Homework One

I. Anisotropic Diffusion [50 points]

1) Implement a Matlab function that does linear filtering with a $3 \times 3$ kernel $h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}/4$. The function has the following interface:

```matlab
function [g] = myfilter(f)
% MYFILTER filters grayscale image f using a
% kernel \[0 \ 1 \ 0; \ 1 \ 0 \ 1; \ 0 \ 1 \ 0\]
```

Use `imfilter` or `conv2` in the implementation. Choose the padding style as ‘replicate’.

2) Implement the same Matlab function without using `imfilter` or `conv2`. Let’s look at the convolution:

$$g(i,j) = (f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1))/4$$

Based on the above equation, the straightforward solution is to use for loops to traverse the image and compute the average at each pixel. Even though such an implementation is natural in Java or C, it is terribly slow in Matlab. We have to eliminate the loops. Revisiting the equation above, we notice that $g$ is in fact the average of 4 images, which are the shifted $f$ by one pixel in 4 different directions. For instance, $f(i-1,j)$ is $f$ that shifts by 1 pixel to the right. We can do the shift in the following way using Matlab:

```matlab
fright = [f(:,1), f(:,1:end-1)];
```

Here we duplicate the first column and discard the last column. Other shifts can be done similarly. Let’s call them `fleft`, `fup` and `fdown`. Then

$$g = (fright + fleft + fup + fdown) / 4;$$

Complete the above Matlab function. Write a script to apply your function to some test image for 1, 10 and 50 times, i.e., feed the result image back to the filter for specific number of times. Attach your code and the result images with your submission.

3) As you will notice, the operation smooths the image and also destroys the sharp structures. We can modify the above implementation so that the smoothing is done mostly on the “flat” areas or along the edges. Let’s rewrite $g$ in a different but equivalent form:

$$g(i,j) = f(i,j) + 0.25 \times (f(i-1,j) - f(i,j) + f(i+1,j) - f(i,j) + f(i,j-1) - f(i,j) + f(i,j+1) - f(i,j))$$

In Matlab, we can use the shift trick mentioned above to eliminate the loops:

```matlab
g = f + 0.25 * ((fup-f) + (fdown-f) + (fright-f) + (fleft-f));
```

where `fup`, `fdown`, `fright` and `fleft` are shifted $f$ by one pixel in four different directions.
The new equation indicates that the filtering process passes the influence of neighbors to the center pixel. If we do the filtering for many times, we gradually diffuse the values throughout the space. The equation in fact is very similar to how heat transmits in uniform media.

We can change the diffusion rate based on the image local structure to achieve the detail preserving effect, i.e., we set the diffusion factor so that it is more effective at the smooth regions or along the edge in which case the neighbor pixel different is relatively small. If the neighbor difference is big, we prohibit the diffusion. An often used diffusion factor function is $w(x) = \frac{K}{K + x^2}$, where $x$ is the difference of neighboring pixels. To simplify the notation, we use $d(m,n)$ to indicate $f(m,n) - f(i,j)$. And our new $g$ is

$$g(i,j) = f(i,j) + 0.25 \times (w(d(i-1,j)) \times d(i-1,j) + w(d(i+1,j)) \times d(i+1,j) + w(d(i,j-1)) \times d(i,j-1) + w(d(i,j+1)) \times d(i,j+1))$$

The above filtering processing is nonlinear and called anisotropic diffusion. Write a Matlab function to implement the function using the following interface:

```matlab
function [g] = anistropic_diffuse(f, K, n)
```

The parameter $n$ is the number of iterations and $K$ is the parameter of the diffusion factor function. You can use the same trick as the previous linear filter to eliminate the loops. Test your function on some images. Typical $K$ is 0.01. Try to increase and decrease $K$ and see what happens. Attach your code and the results with your submission.

4) Spatially Variant Blurring – Bonus Question [25 points] Based on the above methodology, implement a Matlab function that does blurring controlled by a blur map. The blur map can be generated manually. The following example shows how this can be used to turn a picture taken by a cheap camera into a more professional look.

![Fig. 2. Spatially variant blurring. Left: Input image; Middle: blur map; Right: blurred result.](image)

You may need the function `roipoly` to generate a mask for an object that you do not want to blur. Think of some better way to generate the blur map. Submit your code and the results on some test images.
II. IMAGE ALIGNMENT [50 POINTS]

In this project, you’ll automatically create color images from the black and white photographs of Sergei Mikhailovich Prokudin-Gorskii (1863-1944). The download directory contains scanned negatives of the Prokudin-Gorskii images from the Library of Congress. Downdown the images at www.cs.bc.edu/~hjiang/csci343/projects/1/images.zip.

A. Programming

You should write a single function that takes a grayscale Prokudin-Gorskii image as input, and returns a high quality RGB image as output. Your function should be completely automatic, i.e. it should require no help from the user! Note that the order of the color channels, from top to bottom, in the Prokudin-Gorskii images is BGR, not RGB.

Write a simple brute-force alignment algorithm that first splits the image into 3 parts. Use the G channel as the anchor channel, and find the best displacements for the R and B channels. Once you have the displacements, you can construct the RGB image by put them into corresponding color channels and display one the screen. How do you determine which is the best displacement? Use a simple image comparison metric such as the sum of squared difference (SSD) or the normalized cross correlation (NCC). NCC is the dot product between two normalized vectors. You may want to do normalization on small images blocks instead of the whole image. You also may want to exclude the border from the comparison. Consider using the demo code in our class for localizing objects as a basis for this project. The codes are available at the note section of the course web site (http://www.cs.bc.edu/hjiang/csci343/notes.html).

You’ll notice that if the matching kernel size D is large or if the images are large, then this solution is prohibitively slow: The time complexity is $O(n \times D^2)$, where $n$ is the number of pixels in the image. Discuss possible ways to speed up the computation.

Matlab functions that may be useful for this assignment:
- help - as always!
- tic, toc - timer
- circshift - one easy way to deal with displacement
- cat - for creating the RGB image from the 3 parts
- imfilter - image filtering (convolution) with helpful boundary options
- conv2 - raw 2D convolution
- imresize - resize an image
- im2double - convert image values to [0,1]
- rgb2gray - convert a rgb image grayscale

B. Writeup

Show the aligned RGB image for as many input image in the download directory as possible. If you cannot align an image, explain why. Also, for each image, report the time your code took.

III. WHAT TO HAND IN

Submit your code and writeup electronically to the Canvas system. You do not need to hand in a printout.