2 Filtering

In the previous chapter, we made a simple computer vision system. We broke the image into edges, and labeled the edges, to make progress in interpreting the 3d shape of a scene. But we had to work in a very constrained world so that our brittle processing steps would give useful information about edges and their labels.

Of course, we want to build a vision system that operates in the real world. One such system is the human visual system. We have a fairly good idea of what happens at the initial stages of visual processing, and it will turn out to be similar to some of the filterings we discuss in this lecture. While we're inspired by the biology, here we seek some mathematically simple processing that will help us to parse an image into useful tokens, low-level features that will be useful later to construct visual interpretations.

We'd like for our processing to enhance image structures of use for subsequent interpretation, and to remove variability within the image that makes more difficult comparisons with previously learned visual signals. Let's proceed by invoking almost the simplest mathematical processing we can think of, and see how far it takes us toward these goals.

2.1 Linear Filtering

Perhaps the simplest filtering is linear. Each pixel of the output image is replaced by a linear combination of pixels of the input image. If horizontal and vertical positions are indexed by m and n, the output image is $f[m,n]$, and the input image is $g[m,n]$, then a general linear filtering of the image is

$$f[m,n] = \sum_{k,l} h[m,n,k,l]\, g[k,l] \tag{2.1}$$

Typically, we don't know where within the image we expect to find any given item (Fig. 2.1), so we often want to process the image in a spatially invariant manner, the same processing algorithm at every pixel. In that case, the processing becomes a *linear convolution* of the image data with some filter. The weighting, h, for the linear combination of the input image pixels, g, is only a function of the spatial offset from the pixels of g. For a 1-dimensional signal, a linear convolution, denoted $\circ$, of h and g is:

$$fm = h \circ g \tag{2.2}$$

$$= \sum_{k} h[m-k]\, g[k] \tag{2.3}$$

Figure 2.1

A fundamental property of images is translation invariance–the same image may appear at arbitrary spatial positions within the image. Image credit: [**?**].

Figure 2.2 shows the convolution of a kernel, h, with a 1-d signal, g. h and g are shown in the top row. Subsequent rows show the implementation. $hm - k$ is just h, offset by m pixels and reversed. We multiply this term-by-term with g and sum those weighted values of g[m] to form the output signal, f[m].

In two dimensions, the processing is analogous: The input filter is flipped vertically and horizontally, then slid over the image to record the inner product with the image everywhere. Mathematically, this is:

$$f[m,n] = h \circ g \tag{2.4}$$

$$= \sum_{k,l} h[m-k, n-l]\, g[k,l] \tag{2.5}$$

Figure 2.3 shows the 2-d convolution of a kernel h with an image, g. The particular kernel used in the figure averages in the vertical direction and takes differences horizontally. The output image reflects that processing, with horizontal differences accentuated and vertical changes diminished.

When implementing a convolution, one is confronted with the question of what to do at the image boundaries. There's really no satisfactory answer for how to handle the boundaries that works well for all applications. Some typical choices for the convolution output for a pixel where the kernel requires using an pixel value outside of the input image:

- set the output to zero
- set the value to that of the nearest output image pixel with valid mask inputs
- reflect the valid output image pixels over the boundary of valid output pixels.

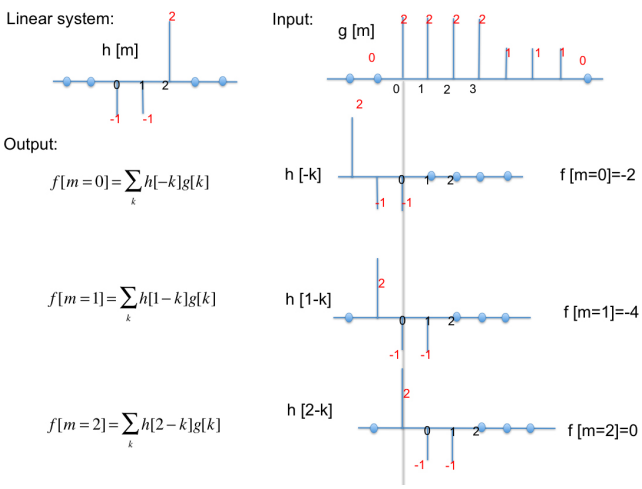


Figure 2.2

Illustration in 1-d of the steps in computing the convolution of a kernel h with a signal g. Shifted and offset versions of the kernel h provide the weights to construct f[m] from a linear combination of the samples of g[m].

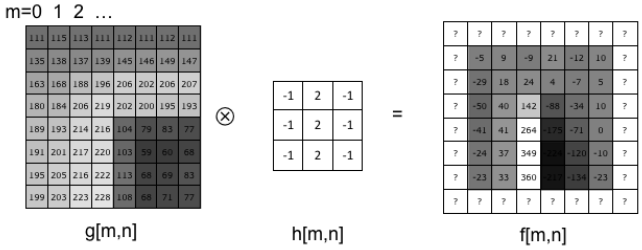


Figure 2.3

Illustration of a 2-d convolution of an input image, g, convolved with a kernel, h, giving the output image, f. The images are shown with both their pixel values and the corresponding image intensities (the assignment of intensities to numbers was rescaled for the output image, f). Border pixel values of the output image are not determined by the convolution, since the kernel would include pixel values outside of the input image.

- crop the output image to omit the border pixels

To build intuition about filter convolutions and their visual results, the following figures provide a visual dictionary of some simple convolution kernels and their output when applied to images. Figure 2.4 (a) is a warm-up: an impulse, convolved with any image, gives back that same image (even at the boundaries, by the way, since any pixels beyond the boundaries are multiplied by zero). Figure 2.4 (b) is a shift. when you take this shifted impulse, flip it, and multiply by the original image, the original image is shifted two pixels to the right.

What linear convolution will cause the image to rotate (Figure 2.4 (c))? At the center of rotation, the center pixel should be output, no matter what the surrounding pixels are, so that can only be implemented by convolution with an impulse. But at the top left corner, one wants to grab a pixel from, say, 5 pixels down and to the right, and from the bottom one needs to grab the pixel from about 5 pixels up and to the right. So this rotation operation can't be written as a spatially invariant convolution.

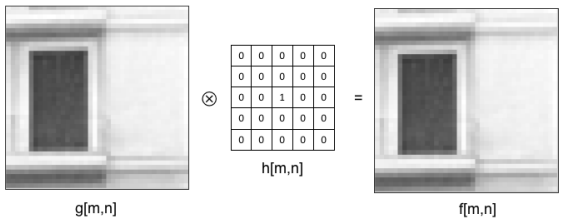
2.1.1 Kernel and convolution examples

Linear convolutions, despite their simplicity, are surprisingly useful for processing and interpreting images. It's often very useful to blur images, in preparation for subsampling or to remove noise, for example. Other useful processing includes edge enhancement and motion analysis.

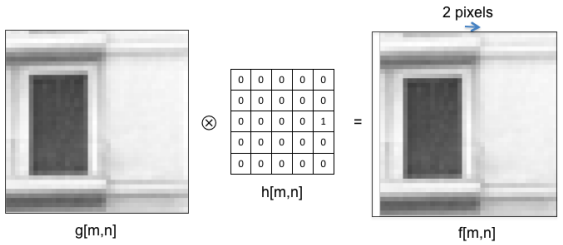
Figure 2.5 shows some blurring kernels and the filtered results. Figure 2.5 (a) shows an image convolved with a uniform, rectangular kernel–each pixel is an average of the input pixels within the rectangle.

Figure 2.5 (b) and (c) show the results of blurring in just one direction.

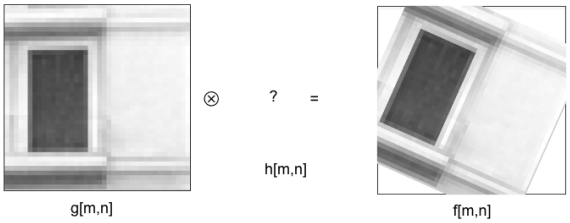
A simple way to design a *sharpening* filter is to de-emphasize the blurry components of an image. By the linearity of the convolution operator, we're allowed to add and subtract kernels to make a new kernel that would give us the same filtered image as if we had added and subtracted the filtered outputs of each of the component kernels. For this example, we start with twice the original image (sharp plus blurred parts), then subtract away the blurred components of the image. That would leave one original image in there, plus an additional component of the sharp details. The perceptual result is that of a sharpened image, Fig. 2.6.



(a)



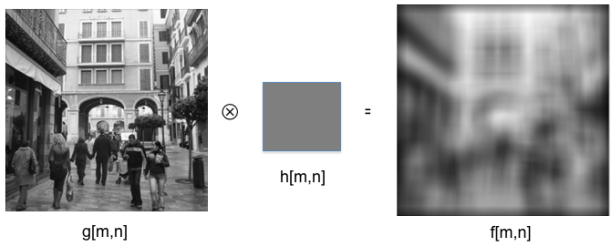
(b)



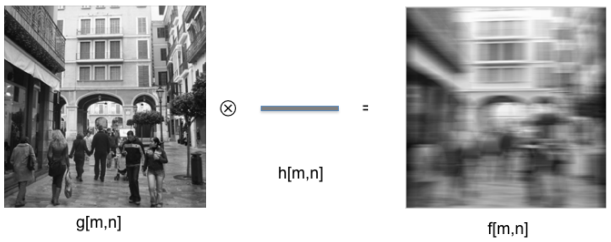
(c)

Figure 2.4

(a) An impulse convolved with the input image gives no change. (b) A shifted impulse shifts the image. (c) The text discusses why there is no space invariant convolution kernel can rotate an image.



(a)



(b)

(c)

Figure 2.5

Blurring with (a) a rectangle, and a (b) horizontal and (c) vertical line. Note the structures that are both averaged out (along the direction of blurring), and maintained (perpendicular to that direction) in each resulting image.

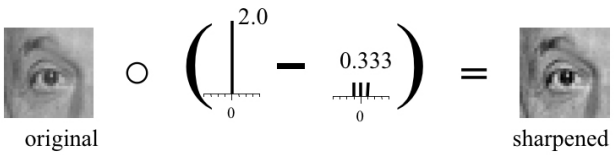


Figure 2.6

Sharpening achieved by subtraction of blurred components (the three filter taps of amplitude $\frac{1}{3}$) from the full image (scaled appropriately).

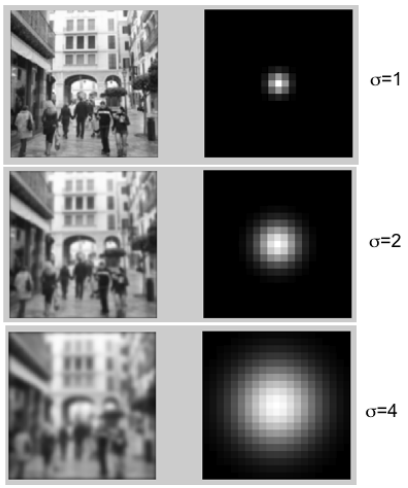


Figure 2.7

A spatially sampled approximation to the convolution of Gaussian kernels applied to an image.

One of the most useful blurring filters is a Gaussian. For a given standard deviation parameter, σ, the kernel is $G(m,n;\sigma)$:

$$G(m,n;\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{m^2+n^2}{2\sigma^2} \tag{2.6}$$

By adjusting the standard deviation, σ, of the Gaussian, it is possible to adjust the level of image detail that appears in the blurred image. Figure 2.7 shows the result of narrow and wider Gaussians applied to an image. The multi-dimensional Gaussian filter has the additional computational advantage that it can be applied as a concatenation of 1-d Gaussian filters. This can be seen by writing the 2-d Gaussian, Eq. (2.6), in the convolution equation, Eq. (2.5).

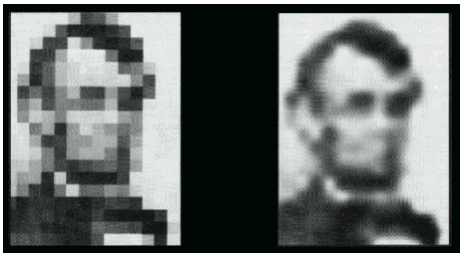


Figure 2.8

Left: input image. Right: blurred version. The left version has many spurious details introduced by the undersampling of the image. The right image has been blurred by a large Gaussian filter. ((image from http://acor.org/sgreene/hmsbeagle/html/content/17/recroom/artgalas.htm, after 1973 image by Bela Julesz and Leon Harmon)).

Letting G^x and G^y be the 1-d Gaussian convolution kernels in the horizontal and vertical directions, we have

$$\begin{aligned} G[m,n] \circ f[m,n] &= \sum_{k,l} G[m-k, n-l] f[k,l] \\ &= \sum_{k,l} \frac{1}{2\pi\sigma^2} \exp - \frac{(m-k)^2 + (n-l)^2}{2\sigma^2} \circ f[m,n] \\ &= \sum_{k} \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(m-k)^2}{2\sigma^2} \sum_{l} \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(n-l)^2}{2\sigma^2} f[k,l] \\ &= G^x \circ (G^y \circ f[m,n]) \end{aligned}$$

This can save quite a bit in computation time when applying the convolution of Eq. (2.7) . If the 2-d convolution kernel is *nxn* samples, then a direct convolution of that 2-d kernel scales in proportion to n^2, since Eq. (2.7) requires one multiplication per image position per kernel sample. Using the cascade of two 1-d kernels, resulting in an equivalent 2-d filter of the same size, scales in proportion to $2n$.

Another application of linear filtering is to remove distracting high-resolution image details. Fig. 2.8 shows a Gaussian low-pass filter applied to remove unwanted image details (the blocky artifacts) from an image.

2.2 Fourier Transform

We need a precise language to talk about the effect of linear filters, and the different image components, than saying "sharp" and "blurry" parts of the image. The Fourier transform is useful for such an analysis.

By analogy with temporal frequencies, which describe how quickly signals vary over time, a "spatial frequency" describes how quickly a signal varies over space. The Fourier transform lets us describe a signal as a sum of complex exponentials, each of a different spatial frequency.

Here's the definition of the discrete Fourier transform, and its inverse transformation.

Fourier Transform

$$F[u,v] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f[m,m] \exp -2\pi i(\frac{um}{M} + \frac{vn}{N}) \tag{2.7}$$

By applying $\frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}$ to both sides of Eq. (2.7) and exploiting the orthonormality between distinct Fourier basis elements, we find the inverse Fourier transform relation,

Inverse Fourier Transform

$$f[m,n] = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F[u,v] \exp +2\pi i(\frac{um}{M} + \frac{vn}{N}) \tag{2.8}$$

As we can see from the transform equation, we re-write the image, instead of as a sum of offset pixel values, as a sum of complex exponentials, each at a different frequency, called a spatial frequency for images, since they describe how quickly things vary across space. Of course, images are real, and thus really we're describing the image as a sum of sines and cosines, which we'll create from the complex exponentials by taking sums and differences of them, at the same amplitude. So to generate a real valued image, the Fourier transform will always have real component that is even, and an imaginary component that is odd.

From the inverse transform formula, we see that to construct an image from a Fourier transform, capital F, we just add-in the corresponding amount of that particular complex exponential (conjugated).

To get a feel for what the Fourier components indicate visually, we can look at a some positions in the Fourier transform plane, and see what those coefficients correspond to, visually.

We usually arrange the coefficients in the complex plane so that the zero frequency, or "DC", coefficient is at the center (this is different than what matlab will give you if you run FFT). Slow, large variations correspond to complex exponentials of frequencies near the origin. If the amplitudes of the complex conjugate exponentials are the same, then their sum will represent a cosine wave; if their amplitudes are opposite, it will be a sine wave. Frequencies further away from the origin represent faster variation with movement across space.

2.2.1 Fourier transform properties

Figure 2.10 shows a color visualization of the complex-valued matrix, which, when used as a multiplicand, yields the Fourier transform of 1-d vectors. Many Fourier transform properties and symmetries can be observed from inspecting that matrix.

Upon first learning about Fourier transforms, it can be such a surprise to learn that one can synthesize any image as a sum of sines and cosines. To help gain insight into how that works, it is informative to show examples of partial sums of complex exponentials. Figure 2.11 shows partial sums of the Fourier components of an image. In each partial sum of N components, we use the largest N components of the Fourier transform. Using the fact that the Fourier basis functions are orthonormal, it is straightforward to show that this is the best least squares reconstruction possible from each given number of Fourier basis components. In this example, the first 500 coefficients are sufficient for recognizing this 256x256 resolution image.

The Fourier transform coefficients are complex numbers. One might ask which is more important, the magnitude of the fourier transform, or its phase. For the global fourier transform, the magnitude of the images can often be quite similar, one to another. The phases carry the information of where the image contours are, by specifying how the phases of the sinusoids must line up in order to create the observed contours and edges.

It's useful to become adept at computing and manipulating simple Fourier transforms. Figure **??** shows a list of useful Fourier transform pairs (temporarily showing figures from Bracewell and Szeliski's books), and these are useful to study and become familiar with.

Figure 2.14 shows the 2-d Fourier transforms of some simple signals. The depicted signals all happen to be symmetric about the spatial origin. From the Fourier transform equation, one can show that real and even input signals

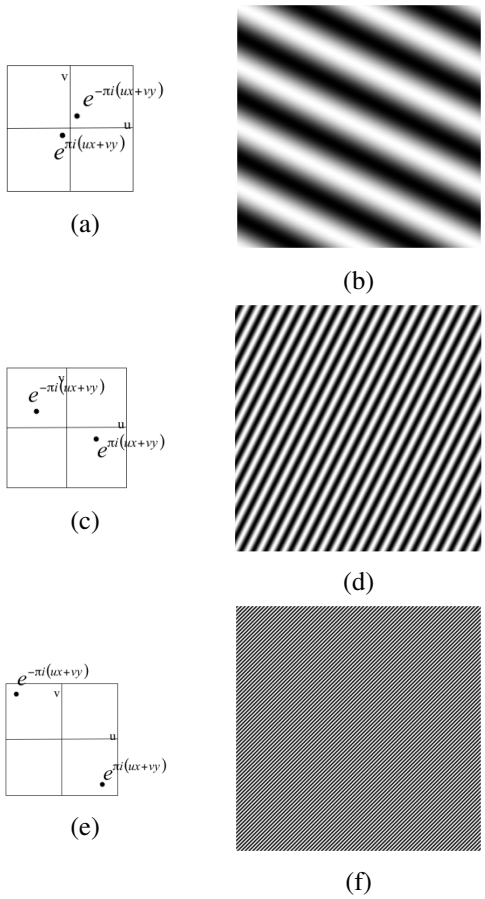


Figure 2.9

(a) To get some sense of what basis elements look like, we plot a basis element — or rather, its real part — as a function of x,y for some fixed u, v. We get a function that is constant when (ux+vy) is constant. The magnitude of the vector (u, v) gives a frequency, and its direction gives an orientation. The function is a sinusoid with this frequency along the direction, and constant perpendicular to the direction. (b)Here u and v are larger than in the previous slide. (c) And larger still...

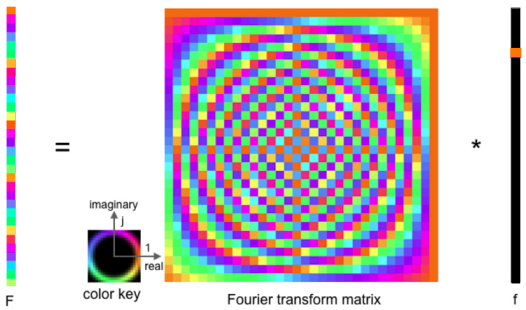


Figure 2.10

Visualization of Fourier transform as a matrix. The signal to be transformed forms the entries of the column vector at right. The complex values of the Fourier Transform matrix are indicated by the color, with the key in the bottom left. In the vector at the right, black values indicate zero.

transform to real and even outputs. So for the examples of Fig. 2.14, we only show the magnitude of the Fourier transform, which in this case is the absolute value of the real component of the transform, and the imaginary component happens to be zero for the signals we'll examine.

Based on this example, and the Fourier transform pairs of Fig. 2.14, take the following quiz: match these Fourier transform magnitudes with the corresponding images in Fig. 2.16

The Fourier transform lets us characterize images by their spatial frequency content. It's also the natural domain in which to analyze space invariant linear processes, because the Fourier bases are the eigenfunctions of all space invariant linear operators. In other words, if you start with a complex exponential, and apply any linear, space invariant operator to it, you always come out with a complex exponential of that same frequency, but, in general, with some different amplitude and phase.

Another way to state that property is through the Fourier convolution theorem, given below. Consider a function f that is the convolution of two functions, g and h:

$$f = g \circ h \tag{2.9}$$

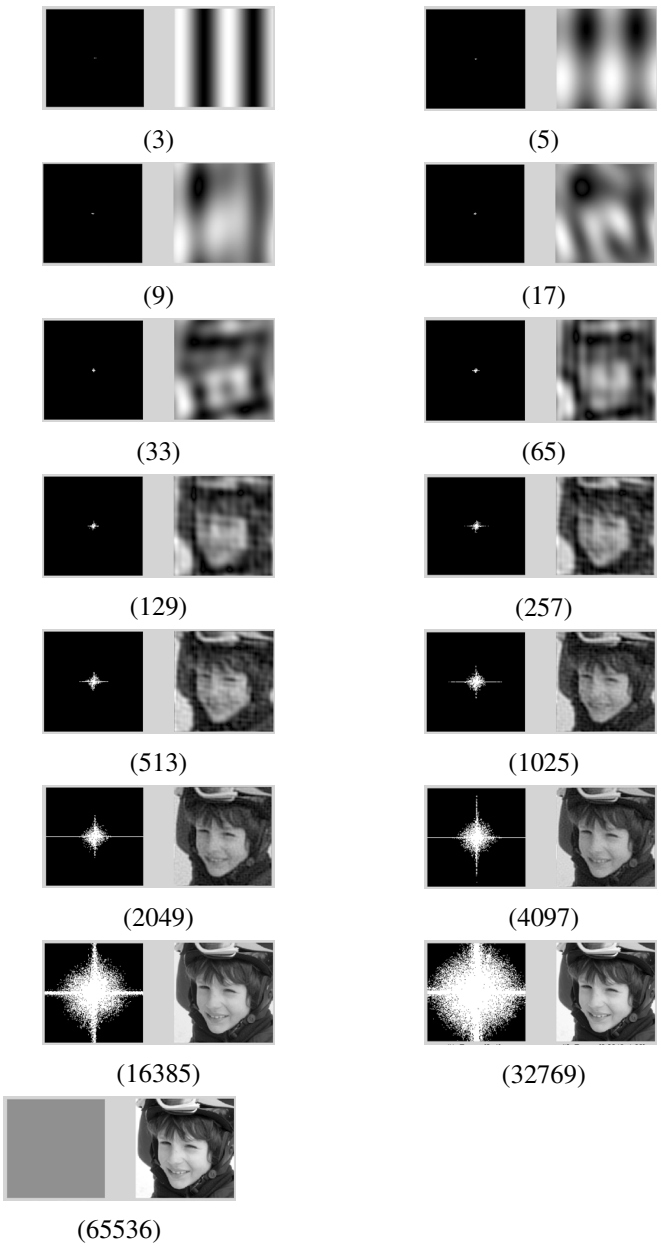


Figure 2.11

Reconstructing an image from the N Fourier coefficients of the largest amplitude. The left frame shows the location, in the Fourier domain, of the N Fourier coefficients which, when inverted, give the image at the right. Using only 1025 coefficients, the image is seen clearly.

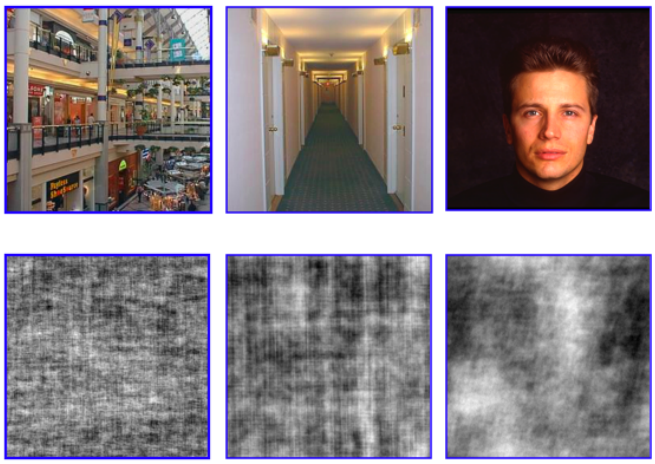


Figure 2.12

The Fourier transform is complex valued, with each spatial frequency having a magnitude and a complex phase value. The phase values of the spatial frequencies dominate in their importance of the visual perception of an image. Top row: three images. Bottom row: image resulting from randomizing the Fourier transform phase value of each image.

If we take the Fourier transform of both sides, and using the definition of the Fourier transform, we obtain

$$F[u,v] = DFT(g \circ h) \tag{2.10}$$

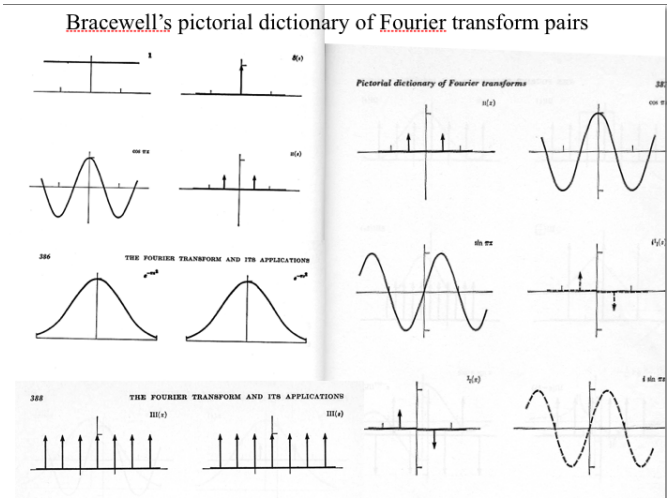
$$= \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g[m-k, n-l]\, h[k,l]\, e^{-2\pi i(\frac{um}{M} + \frac{nv}{N})} \tag{2.11}$$

Changing the dummy variables in the sums (introducing $\mu = m - k$ and $\nu = n - l$), we have

$$F[u,v] = \sum_{\mu=-k}^{M-k-1} \sum_{\nu=-l}^{N-l-1} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g[\mu,\nu]\, h[k,l]\, e^{-2\pi i(\frac{(\mu+k)u}{M} + \frac{(\nu+l)v}{N}} \tag{2.12}$$

Recognizing that the first two summations give the DFT of g, using circular boundary conditions, gives

$$F[u,v] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} Gu, v e^{2\pi i(\frac{ku}{M} + \frac{lv}{N})} h[k,l] \tag{2.13}$$



(a)

Name	Signal		Transform
impulse	$\delta(x)$	$\Leftrightarrow$	1
shifted impulse	$\delta(x-u)$	$\Leftrightarrow$	$e^{-j\omega u}$
box filter	$\text{box}(x/a)$	$\Leftrightarrow$	$a\text{sinc}(a\omega)$
tent	$\text{tent}(x/a)$	$\Leftrightarrow$	$a\text{sinc}^2(a\omega)$
Gaussian	$G(x;\sigma)$	$\Leftrightarrow$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega;\sigma^{-1})$
Laplacian of Gaussian	$(\frac{x^2}{\sigma^4}-\frac{1}{\sigma^2})G(x;\sigma)$	$\Leftrightarrow$	$-\frac{\sqrt{2\pi}}{\sigma}\omega^2 G(\omega;\sigma^{-1})$
Gabor	$\cos(\omega_0 x)G(x;\sigma)$	$\Leftrightarrow$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega\pm\omega_0;\sigma^{-1})$
unsharp mask	$(1+\gamma)\delta(x)$ $-\gamma G(x;\sigma)$	$\Leftrightarrow$	$(1+\gamma)-$ $\frac{\sqrt{2\pi}\gamma}{\sigma}G(\omega;\sigma^{-1})$
windowed sinc	$\text{rcos}(x/(aW))$ $\text{sinc}(x/a)$	$\Leftrightarrow$	(see Figure 3.29)

Szeliski, Computer Vision, 2010

Table 3.2 Some useful (continuous) Fourier transform pairs: The dashed line in the Fourier

(b)

Figure 2.13
(a) and (b): A collection of useful Fourier transform pairs, from [**?**] and [**?**].

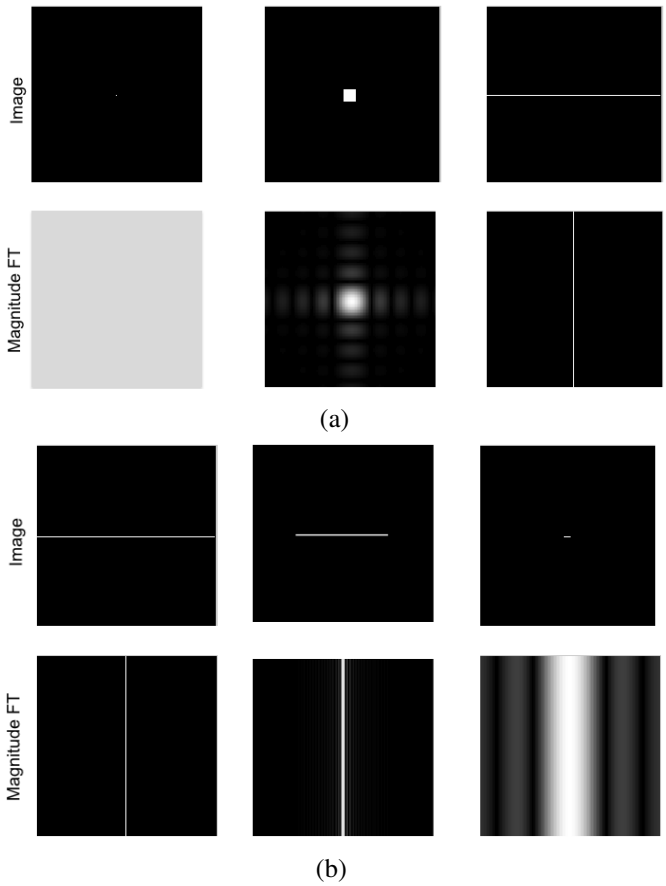
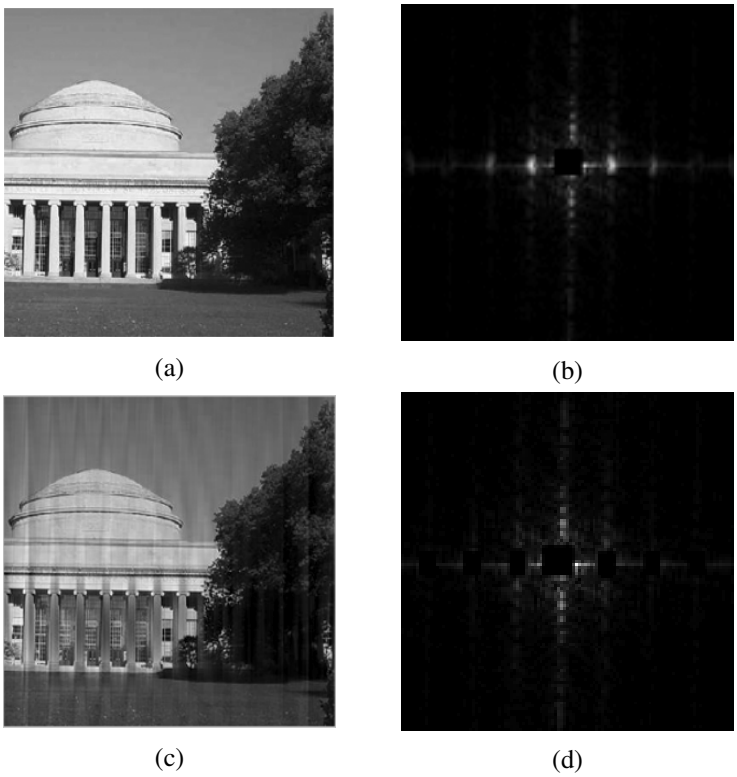


Figure 2.14

(a) and (b): Some two-dimensional Fourier transform pairs. Note the trends visible in the collection of transform pairs: As the support of the image in one domain gets larger, the magnitude in the other domain becomes more localized. A line transforms to a line oriented perpendicularly to the first.



(a) (b)

(c) (d)

Figure 2.15

Simple filtering in the Fourier domain. (a) The repeated columns of the building of the MIT dome generate harmonics along a horizontal line in the Fourier domain. (b) By zeroing out those Fourier components, the columns of the building are substantially removed.

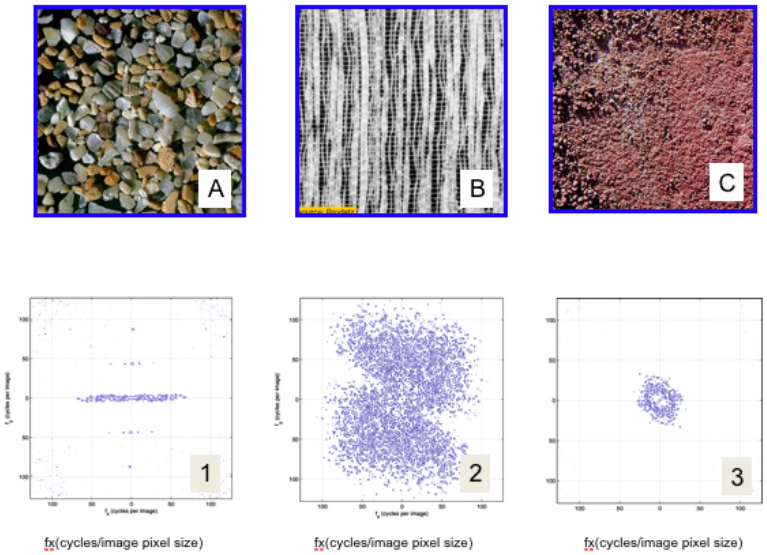


Figure 2.16
Match the image to the corresponding plot of the log of the magnitude of its Fourier tranform.

Performing the DFT indicated by the second two summations gives the desired result,

$$F[u,v] = G[u,v]H[u,v] \tag{2.14}$$

Thus, the operation of a convolution, in the Fourier domain, is just a multiplication of the Fourier transform of each term in the Fourier domain.

This property lets us examine the operation of a filter on any image by examining how it modulates the fourier coefficients of any image. This lets us make precise our coarse description of, say, how sharpening works. So let's just revisit that one example.

2.2.2 Sampling and Aliasing

2.2.2.1 shah function and its Fourier transform

2.2.2.2 Nyquist frequency

Figure 2.17.

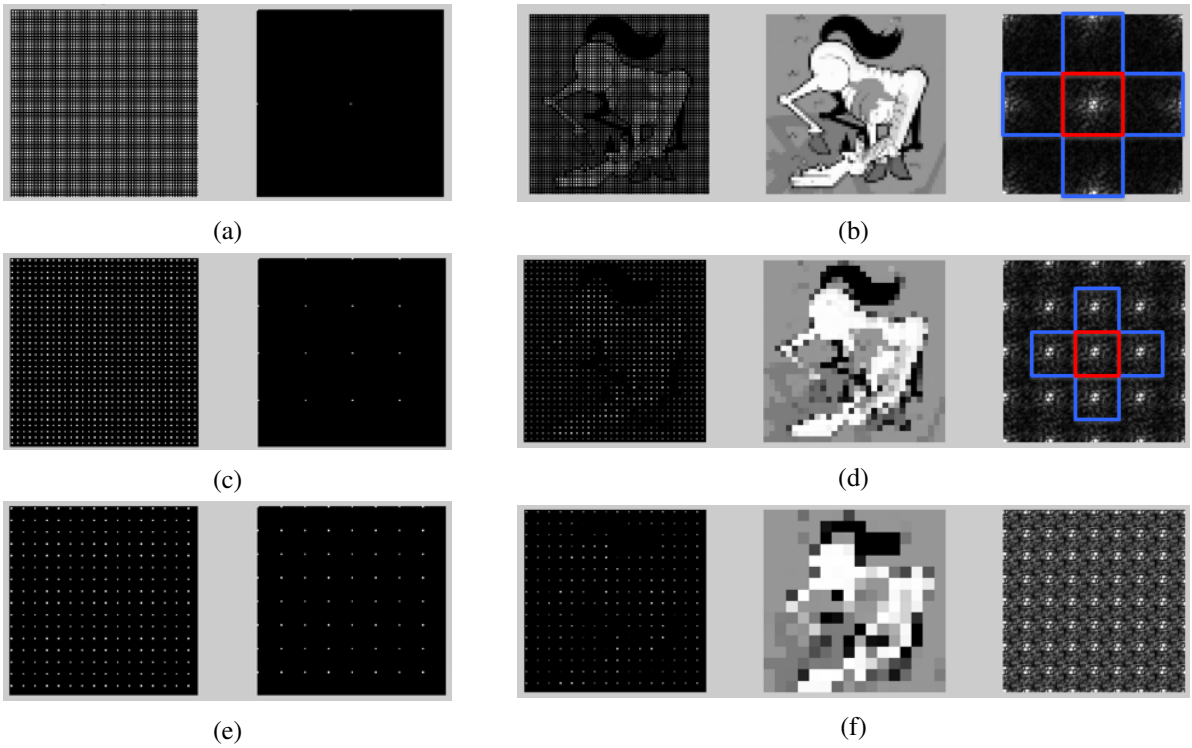


Figure 2.17

Aliasing examples. (a) - (f) Far left column: spatial sampling pattern. 2nd column: Fourier transform of that spatial pattern, revealing replication locations of the Fourier transform spectrum of the subsampled image. The subsampled image is shown in the 3rd column. Zeroing out all but the central replication of the image spectrum (far right), yields the interpolated images of the 4th column.

Bibliography