Computational Photography - Assignment 2.

March 6, 2008

1 Introduction

The objective of this assignment is for you to become familiar with some simple image processing techniques using image pyramids.

The assignment has three main parts:

- (i) A demonstration of convolution using the Fast Fourier transform, and how it can speed up simple processing operations.
- (ii) Image blending using Laplacian pyramids.
- (iii) Noise removal using a steerable pyramid.

2 Requirements

You should perform this assignment in Matlab. Please turn in all code and result figures/images. The code should be clearly commented, explaining what each line is doing.

Parts (ii) and (iii) of the assignment require the use of Eero Simoncelli's excellent Matlab Pyramid toolbox. This can be downloaded from `http://www.cns.nyu.edu/pub/eero/matlabPyrTools.tar.gz`. Download the code and unzip it (if you are using Windows then WinRAR can uncompress `.tar.gz` files). Add the path `../matlabPyrTools`[1] to your Matlab path with the `addpath` command.

[1] Expert users only: If you want to use compiled versions of the functions (which prevent the error messages happnening) you need to compile the MEX versions and then add `../matlabPyrTools/MEX` to your Matlab path.

To check that you've got things setup correctly, load the `einstein.pgm` image (in `assign2.zip` from the project webpage) with the `imread` command. Then try the following few lines:

```
[pyr,ind]=buildSFpyr(im,3,3); % 3 levels, 4 orientation bands
showSpyr(pyr,ind); % show steerable pyramid
res=reconSFpyr(pyr,ind); % invert pyramid
imshow(uint8(res)); % display reconstructed image
```

You can ignore the warning messages – you don't need to build the MEX files for this assignment, the code is plenty fast enough. Also, remember that you can always get help in Matlab by typing `help` *`command_name`*.

3 Convolution using Fast Fourier Transform (FFT) [10%]

The Fast Fourier Transform (FFT) is a widely used trick for efficiently convolving two signals. Here you will explore the speed advantage it gives over standard, spatial domain, convolution.

First, create a random 1 megapixel array (1024x1024 pixels) using the `rand` command. If you have an actual image handy you can use this instead, the exact size not being critical to the results. Next, generate some random k by k filter of size $k = 15$ using the `rand` command again.

The straightforward approach to 2D convolution is to use the `conv2` command. Use the `same` option, so the input is the same size as the output. Using this command, measure how long it takes to convolve the 1 megapixel array with the filter using the `tic` and `toc` commands.

An alternative approach is to transform both the image and filter into the Fourier domain using the command `fft2`, multiply the two together (recall that a convolution in the spatial domain becomes a multiplication in the Fourier domain) and then invert back to the spatial domain with `ifft2`. Try this out and time how long it takes. Check that you get the same answer as with the `conv2` command. You may need to remove the imaginary part of the answer (due to numerical precision issues) with the command `real`.

Now loop over different values of k, using $k = 3$ upto $k = 35$, and plot a graph showing the relative performance of the two methods. Please export the plot to an image turn it in.

4 Image blending [35%]

On the course web-page, download the second Burt and Adelson 1983 paper, "A Multiresolution Spline with Application to Image Mosaics". You will be implementing the algorithm that splines regions of arbitrary shape together, as described in section 3.2 of the paper. I suggest that you read the whole paper though – it is well written and will reinforce the concepts mentioned in the lecture.

Pick two images containing objects that you want to blend together. Extra credit will be given for the artistic/horror/shock value of the final image, so choose something fun! Then generate the mask image to select the blending boundary. The `roipoly` command may be useful.

Now implement the algorithm described on page 230 of the paper. The `matlabPyrTools` contains functions for computing Gaussian and Laplacian pyramids (and reconstructing them): `buildGpyr`, `buildLpyr` and `reconLpyr`. Note that you should not directly manipulate the `pyr` or `indices` structure: use the functions `pyrBand` and `setPyrBand` to set/get the sub-bands of the pyramid. Save your blended image and turn it in.

5 Image denoising [45%]

Here you will implement an approach from the Simoncelli and Adelson 1996 paper, "Noise Removal Via Bayesian Wavelet Coring", posted on the web-page. The goal is to denoise an image using the distributions of sub-band coefficients in a steerable pyramid. Since these distributions are different to those produced by noise, it is possible to build a "coring function", a 1-D non-linear mapping of the sub-band coefficients that will reduce noise in the sub-bands, yielding a denoised image.

The key equation in the paper is Equation 1 on page 2 of the Simoncelli paper. This equation gives us a Bayesian estimator of the value of the true sub-band coefficient value x, given a noisy value y. Note that both the top and bottom parts of the equation are convolutions of the signal and noise distributions ($P_x(x)$ and $P_n(x)$ respectively), with the added x term in the top.

First, download `assign2.zip` from the project webpage. It contains two images, `feynman.pgm` and `einstein.pgm`. Load them up into memory and make the images in the range $0 \ldots 1$. The Feynman image will be used to

build the Bayesian estimators (note the plural – we will build a separate one for each of the high-frequency sub-bands). Although the two images are quite different to one another, their sub-band statistics will be similar (compared to those of the noise), hence we can use one image as a prior for denoising the other image.

The Einstein image will act as our test image. To make it noisy, add zero mean Gaussian noise to it with standard deviation $\sigma = 0.05$ using the function `randn`. Also, generate a pure Gaussian noise image of the same size as the Einstein images with the same standard deviation.

You should now have 3 images: a noisy Einstein image (henceforth the *signal* image); a clean Feynman image (henceforth the *prior* image); and a noise image. Now compute the steerable pyramid of each image, using the command: `[s_pyr,s_ind]=buildSFpyr(signal_im,3,3);` for the signal image and likewise for the other two. This gives us pyramids with 3 levels, each having 4 orientations. `s_ind` contains the indices of the different sub-bands. Note the high-frequency ones (which have more elements) come first, followed by the lower frequencies.

Now, we will only be denoising the high frequency residual and the 4 highest frequency sub-bands, since they contain most of the noise. In other words, we are only going to alter the top 5 levels of the pyramid.

So, for each of the top 5 sub-bands:

1. Extract bands from the prior and noise pyramids using `pyrBand`.

2. Histogram the coefficients of the band from the prior image, using bins from -0.3 to 0.3. Add a small regularization term of 0.1 to all bins (having zero causes mayhem).

3. Histogram the coefficients of the band from the noise image, using bins from -0.3 to 0.3. Add a tiny regularization term of *eps* to all bins (having zero causes mayhem).

4. Use these two histograms as estimates of $P_x(x)$ and $P_n(x)$ and compute the estimator for the sub-band using Equation 1. You can use the `conv2` command with the `'same'` option, even though the signals are 1-D, to get a same-sized vector as the input.

5. Store the estimator.

Ok, now plot the estimator curves out. Thy should be close to the diagonal but will wiggles near zero. Now we can use them to denoise the signal image. In turn, extract the same 5 high-frequency sub-bands from the *signal* steerable pyramid. Use `interp1` to remap the sub-band coefficients, making sure you use the estimator for that particular sub-band. Re-insert the remapped sub-band into the pyramid using the command `setPyrBand`. Finally, convert the signal pyramid back to an image using `reconSFpyr`. Hopefully it is less noisy than the input image.

For a baseline comparison, use the function `wiener2` to denoise the signal image, using $M = 3, N = 3$ and a noise level of 0.05^2 (the function requires the noise power, not the noise standard deviation).

Now compute Power Signal to Noise Ratios (PSNR's) between the clean Einstein image and the noisy version, the Wiener baseline and your output. The function `PSNR.m` does this for you and is in the .zip file. Bigger numbers are better here, so hopefully your code is able to beat the Wiener denoising. Print out these figure and turn them in along with images of the signal image, Wiener denoised image and your denoised image.