MP7 image convolution

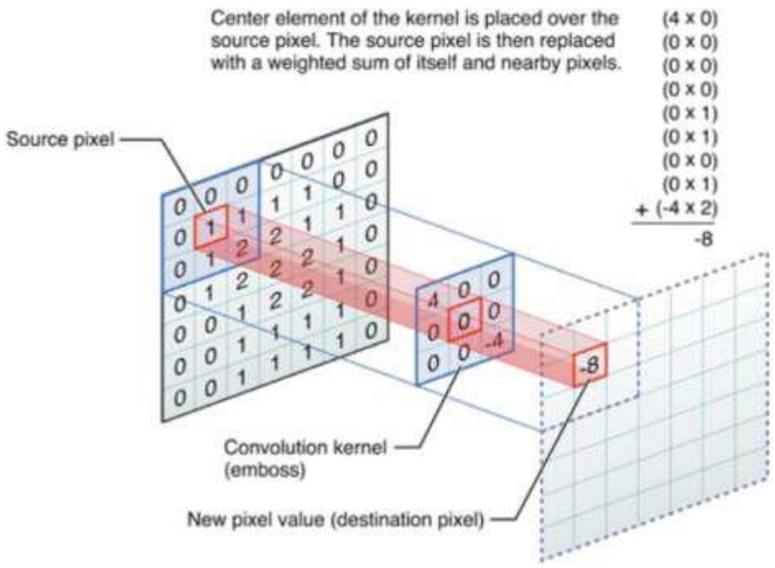
Due on Wednesday Oct 15th 2014

Image Convolution/Pixelate

Your task this week is to implement a Gaussian filter, image convolution and pixelate. Gaussian filter is a type of image filter that uses a Gaussian function for calculating the filter to apply to each pixel in the image by convolving the image with the filter. The contain also contains other filters that you can apply to input images to achieve other effects once you have correctly implemented image convolution. Finally, you will implement pixelate which decreases the resolution of the image by changing the values of pixels in a block with the average taken over the entire block. The objective for this week is for you to gain some experience with arrays and pointers, to implement code using subroutines, and to solve a problem using programming knowledge gained from the class.

Background

An image is composed of an array of pixels. Each pixel has four bytes of information in the RGBA format representing the red, blue, green and alpha channels. The values for each of the channels range from 0 to 255. Alpha channel is reserved for transparency information and may not be present in all images. Images can be modified by a process called filtering. A filter in image processing is an array that is applied to the entire image by a process shown in the image below:


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

You overlay the filter over the source pixel and multiply each overlapping value in the filter by the corresponding value in the image, then sum all of these values (as shown in the upper-right of the figure). This process is called convolution. The resulting sum of products should never be out of bounds (< 0 or > 255). You should clamp the values should they go out of bounds. In other words, replace values larger than 255 with 255, and values smaller than 0 with 0.
The convolution process is performed for each pixel of each color array (red, green, blue). The alpha

channel may or may not be included in the process and is dependent on the type of image processing done. It will be made clear before the filter process whether to include or not include the alpha channel. You may notice that if the filter is placed over a pixel at the edge of the image, it will not have enough information to process. Therefore, you can consider the pixels outside of the image to be zero-valued pixels (equivalently, ignore them). In other words, you will apply the filter only to the available pixels. Do not access values beyond the array bounds!
In order to generate the filter for Gaussian blur, a Gaussian function is used. The equation of the Gaussian function in two dimensions is given as

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right]$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and sigma is the standard deviation of the Gaussian distribution. x and y go from -radius to radius. The meaning of radius will be explained soon. In theory, the Gaussian function at every point on the image will be non-zero, meaning that the entire image would need to be included in the calculations for each pixel. In practice, when computing a discrete approximation of the Gaussian function, pixels at a distance of more than 3 sigma are small enough to be considered effectively zero. Thus contributions from pixels outside that range can be ignored. Hence, sigma determines the size of the filter. The radius of the filter which is the number of elements extending in the either direction from the center can be determined by the following equation

radius= ceil(3*sigma)

For example, if sigma = 2.5, then radius = ceil(7.5) = 8. This means that the filter array is 17 × 17 (indices from 0 to 16 in the filter, applied from -8 to 8 around each pixel in the image). Note that the ceil function is a standard C math library function.

Pieces

Let's discuss each of these in a little more detail:

functions.h- This header file provides function declarations and descriptions of the functions that you must write for this assignment.
functions.c- The main source file for your code.Function headers for all regular and challenge functions are provided to help you get started.
test – This is the executable file that tests your code. It compares your solution with the gold solution for all the functions you have written.
imageData.c/imageData.h- These two files implement different filters that you apply to your image. It also implements the function that encodes/decodes png files.

Details

Below are the tasks you need to complete for this assignment.

1 getRadius and calculateGausFilter functions will be done in lab section
2 Implement the convolveImage function
3 Implement the pixelate function
4 Implement the challenge functions

Creating the Gaussian Filter

To create the Gaussian filter, you will use Equation (1). In the program, you will create the Gaussian filter in the function:

```
void calculateGausFilter (double *gausFilter, double sigma);
```

Here, **gausFilter** represents a pointer which points to the Gaussian filter which has already been allocated enough memory. You need not allocate any memory. You just have to calculate the Gaussian values. The radius of the filter is calculated using Equation (2). You need to implement this in the function and use it to get the radius value:

```
 int getRadius(double sigma);
```

Once the Gaussian values for the filter are calculated, you need to **normalize** the filter. To normalize the Gaussian filter, you first add up the values of each element in the filter array. This sum is known as the 'weight'. You then divide the value of each element in the filter array by the weight. After normalization, the numbers in your filter should sum to exactly 1.

Applying the Gaussian Filter

To apply the Gaussian filter to the image you will use the Gaussian filter created in the above function and the convolution process described in the Background section. **You need to apply convolution for only the red, blue, and green channels and copy the alpha channel unchanged from the source image**. Note that if the radius of the filter is less than one, the input image should not be altered. In the program, you will implement your logic in the function:

```
void convolveImage (uint8_t *inRed, uint8_t *inBlue, uint8_t *inGreen,



uint8_t *inAlpha, uint8_t *outRed, uint8_t *outBlue, uint8_t *outGreen, uint8_t
*outAlpha, const double *filter, int radius, int width, int height);
```

Here, **filter** is the pointer to either the Gaussian kernel created in the previous function or other filter provide in ImageData, height and width are the height and width of the image. In your program you should make use of the following math functions and constant which are already defined and included:

```
double sqrt (double x );
double ceil (double x );
double exp (double x );
MPI - pi constant
max(x,y);
min(x,y);
```

Pixelate

To create a pixelated image, use the pixelX/Y parameters which create a block of pixelX*pixelY dimensions that is overlaid repeatedly from the upper left corner of the image (starting at (0,0)). Set the color of each block to the average color of all pixels in the block including the alpha channel. At the boundaries, use only the available pixels.

Challenges

Here are some challenges for this week. You can find the function signatures in the header or the source file. For the challenges, you
must either write your own tests or use the graphical version to test on the lab machines.

(20 points total) - Implement Pencil sketch effect. To do this, you need to implement the following functions.
invertImage (5 points) - Inverts the value of the pixel p, in each channel to 255 - p. Alpha channel is not inverted but just copied from the input channel.
colorDodge (5 points) – Blends two images together. The relation between the two pixels top and bottom is *((top=255)?top:min(((bottom<<8)/(255-top)),255))*. This equation makes use of the ternary operation which is essentially an if statement rewritten into a single line. The ternay operation can be described as: *condition ? value_if_true : value_if_false*. You will have to perform the above equation on only the red, blue, and green channels and set the alpha channel to that of the bottom channel.
convertToGray (3 points) - Converts the image to gray scale. Each pixel will be set to the weighted average according to the formula 0.299R + 0.587G + 0.114B. Alpha channel will be retained from the input image. You just have to implement these functions. We have written the code to call each of these functions serially to obtain the pencil sketch effect.
flipImage (5 points)- Flips the image in both the horizontal and vertical directions.
pencilSketch (2 points) – **To complete this challenge, you will have to implement convertToGray, invertImage, and colorDodge first.** This function should be a simple function that calls on other helper function. To perform this effect, you will need convert the input image to grayscale and invert that image, apply a Gaussian blur to the input image, and then use the colorDodge function to combine these two images together. Note that you will be generating two intermediate images which can be stored at the locations pointed by the inv* and blur* pointers. When combining the two images with a colorDodge, the blurred image should be the bottom layer while the inverted grayscale should be the top.

Specifics

- You program must be written in C in the file **functions.c** provided for you.
- You must implement **getRadius**, **calculateGFilter**, **convolveImage**, and **pixelate** correctly.
- Your routine's return values and outputs must be correct.
- Your code must be well commented.

Building and Testing

We suggest that you begin by developing your code on the Linux machines in the lab. **You should get in the habit of writing your own tests.** We have provided a test program that individually checks each of the functions implemented. It checks the output of your functions with the output from the gold solution and tells you whether or not your input it correct. We have also provided a zipped file of images on MP page so you can see what the output of each of the

make
Once compiled, run the program from MP7 directory as:
./mp7 <inputfile> <outputfile> <command> <sigma>
Sigma is an optional non-negative value that will only be used when performing convolution with a Gaussian filter or performing a pencil sketch (command 0,11); otherwise, the program will give you an error. If omitted, sigma is initialized to 3.

Run the test from MP7 directory as:
./test

Grading Rubric

Functionality (70%)
10% - **calculateGFilter** works correctly.
35%- **convolveImage** function works correctly.
30%- **pixelate** function works correctly
5% - **getRadius** works correctly

Style (10%)
5% - Compilation generates no warnings. Any warnings means 0.
5%- Indentation and variable names are appropriate and reasonably meaningful

Comments, clarity and write-up (10%)
5% - Introductory paragraph explaining what you did. Even if it is just the required work.
5%- Code is clear and well commented.

Some point categories in the rubric may depend on other categories. If your code does not compile, you may receive a score close to 0 points.