, or none.

- **Nonphotorealism** (up to 10 extra): Implement any non-trivial painterly filter. For inspiration, take a look at the effects available in programs like *Gimp* or *PhotoShop*, (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.

- **Lossy Compression** (up to 10 extra)**:** Design your own compressed image format. Similar to jpeg, this should be a lossy (not lossless) format. The image you save will be slightly different from the original, but the saved image and original image should look very similar. You must write your own code both to save an image to a file in your format and to read it back. Points will be awarded based on how well images are compressed (i.e., low file size and high image quality), the explanation of your approach, and the analysis of your results.

- **Ordered dither** (up to 5 extra): Convert an image to a given number of bits per channel, using a 4x4 ordered dithering matrix. You will have to scale these matrix values based on the level of quantization.

Submission Details:
Submission should be in the form of a link to a webpage. The web page must contain:
- Your modified image.cpp
- At least one result image for each implemented filter, along with the command-line arguments used to make the image
- At least one triplet of images showing the differences between different sampling methods
- A write-up of your experiences, difficulties and successes (~1 paragraph)
- Submission of an artistically interesting image to the art contest (this is optional, but this is probably the easiest assignment to do it for…)

Hints:
- Do the simplest filters first!
- Look at pixel.[cpp/h]. You may find helpful functions for manipulating pixels there (e.g. Interpolation)
- I found it useful to convert the pixel values to floats in the range (0,1), manipulate them, and then convert back to 0-255. This can avoid some problems with integer math.
- Be careful about negative numbers of those greater than 255. This will wrap causing strange looking errors in your output.
- The brighten filter is implemented for you. This should get you started and show you how to access and modify pixels in an image.