Where's Abe?

MATLAB Concepts Covered:
1. Image manipulation: blkproc
2. Edge detection: LoG and thresholding
3. Image analysis in the frequency domain: 'fft2', 'ifft2'
4. Band-rejection filters
5. Critical-band Masking (see accompanying paper: Science 1973)
6. Zero-crossing and the "raw primal sketch"

Corresponding MATLAB script is lincoln2.m

Reference: pp. 54-79 of David Marr's monograph, "Vision" (New York: W. H. Freeman and Company; 1982)
See also: Harmon LD & Julesz B (1973) "Masking in visual recognition: Effects of two-dimensional filtered noise." Science, 180:1194-7.

Certain types of image manipulations can make them quite difficult to recognize. For example, removing information from an image, whether through excessive smoothing, thresholding or discrete sampling can make a previously clear image uninterpretable. However, in a previous example (see: natural images tutorial) we added random noise to an image and saw that, because of the large amount of correlation amongst neighboring pixels, we could still clearly decipher the image.

Neuroscientist David Marr postulated that the visual system might extract edges and other salient features from images by relying on something called 'zero-crossings.' He argued that by comparing these features across multiple spatial scales, the brain could easily extract important aspects of the visual world. To gain an intuition about these ideas and see how MATLAB can assist with image analysis and manipulation, let's first load an image of America's 16th president.

Step 1: Coarsely re-sample Honest Abe
Discretely sampling and then quantizing and image can render it difficult to recognize. For example:

> *Load the image of Lincoln ("lincoln.jpg") and display it.*
> *Then use 'blkproc' to coarsely sample the image, replacing each block of native pixels with the mean for that block. HINT: use 'bestblk.m' to create your block size.*
> *What does this image look like?*
> *To gain an intuition into how this changes the image, try a few different size 'grains' for sampling*
> See Step 1 in lincoln2.m for help

The discretized image is difficult to recognize. Yet, if you squint, you might notice something interesting.

> *What do you notice?*

Step 2: Find zero-crossings using the 'log' filter
In 1980, David Marr and Ellen Hildreth wrote a very important paper in which they developed the now-famous "vel-squared-G" filter for image processing—we've met this previously as the "LoG" or "Laplacian of Gaussian" filter. Marr went on to argue in his monograph that one important function of early vision was to determine which changes in intensity in the image corresponded to real features in the world, such as edges. He suggested that the way this was accomplished was to compare zero-crossings at multiple spatial scales. Let's see what these actually look like for our pixelated Abe.

> *What do we mean by zero-crossings? How can we find them?*

Use the 'log' filter to detect zero-crossings at three different spatial scales, corresponding to sigmas of 4, 7 and 12. See Step 2 in lincoln2.m for help
Why did we use 3 different sigma values, what might they correspond to? (see below)
What is the relationship between the size of sigma used in this function on the numbers and types of zero-crossings that are observed?

Marr argued that the nature of the physical world placed constraints on the zero-crossing geometry in the different channels, referring to this as the "spatial coincidence assumption":

> "If a zero-crossing segment is present in a set of independent [LoG] channels over a contiguous range of sizes, and the segment has the same position and orientation in each channel, then the set of such zero-crossing segments indicates the presence of an intensity change in the image that is due to a single physical phenomenon . . . ."

He further argued that if the zero-crossings of the larger channels are "accounted for" by the smaller ones, then the visual system interprets the world as roughly what the smaller channels are "seeing."

Step 3: Combining zero crossings from different channels

Compare the zero-crossings in the 3 different channels from Step 2, by overlaying them using imagesc.
A better way to visualize this, would be to preserve the identity of each channel. Make a plot that does this.
See Step 3 in lincoln2.m for help

So, according to Marr, even though our large channels are seeing something we might recognize as Abe, the fact that our small channels can account for it causes us to see what the small channels are seeing: a bunch of blocks. By squinting, we are removing the small-channel information (essentially, we are applying a bigger Gaussian) and thus revealing the stuff in the larger channels, which look more like Abe.

Step 4: Blurring with a Gaussian instead of a squint
Instead of asking MATLAB to squint for us, we can instead see Abe by removing the high spatial frequency information that is provided by the small channels.

Start with the output of blkproc from Step 1 and remove the high spatial frequency (i.e. small channel) information by blurring the image with a Gaussian filter of sigma = 7. See Step 4 in lincoln2.m for help
Try other filter sizes, how does this change the output?

Not bad. That actually looks more like Abe than the pixellated image we started with. Another way of thinking about what we just did is to say that the signal in this image was in the low frequencies and the noise was predominantly in the high frequencies. So, our Gaussian filter removed noise by getting rid of the high frequencies.

Step 5: What's the frequency, Abe? Viewing the FFT

In the previous section we made a spatial filter and applied it to the image by convolution—that is, we slid our filter across the image and computed the result at each location. But we can do the same thing in the so-called "frequency domain" by first taking the Fourier Transform ('fft2') of the image, multiplying it by our filter, and then re-converting to a spatial representation using the Inverse Fourier Transform. This might seem very inefficient, but in many cases it's actually faster than the spatial method. This is because very fast algorithms exist for the Fourier Transform and multiplication is faster than convolution.

First, display the amplitude spectrum of the Lincoln image. See Step 5 in lincoln2.m for help

Believe it or not, the same information regarding Abe's image is present in the Fourier-transformed representation. We don't see all of it in this figure because we are only able to render an image of the real part of the transform, which gives us the amplitudes of the various sine waves. The imaginary part gives the phase relationships, which are critical for placing the sine waves in the correct place when reconstructing the image.

NOTE: If you'd like a more intuitive notion of what the Fourier Transform does, see fft_demo.m, which simplifies things by using a 1-dimensional signal rather than a 2-dimensional image.

Step 6: What's the frequency, Abe? Filtering an image in the Fourier domain.
Now, let's take on the next step of applying a low-pass filter in the Fourier domain.

> *Low-pass filter the Lincoln image in the frequency domain. Apply a circular window to pass only the spatial frequencies around the origin. Plot the result of this filter in the Fourier domain to make sure it is correct. Using the values that passed the filter, convert back to a recognizable image using the Inverse Fourier Transform ('ifft2').* See Step 6 in lincoln2.m for help
> *What does the resulting image look like?*

Step 7: A band-rejection filter in Fourier space.
Let's return to Marr's way of thinking about this issue. According to him, what would happen if we removed just the middle frequencies? If we did this, the zero-crossings in the mid-sized filters would no longer corroborate those in the large or the small channels. The prediction is that the visual system would attribute the two different kinds of information (i.e. that in the small and large channels) to different physical phenomena. To see if this is the case, we need a band-rejection filter. In frequency space, this would consist of an annulus (or donut) of zeros in a sea of ones.

> *Use the accompanying helper functions ('brf.m' and 'circle.m') to generate a "band rejection filter."*
> *Multiply this filter by the Fourier data and plot the resulting amplitude spectrum.*
> *Use 'ifft2' to convert the filtered data back into a spatial representation.*
> See Step 7 in lincoln2.m for help and information about the helper functions

Interesting. We have the same pretty good rendering of Abe, but he appears as if behind a screen. The high frequency noise is still there (unlike in the low-pass filtered image), but somehow it doesn't interfere with our ability to recognize Abe. *How would Marr explain this?*

Another framework in which to think about this phenomenon is that of "critical-band masking." The basic idea is that it's not just the presence of noise that matters, but also how close it is (in frequency space, in this case) to the signal. This conceptual framework is laid out nicely in a paper by Harmon and Julesz.

EXTRA Step 8: Developing an intuition about FFTs

> *Run fft2 on some of the other images that were created in this tutorial. For example, start with J, the original image, gray-scaled. Next, try different versions of that image at different 'grains' of resampling. Can you predict which images will have comparatively more power in the low frequencies? In the high frequencies?*

EXTRA Step 9: Adding noise to an image in the fourier domain

> *Building on some of the concepts we have learned, how could you inject frequency (or phase) noise into our Lincoln picture? What would this look like?*

EXTRA Step 10: Image sharpening and edge detection

If any of you are interested in digital photography, you have probable come across the concept of 'sharpening' your

digital images. In Photoshop and other programs, this is often called ‘Unsharp Mask’ because the first step actually blurs an image and then edges are detected from the blurred image. Those two images are then combined to create an image that appears shaper than the original. See Extra Step 10 in lincoln2.m for an example of this process

> *How do these results change with different amounts of blurring? What if you used LoG ‘edge’ detection instead?*

Version History:
RTB wrote it, 16 May 2003; re-worked for QMBC, June 2011
DAR modified it, August 2011 as two files. This pdf and corresponding “lincoln2.m”