An Introduction to Image Processing and the Fourier Transform

J. Christian Russ, Analytical Vision, Inc.

Abstract:

By now, whether they were aware of it or not, most people have had an opportunity to play with Image Processing. Photoshop is bundled with nigh everything, and programs such as NIH Image are free for public use. This paper discusses the fundamentals of image processing, filters, and explains some of the uses of the 2-D Fourier Transform.

This paper is intended as an introduction to grayscale image processing, for people who are moderately mathematically inclined. No great mathematical background is necessary, no nasty double-integrals are shown. Instead a conceptual relationship between spatial images (2-D arrays of pixels) and some ways of manipulating them is drawn, with a nod to the computational resources required.

As the amount of mathematical computing power increases in the hands of users, with the 68882, the 68040, the AT&T 3210, and the PowerPC 601, techniques that were previously too computationally painful or required cheap (and not terribly satisfying) work-arounds are becoming feasible and common.

We will discuss convolution, the Fourier transform, and other methods that require transcendental functions.

Matrix Operators

The process of applying a square matrix to all of the pixels in an image is called **convolution**. The process of convolution is to take a **kernel** (or square matrix of numbers – also called a **filter** or **operator**) and apply it to each pixel and its neighbors in an image. This is a dot-product of the values in the kernel with the values in the image in the following manner:

Kernel (J x J)

$$\begin{vmatrix} k_{1,1} & k_{1,2} & \dots & k_{1,j} \\ k_{2,1} & k_{2,2} & \dots & k_{2,j} \\ | & | & \backslash & | \\ k_{j,1} & k_{j,2} & \dots & k_{j,j} \end{vmatrix}$$

Image to be filtered (N x M)

$$\begin{matrix} P_{1,1} & P_{1,2} & \dots & \dots & \dots & P_{1,m} \\ P_{2,1} & P_{2,2} & \dots & \dots & \dots & P_{2,m} \\ | & | & \backslash & & & | \\ | & | & & \backslash & & | \\ P_{n,1} & P_{n,2} & \dots & \dots & \dots & P_{n,m} \end{matrix}$$

To calculate a new pixel $P_{x,y}$ the following equation would be used:

$$\text{New Px,y} = \sum_{h=1}^{j} \sum_{v=1}^{j} k_{h,v} * P_{(x - h - j/2 - 1), (y - v - j/2 -}$$

This process is graphically explained in the next section.

Unfortunately the typical values in the kernels are not in the range 0..1, and some automatic methods must be used to determine a divisor to yield a reasonable pixel value, e.g. one that fits in a byte. The best rule-of-thumb is that if the sum of the kernel values is zero then the divisor should be the sum of the positive values in the kernel, and if the sum is non-zero then that sum should be the divisor. Also, if the sum of the kernel values is zero then 128 is added to the result, because there are often negative values returned from convolution.

One of the important concepts here is that the new value cannot be immediately replaced in the image but

must either be placed into a new image, or buffered, so that the old pixel values can be used in calculations. Typically j/2 lines need to be buffered.

Another term that will be used is **spatial domain**. This is another way of saying a 2-D array of pixels in a

Cartesian coordinate system (as opposed to the Fourier domain, which is a 2-D array of amplitudes and phases - to be defined later).

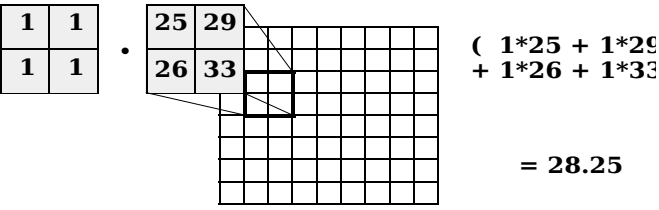
In the first few examples, a JxJ kernel will be shown in the following manner to better illustrate the process of convolution:

$k_{1,1}$	$k_{1,2}$	···	$k_{1,j}$
$k_{2,1}$	$k_{2,2}$	···	$k_{2,j}$
|	|	\	|
$k_{j,1}$	$k_{j,2}$	···	$k_{j,j}$

Image Smoothing

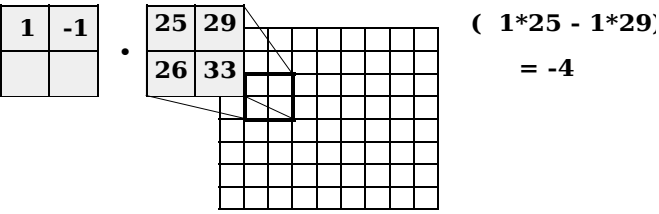
One of the simplest and most useful forms of image processing is “Smoothing” or reducing the noise in an image. This is accomplished by averaging neighboring pixels together. In this example, a pixel is averaged with its three neighbors (beneath, right, and diagonally to the right), and the result is placed into a new image.

This can be represented as a kernel that is 2x2. We are using four pixels in the average, and then dividing down by the number of pixels to produce a new value.



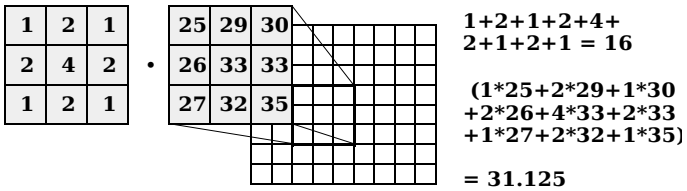
Not all values in the kernel or filter matrix have to be the same. For instance, if a 1 and -1 were used, a directional derivative could be computed (first derivative, in a specific direction), but this would yield a number that might not fit into a byte: -255..255.

This example show how a horizontal directional derivative (partial) can be computed from an image.



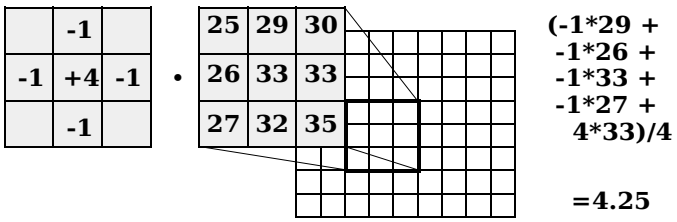
3x3 Smooth operator

In order to get around the pixel shifting problem (a result of averaging with neighbors on just two sides), we should look at **operators** (yet another word for kernels) that have odd-dimensions. Here is a common 3x3 operator that is optimized for computers that have bit-shift instructions:



Laplacian 3x3

A common edge-detector that does not shift position of the “edgeness” is called the Laplacian. It is essentially a second-derivative (rather than two orthogonal first-derivatives that we will see later in the Sobel). This one also has the advantages of being computer-optimized; all of the multiplications and divisions are bit-shifts:



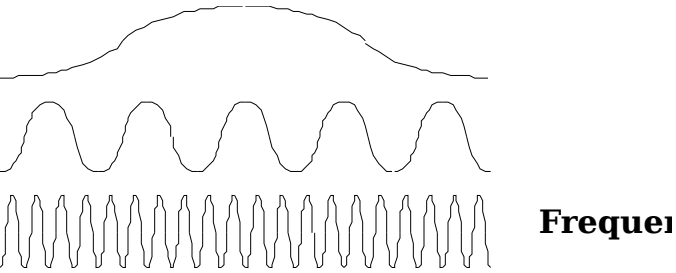
There are other kinds of operators or convolution filters that are larger, but most of them have odd dimensions to prevent pixel-shifting problems that are present in the 2x2.

Images in Fourier Space

This is intended as a qualitative description of “Fourier Space” and the spatial domain. The detailed mathematics may be found in any number of textbooks, along with sample code. Also, code from Motorola is often available for most of their processors.

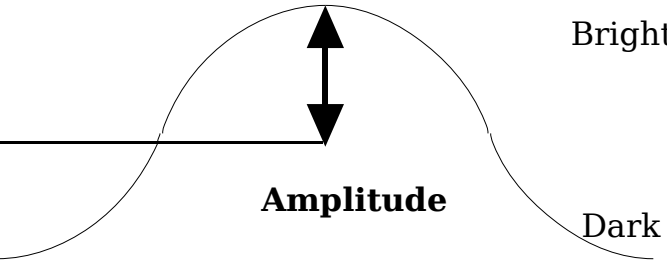
There are three concepts that are necessary to understand Fourier space. The first is **frequency**. Frequency is the inverse of wavelength (we’re using sine waves to build images – long waves have low frequencies, cover lots of pixels, and represent gradual variation of brightness, short waves have high

frequencies, take just a few pixels, and represent sharp transitions).

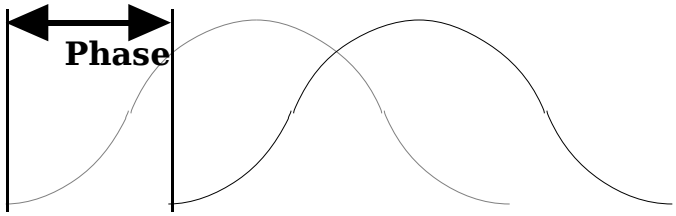


It is easier to understand the Fourier transform from basic principles. The first is frequency composition. It was discovered by von Helmholtz that any sound could be reproduced by taking a number of tuning forks and reproducing the component frequencies in that sound. Look at an equalizer on a stereo; each band consists of a range of pitches.

When a sound is played, some frequencies are present, and by reproducing those frequencies with the right **amplitude** or volume, the sound is re-created. This wave-composition (as each pitch is a sine-wave) is the basis for most audio equipment we have today. In the case of images, amplitude is the amount of change from gray (brightness 128) to the brightness that we see. A frequency with no amplitude (or **power**) will not affect the image.



The Fourier transform takes a wave or sequence of information and identifies component frequencies, using a set of sine-waves that are present in varying degrees, however it takes more than just amplitude to specify a sine-wave. It also takes **phase**. Phase in this context is really an indicator of where that sine-wave starts. It doesn't matter to the human ear, because we cannot detect when the pitch started, but when this method is applied to images it becomes important:



Now the difficult part. We take this idea of phase and amplitude to two dimensions. Instead of amplitude representing air-pressure, as it does with sound, it represents brightness. High-frequencies are short, sudden changes and low-frequencies are long, slow changes in the image. Phase is position, or where the change takes place. Let's look at the following image:

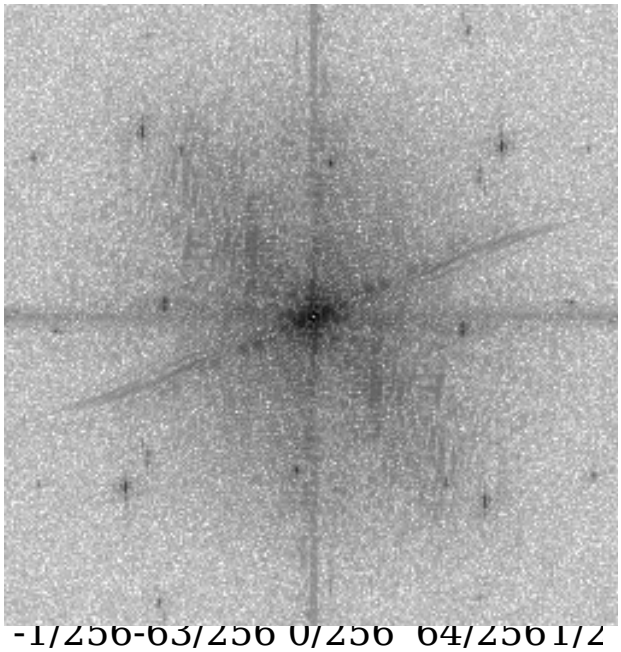
Original Scanned Image

This image has been scanned from an offset-print page. Please note the half-tone screen in the image. The half-tone dots of the image will be our principle landmarks. They are a repeating pattern at a specific angle. They will show up as large amplitudes at a specific set of frequencies.

Fourier Power Spectrum

The power-spectrum for an image is just the amplitude information for each frequency. The common method for displaying frequencies is with the zero frequency (called DC) in the center and the highest frequencies at

the outside. Here we can see a number of spikes that correspond to the halftones, and a diagonal line, 30° off of the horizontal that corresponds to the frequencies that it takes to represent the minute hand in the original picture.



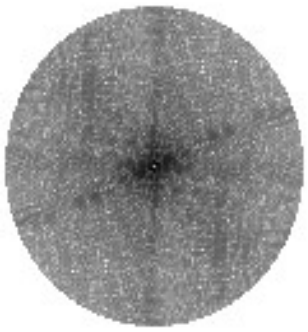
FFT Power Spectrum

Filtering in Fourier space

The first convolution filter we looked at was for smoothing. Smoothing is really a low-pass filter (low frequencies are kept, high frequencies are thrown away), and is remarkably similar to the effect of a camera out-of-focus. Here we keep just the frequencies that are lower than the half-tone dots:

Cut-off frequency =
(Half Tone Screen) / Image Width

So if the half tone or screen is 35 dpi in the image (each dot is a pixel) then there are 150 half-tone dots across the image. If the image is 256 pixels wide then the frequency cut-off should be 35 / 256. In this example, all frequencies higher than that have their amplitudes set to zero:



Center (lowest) Frequencies

and producing an image that looks like this:

Low-Pass without Half-tone screen dots

There are a number of side-effects, in this case **ringing**, which are caused by the assumption that the image repeats both horizontally and vertically. (That is to say, the image repeats off the right edge, the bottom edge, the top edge, and the left edge.) There are ways to reduce this effect, including zero-padding.

Smoothing in the Spatial Domain

The same filter as above, but performed in the spatial domain with a convolution filter looks like this. Note, that now we missed some processing on the edges, because there is no information past the edge and we cannot compute the correct result for points near the edge.

Spatial Domain - Gaussian (Low-Pass)

The kernel that was used is a Gaussian, with a standard deviation (σ) of 1.0 pixel. This means that 68% of the weight lies within 1.0 pixels of the center.

$$G(x,y) = e^{-\left(\frac{x^2+y^2}{2\pi\sigma^2}\right)}$$

0	0	1	1	1	1	1	0	0
0	1	2	3	3	3	2	1	0
1	2	3	6	7	6	3	2	1
1	3	6	9	11	9	6	3	1
1	3	7	11	12	11	7	3	1
1	3	6	9	11	9	6	3	1
1	2	3	6	7	6	3	2	1
0	1	2	3	3	3	2	1	0
0	0	1	1	1	1	1	0	0

σ = 1.0, 9x9 Gaussian smooth kernel

In order to perform this convolution, approximately 81 multiplies and 1 divide were performed for each pixel in the source image. This can be improved, if multiplication is expensive, by using 81 look-up tables of 256 entries each. The tables can be computed once at the start of processing. They can even be normalized so that division can be accomplished by dropping a byte or bit-shifting. A 512 x 512 image with one byte per pixel and this kernel would involve about 21 million multiplications, plus the occasional divide.

The Fourier method of convolution using this same kernel performs an FFT (Fast Fourier Transform) of the kernel, an FFT of the image to put the image into Fourier space, a dot-product (complex multiply in Fourier-space) and then an inverse FFT to bring the image back into the spatial domain.

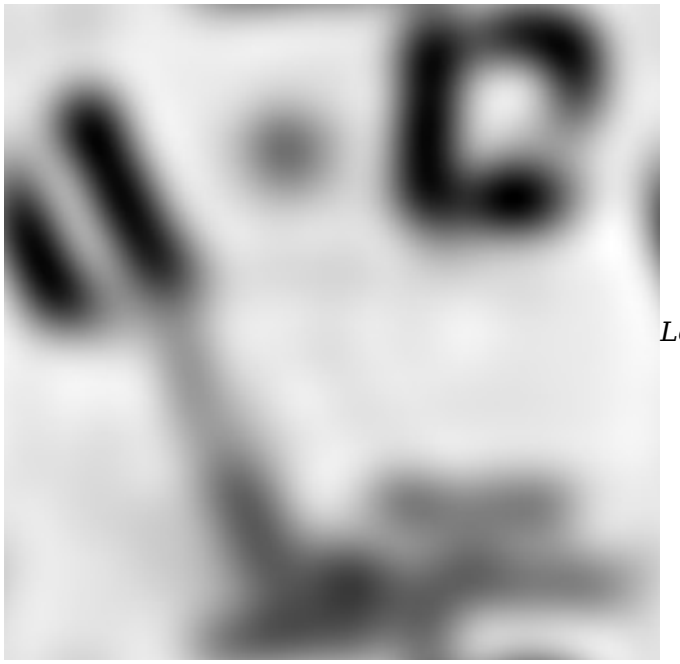
Depending upon the implementation of the FFT function, somewhere around 13x13, the FFT / convolution / IFFT process becomes more efficient than simple convolution. This is because the number of multiplies goes up with the number of elements in the kernel, which has to get much larger for successively lower frequencies. Conversely, the spatial convolution is significantly more efficient for high-frequency processing. The cut-off is approximately 9x9 with the Fast Hartley (half the number of multiplies and half the storage requirement), but Stanford has a patent on the FHT.

Enhancement in Frequency Domain

A high-pass filter keeps mostly edges. There are two ways to accomplish this in frequency space. The first is to reduce the amplitude (without changing the phase) of the lower frequencies, or conversely to increase the amplitude of the high frequencies. If you have ever seen footage of the Ed Sullivan Show or other ancient Black and White television shows, you probably have noticed a halo effect around people and objects. This is due to modern processing making the picture look better. This same effect is demonstrated in the following example.

The second method to enhance an image is to perform a low-pass filter (where the low frequencies are kept) and subtract it from the original image. Since the Gaussian is a low-pass filter, this method works in the spatial domain, too.

High-Pass

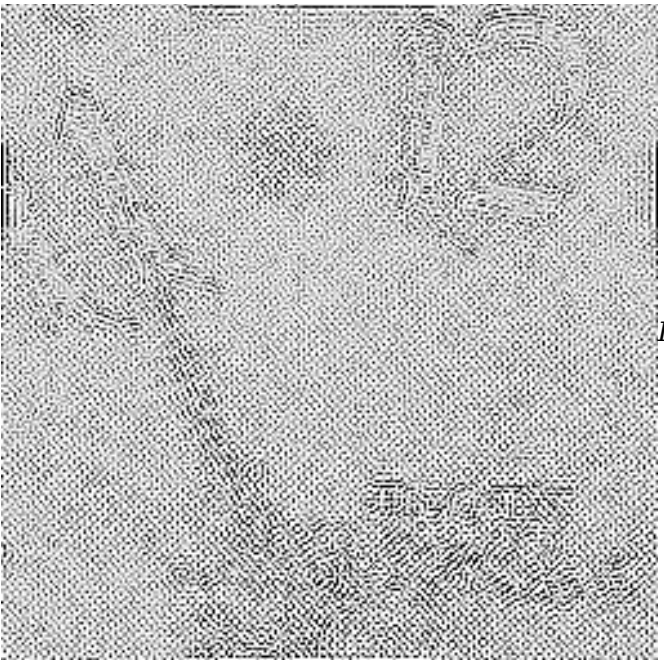


Low-Pass

A variation of this technique is auto-focusing. The cameras and microscopes that autofocus have direct control of a "Z" axis or focal position. When an image is in focus (and there are edges present in the image, as opposed to blank sky or a black lens cap) the high-frequencies are maximized. That is to say, there is greater amplitude in the higher frequencies within the image. (The edges are sharper!) Since people tend to consider the center of the image the most interesting, this method is applied with either several 1-D lines near and through the center of the image, or a transform on a small section of the center.

The low-pass image, below, does not contain much information about the image. It appears to be really out-of-focus, but this image, when added to the one above, yields the original. The frequency cut-off for these two images was approximately 1/10 of a wavelength (1/f * width = wave-length) of 26 pixels.

By selecting a very-high frequency cut-off, noise or other high-frequency artifacts can be isolated. Unless noise is a periodic function (60Hz noise from A/C) or microwave oven or something like that, it has a very high frequency. A simple choke or low-pass filter, set to a fairly high frequency can remove all sorts of stuff. Here is an example of what is removed from this image in the one to two pixel range. This is a higher frequency than the half-tone dots. It also occupies a very large portion of the frequency space image.



Highest Frequencies

Non-Linear Filters

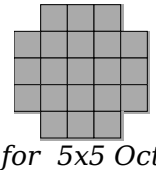
We have seen that there is a strong relationship between convolution filters and the Fourier transform. These are called linear filters. The next class of filters are very powerful and are called non-linear filters. That is to say they cannot be done in Fourier space - there are no Fourier equivalents.

Rank Filters

The first example is a median filter. It is accomplished by taking the 3x3 area around a pixel and sorting the values that are present, deriving a new image from the median value (3x3 = 9 values, median = 5th value). This can be significantly different than the average value for the area, and there are some more advanced techniques that depend upon this difference. This is excellent for reducing noise, since a noise value is likely to be either the top or bottom of the brightness range. It does have the notable characteristic that edges (but not corners) are left pretty much alone. Objects that are not noise, but are smaller than 3x3 can be obliterated. For those familiar with PhotoShop, this is called **despeckle**.

3x3 Median

A 5x5 median is the same as the 3x3, but now there are 25 values to sort and the 13th is kept. A variation on the 5x5 is called the 5x5 octagonal median. This skips the four farthest-away pixels, is a better approximation of a circle, and therefore is less sensitive to the orientation of edges and objects in the image. This can obliterate larger objects; any-thing under about 3 pixels can be wiped out entirely. Notice that the half-tone dots and thin lines on the letters are gone.



Neighborhood for 5x5 Octagonal Median

5x5 Octagonal Median

By continually re-applying the median until there are no changes in the image the image is reduced to a series of regions with common shades. It is an interesting artistic effect and is called **Posterization**.

Posterization

There are other **rank operators** besides the median. These include min, max, and range. Rather than sorting, min will keep the brightest value within a 3x3 area, and max will keep the darkest. The major problem with this is that noise is most likely to be darker or lighter than the neighboring pixels. Notice the white spots in the numbers, below:

Min (Brightest)

Max

(Darkest)

If the noise is reduced in the image, either through averaging several frames, or using high-fidelity equipment, then this yields better results. The range is the difference between the Darkest and Lightest pixels, and an image can be made from the range.

Edge Detection - Sobel

An edge is a transition from one brightness level to another. The "edgeness" or magnitude of the edge is really how steep the transition is or how rapidly the numbers change. (For those following the Fourier terminology, the steeper the edges, the higher the frequencies it takes to represent such an edge. From a calculus standpoint, the first derivative is higher on a steep edge.)

Another category of non-linear operators includes edge-detection. We've already seen the Laplacian (second derivative) filter, but it has some notorious problems. The first is that it is a wonderful noise amplifier. If a pixel varies by a little bit from its neighbors, that difference is multiplied by four.

The Sobel edge filter is one of the most interesting and powerful. It computes both the horizontal and vertical partial derivatives of the edge.

Vert. kernel

1	2	1
0	0	0
-1	-2	-1

Horiz. kernel

1	0	-1
2	0	-2
1	0	-1

These directional derivatives (partials) are combined in the equation:

$$\text{Edge Magnitude} = \text{Sqrt}\,(V^2 + H^2)$$

and a new image is generated. Because there is a low-pass component in these kernels (other kernels work, as long as they are orthogonal and have the same multipliers as each other) the edges are a bit wide.

Sobel Edge Image – Edges are dark

The biggest problem with the Sobel is that a square-root is required in computation. Common approximations are to either add the partial derivatives together or to take the maximum. With faster processors, this is not as big an issue.

Edge Detection - Kirsch

One of the filters that works around that problem is the Kirsch edge- detector. It applies eight kernels, with the merit of multiplying each pixel by -3 and by 5 (other values can be used) and then the maximum of the eight kernels is put into the new image. This eliminates the need for transcendental and produces similar results to the Sobel.

Kirsch (max of 8 directional derivatives):

5	5	5
-3	0	-3
-3	-3	-3

5	5	-3
5	0	-3
-3	-3	-3

5	-3	-3
5	0	-3
5	-3	-3

-3	-3	-3
5	0	-3
5	5	-3

$$\begin{vmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{vmatrix} \qquad \begin{vmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{vmatrix}$$

$$\begin{vmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{vmatrix} \quad \begin{vmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{vmatrix}$$

Kirsch

Edge Image – Edges are dark

Edge Detection - Frei & Chen

Another approach toward edge detection is the Frei & Chen (1977) filter. It applies nine convolution kernels to an image to create a basis function (S) for how organized the pixels are, but then uses two of the kernels (E) to measure how much of that organization is an edge instead of a line or a corner, etc. It is able to detect fainter edges, and it yields thinner lines.

The following kernels constitute the basis functions for the Frei & Chen filter:

$$b_0 = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} \quad b_1 = \begin{vmatrix} -1 & -\sqrt{2} & -1 \\ -3 & 0 & 5 \\ 1 & \sqrt{2} & 1 \end{vmatrix}$$

$$b_2 = \begin{vmatrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{vmatrix} \quad b_3 = \begin{vmatrix} 0 & -1 & \sqrt{2} \\ 1 & 0 & -1 \\ -\sqrt{2} & 1 & 0 \end{vmatrix}$$

$$b_4 = \begin{vmatrix} \sqrt{2} & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & -\sqrt{2} \end{vmatrix} \quad b_5 = \begin{vmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{vmatrix}$$

$$b_6 = \begin{vmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{vmatrix} \quad b_7 = \begin{vmatrix} 1 & -2 & -1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{vmatrix}$$

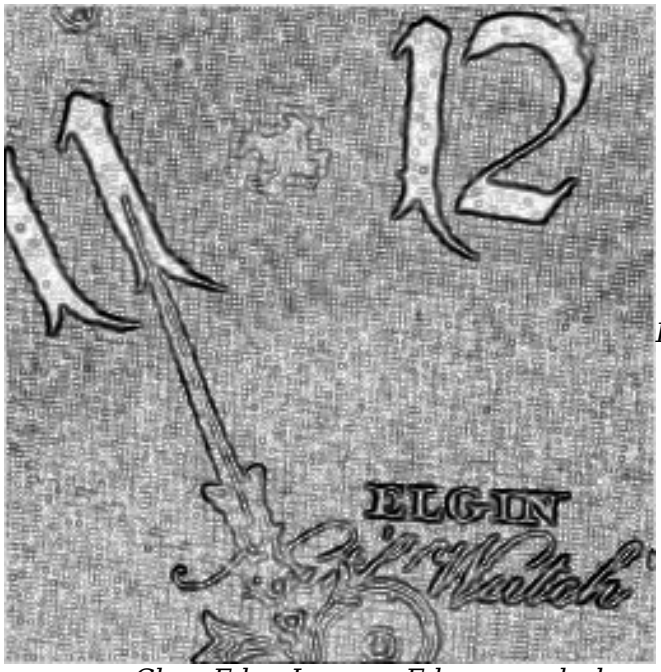
$$b_8 = \begin{vmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{vmatrix}$$

After each is applied, the two values E and S are computed as:

$$S = (b_0)^2 + (b_1)^2 + (b_2)^2 + (b_3)^2 + (b_4)^2 + (b_5)^2 + (b_6)^2 + (b_7)^2 + (b_8)^2$$

$$E = (b_1)^2 + (b_2)^2$$

$$\text{Edge Mag.} = \cos^{-1}(1 - \text{Sqrt}(E / S))$$



Frei & Chen Edge Image – Edges are dark

Image enhancement

A common technique to edge enhance an image is to average an edge image with the original. With the Laplacian this is instead performed by using the kernel:

$$\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$$

Notice the 5 in the center – this adds a fraction of the original back into the edge image.

(Again, the Fourier equivalent, for linear filters, is to de-emphasize but not eliminate the lower frequencies in the image.)

Conclusion

Image processing is a very computer intensive application. As newer processors come along, the need for DSP's and array processors (which can process a whole line of pixels at once with real-time video) are reduced and the more powerful techniques can be placed into the hands of the casual user.

Color is a whole different ball game.

References

Ballard, D. H. and C. M. Brown (1982) *Computer Vision*, Prentice-Hall, Englewood Cliffs, NJ.

Frei, W. and C. C. Chen. (1977) *Fast boundary detection: a generalization and a new algorithm.. IEEE Trans. Computers* C-26: 988-998.

Marr, David (1982) Vision, W. H. Freeman, San Francisco, CA.

Reeves, A. A. *Optimized Fast Hartley Transform with Applications in Image Processing* Thesis, Dartmouth University, March 1990.

Rosenfeld, A. and A. C. Kak (1982) *Digital Picture Processing* vol 1 & 2, Academic Press, New York, NY.

Russ, John C. (1992) *The Image Processing Handbook,* CRC Press, Boca Raton, FL.