Image Filtering

In this assignment, we use matlab in image filtering. Question I and II have to been answered. Answering Question III is optional. You are encouraged to use your favorite images to test your code. You should write a report that includes codes and image processing results. Submit your report in PDF. The Matlab codes must also be submitted separately to blackboard. This assignment is due on Sept. 30, 2009.

I. MATLAB BASICS [10 POINTS]

1) Go through the matlab tutorial at www.cs.bc.edu/~hjiang/cs390/matlab.pdf. If you are familiar with Matlab, go to 2) in this section.
2) Write the output of the following Matlab expressions without using Matlab command line.
   a) `v = 1: 2: 5`
   b) `A = [ones(1,2) zeros(1,3); zeros(2,3) ones(2,2)]`
   c) `A = [1 2 3; 4 5 6; 7 8 9]; B = [A(1,1:2); A(3,1:2)]`
   d) `A = [1 2 3; 4 5 6; 7 8 9]; B = A([1 3], end)`
   e) `A = [1 2 3; 4 5 6; 7 8 9]; B = [A(:,1) A(:,1:2)]`

II. ANISOTROPIC DIFFUSION [50 POINTS]

1) Implement a Matlab function that does linear filtering with a 3×3 kernel `h = [0 1 0; 1 0 1; 0 1 0]/4`. The function has the following interface:

```
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 expand 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:

```
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 smoothes 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 * (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))
```

Fig. 1. Comparison of image filtering. Left: original image; Middle: smoothed result using linear filtering for 50 iterations; Right: anisotropic diffusion with $K = 0.0001$ in 50 iterations. Anisotropic diffusion preserves local details in the smoothing.

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 $w(x) = K/(K + x^2)$, 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 * (w(d(i-1,j)) * d(i-1,j) +
                          w(d(i+1,j)) * d(i+1,j) +
                          w(d(i,j-1)) * d(i,j-1) +
                          w(d(i,j+1)) * 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:

```
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.

III. 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.

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.