Digital Image Processing

By

Sandeep Gupta

Apache Technologies Pvt. Ltd.

Contents

Digital Image Definitions

A digital image $a[m, n]$ described in a 2D discrete space is derived from an analog image $a(x, y)$ in a 2D continuous space through a *sampling* process that is frequently referred to as digitization. The mathematics of that sampling process will be described in Section 5. For now we will look at some basic definitions associated with the digital image. The effect of digitization is shown in Figure 1.
The 2D continuous image $a(x, y)$ is divided into N *rows* and M *columns*. The intersection of a row and a column is termed a *pixel*. The value assigned to the integer coordinates $[m, n]$ with $\{m=0,1,2,...,M-1\}$ and $\{n=0,1,2,...,N-1\}$ is $a[m, n]$. In fact, in most cases $a(x, y)$--which we might consider to be the physical signal that impinges on the face of a 2D sensor--is actually a function of many variables including depth (z), color (λ), and time (t). Unless otherwise stated, we will consider the case of 2D, monochromatic, static images in this chapter.

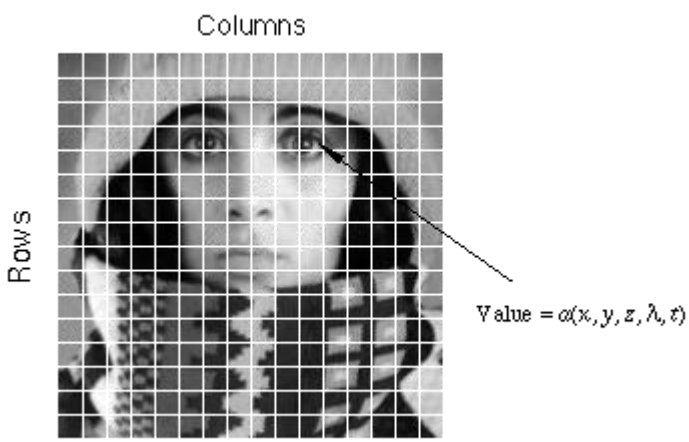


Figure 1: Digitization of a continuous image. The pixel at coordinates $[m=10, n=3]$ has the integer brightness value 110.

The image shown in Figure 1 has been divided into N = 16 rows and M = 16 columns. The value assigned to every pixel is the average brightness in the pixel rounded to the nearest integer value. The process of representing the amplitude of the 2D signal at a given coordinate as an integer value with L different gray levels is usually referred to as amplitude quantization or simply *quantization*.

Common Values

There are standard values for the various parameters encountered in digital image processing. These values can be caused by video standards, by algorithmic requirements, or by the desire to keep digital circuitry simple. Table 1 gives some commonly encountered values.

Parameter	*Symbol*	*Typical values*
Rows	N	256,512,525,625,1024,1035
Columns	M	256,512,768,1024,1320
Gray Levels	L	2,64,256,1024,4096,16384

Table 1: Common values of digital image parameters

Quite frequently we see cases of $M=N=2^K$ where $\{K = 8,9,10\}$. This can be motivated by digital circuitry or by the use of certain algorithms such as the (fast) Fourier transform (see Section 3.3).
The number of distinct gray levels is usually a power of 2, that is, $L=2^B$ where B is the number of bits in the binary representation of the brightness levels. When $B>1$ we speak of a *gray-level image*; when $B=1$ we speak of a *binary image*. In a binary image there are just two gray levels which can be referred to, for example, as "black" and "white" or "0" and "1".

Characteristics of Image Operations

There is a variety of ways to classify and characterize image operations. The reason for doing so is to understand what type of results we might expect to achieve with a given type of operation or what might be the computational burden associated with a given operation.

Types of operations

The types of operations that can be applied to digital images to transform an input image $a[m, n]$ into an output image $b[m, n]$ (or another representation) can be classified into three categories as shown in Table 2.

Operation	Characterization	Generic Complexity/Pixel
* *Point*	- The output value at a specific coordinate is dependent only on the input value at that same coordinate.	*Constant*
* *Local*	- The output value at a specific coordinate is dependent on the input values in the *neighborhood* of that same coordinate.	P^2
* *Global*	- The output value at a specific coordinate is dependent on all the values in the input image.	N^2

Table 2: Types of image operations.

Image size = N x N; neighborhood size = P x P. Note that the complexity is specified in operations *per pixel*. This is shown graphically in Figure 2.

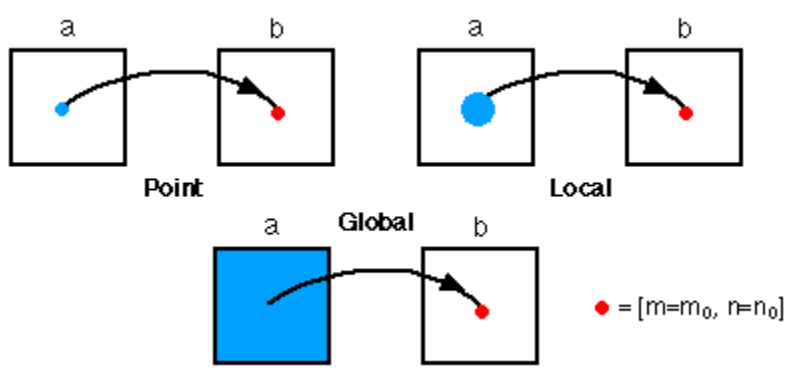


Figure 2: Illustration of various types of image operations

Types of neighborhoods

Neighborhood operations play a key role in modern digital image processing. It is therefore important to understand how images can be sampled and how that relates to the various neighborhoods that can be used to process an image.

* Rectangular sampling - In most cases, images are sampled by laying a rectangular grid over an image as illustrated in Figure 1. This results in the type of sampling shown in Figure 3ab.

* Hexagonal sampling - An alternative sampling scheme is shown in Figure 3c and is termed hexagonal sampling.

Both sampling schemes have been studied extensively and both represent a possible periodic tiling of the continuous image space. We will restrict our attention, however, to only rectangular sampling, as it remains, due to hardware and software considerations, the method of choice.

Local operations produce an output pixel value $b[m=m_o, n=n_o]$ based upon the pixel values in the *neighborhood* of $a[m=m_o, n=n_o]$. Some of the most common neighborhoods are the 4-connected neighborhood and the 8-connected neighborhood in the case of rectangular sampling and the 6-connected neighborhood in the case of hexagonal sampling illustrated in Figure 3.

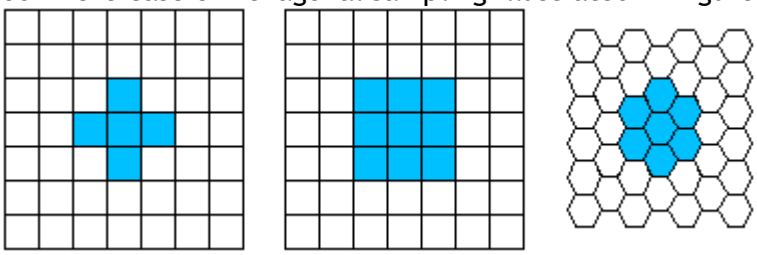


Figure 3a Figure 3b Figure 3c

Rectangular sampling Rectangular sampling hexagonal sampling 4-connected 8-connected 6-connected

Video Parameters

We do not propose to describe the processing of dynamically changing images in this introduction. It is appropriate--given that many static images are derived from video cameras and frame grabbers--to mention the standards that are associated with the three standard video schemes that are currently in worldwide use - NTSC, PAL, and SECAM. This information is summarized in Table 3.

Standard	**NTSC**	**PAL**	**SECAM**
Property			
Images / second	29.97	25	25
Ms / image	33.37	40.0	40.0
Lines / image	525	625	625
(horiz./vert.) = Aspect ratio	4:3	4:3	4:3
Interlace	2:1	2:1	2:1
Us / line	63.56	64.00	64.00

Table 3: Standard video parameters

In an interlaced image the odd numbered lines (1,3,5...) are scanned in half of the allotted time (e.g. 20 ms in PAL) and the even numbered lines (2,4,6...) are scanned in the remaining half. The image display must be coordinated with this scanning format. (See Section 8.2.) The reason for interlacing the scan lines of a video image is to reduce the perception of flicker in a displayed image. If one is planning to use images that have been scanned from an interlaced video source, it is important to know if the two half-images have been appropriately "shuffled" by the digitization hardware or if that should be implemented in software. Further, the analysis of moving objects requires special care with interlaced video to avoid "zigzag" edges.
The number of rows (N) from a video source generally corresponds one-to-one with lines in the video image. The number of columns, however, depends on the nature of the electronics that is used to digitize the image. Different frame grabbers for the same video camera might produce M = 384, 512, or 768 columns (pixels) per line.

Image Sampling

Converting from a continuous image *a*(*x*, *y*) to its digital representation *b*[*m*, *n*] requires the process of sampling. In the ideal sampling system *a*(*x*, *y*) is multiplied by an ideal 2D impulse train:

$$b_{ideal}[m.n] = a(x,y) \bullet \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \delta(x - mX_o, y - nY_o)$$

$$= \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} a(mX_o, nY_o)\delta(x - mX_o, y - nY_o)$$

Where X_o and Y_o are the sampling distances or intervals, *d*(*,*) is the ideal impulse function, and we have used equation. (At some point, of course, the impulse function *d*(*x*, *y*) is converted to the discrete impulse function *d*[*m*, *n*].)
***Square sampling* implies that $X_o = Y_o$.**

Sampling with an impulse function corresponds to sampling with an infinitesimally small point. This, however, does not correspond to the usual situation as illustrated in Figure 1. To take the effects of a *finite* sampling aperture *p*(*x*, *y*) into account, we can modify the sampling model as follows:

$$b[m,n] = \left(a(x,y) \otimes p(x,y)\right) \bullet \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \delta(x - mX_o, y - nY_o)$$

The combined effect of the aperture and sampling are best understood by examining the Fourier domain representation.

$$B(\Omega, \Psi) = \frac{1}{4\pi^2} \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} A(\Omega - m\Omega_s, \Psi - n\Psi_s) \bullet P(\Omega - m\Omega_s, \Psi - n\Psi_s)$$

Where $\Omega_s = 2\pi / X_o$ is the sampling frequency in the *x* direction and $\Psi_s = 2\pi / Y_o$ is the sampling frequency in the *y* direction. The aperture *p*(*x*, *y*) is frequently square, circular, or Gaussian with the associated $P(\Omega, \Psi)$. (See Table 4.) The periodic nature of the spectrum, described in from equation.

Sampling Density for Image Processing

Sampling aperture
To prevent the possible *aliasing* (overlapping) of spectral terms that is inherent in equation two conditions must hold:

** Bandlimited A(u,v) -*

$$|A(u,v)| \equiv 0 \quad \text{for} \quad |u| > u_c \quad \text{and} \quad |v| > v_c$$

** Nyquist sampling frequency -*

$$\Omega_s > 2 \bullet u_c \quad \text{and} \quad \Psi_s > 2 \bullet v_c$$

Where u_c and v_c are the *cutoff frequencies* in the *x* and *y* direction, respectively. Images that are acquired through lenses that are circularly symmetric, aberration-free, and diffraction-limited will, in general, be band limited. The lens acts as a low pass filter with a cutoff frequency in the frequency domain (eq.) given by:

$$u_c = v_c = \frac{2NA}{\lambda}$$

Where *NA* is the numerical aperture of the lens and λ is the shortest wavelength of light used with the lens. If the lens does not meet one or more of these assumptions then it will still be band limited but at lower cutoff frequencies than those given in equation. When working with the F-number (*F*) of the optics instead of the *NA* and in air (with *index of refraction* = 1.0), equation becomes:

$$u_c = v_c = \frac{2}{\lambda}\left(\frac{1}{\sqrt{4F^2+1}}\right)$$

Sampling aperture

The aperture $p(x, y)$ described above will have only a marginal effect on the final signal if the two conditions equation and are satisfied. Given, for example, the distance between samples X_o equals Y_o and a sampling aperture that is not wider than X_o, the effect on the overall spectrum-due to the $A(u, v)P(u, v)$ behavior implied by equation-is illustrated in Figure 16 for square and Gaussian apertures.
The spectra are evaluated along one axis of the 2D Fourier transform. The Gaussian aperture in Figure 16 has a width such that the sampling interval X_o contains +/-3σ (99.7%) of the Gaussian. The rectangular apertures have a width such that one occupies 95% of the sampling interval and the other occupies 50% of the sampling interval. The 95% width translates to a *fill factor* of 90% and the 50% width to a *fill factor* of 25%. The *fill factor* is discussed in Section 7.5.2.

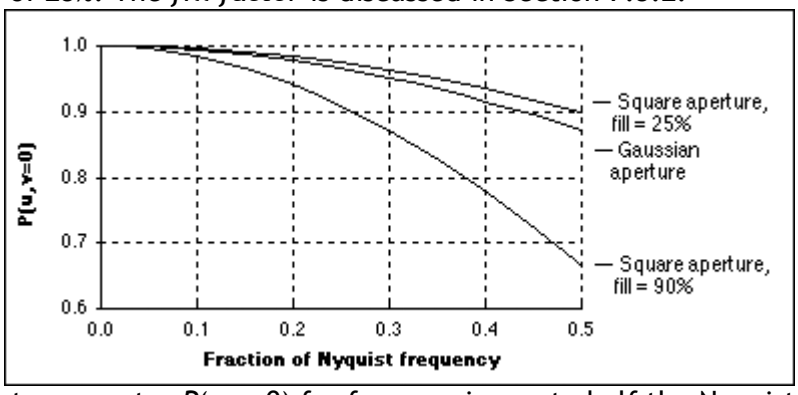


Figure 16: Aperture spectra $P(u,v=0)$ for frequencies up to half the Nyquist frequency. For explanation of "fill" see text.

Derivative-based Operations

Just as **smoothing** is a fundamental operation in image processing so is the ability to take one or more **spatial derivatives** of the image.
The fundamental problem is that, according to the mathematical definition of a derivative, this cannot be done.
A digitized image is not a continuous function *a*(*x*, *y*) of the spatial variables but rather a discrete function *a*[*m*, *n*] of the integer spatial coordinates. As a result the algorithms we will present can only be seen as ***approximations*** to the true spatial derivatives of the original spatially continuous image.
Further, as we can see from the Fourier property in equation, taking a derivative multiplies the signal spectrum by either *u* or *v*. This means that high frequency noise will be emphasized in the resulting image. The general solution to this problem is to combine the derivative operation with one that suppresses high frequency noise, in short, smoothing in combination with the desired derivative operation.

First Derivatives

As an image is a function of two (or more) variables it is necessary to define the direction in which the derivative is taken. For the two-dimensional case we have the horizontal direction, the vertical direction, or an arbitrary direction, which can be considered as a combination of the two. If we use $\mathbf{h}_x$ to denote a horizontal derivative filter (matrix), $\mathbf{h}_y$ to denote a vertical derivative filter (matrix), and $\mathbf{h}^{\vartheta}$ to denote the arbitrary angle derivative filter (matrix), then:

$$[\mathbf{h}_\theta] = \cos\theta \bullet [\mathbf{h}_x] + \sin\theta \bullet [\mathbf{h}_y]$$

* ***Gradient filters*** - It is also possible to generate a vector derivative description as the *gradient*, ∇ *a*[*m*, *n*], of an image:

$$\nabla a = \frac{\partial a}{\partial x}\vec{i}_x + \frac{\partial a}{\partial y}\vec{i}_y = (h_x \otimes a)\vec{i}_x + (h_y \otimes a)\vec{i}_y$$

Where $\vec{i}_x$ and $\vec{i}_y$ are unit vectors in the horizontal and vertical direction respectively.

This leads to two descriptions:

Gradient magnitude - $|\nabla a| = \sqrt{(h_x \otimes a)^2 + (h_y \otimes a)^2}$
and

Gradient direction - $\psi(\nabla a) = \arctan\left\{ (h_y \otimes a) / (h_x \otimes a) \right\}$

The gradient magnitude is sometimes approximated by:

Approx. Gradient magnitude - $|\nabla a| \cong |h_x \otimes a| + |h_y \otimes a|$

The final results of these calculations depend strongly on the choices of $\mathbf{h}_x$ and $\mathbf{h}_y$. A number of possible choices for ($\mathbf{h}_x$, $\mathbf{h}_y$) will now be described.

* ***Basic derivative filters*** - These filters are specified by:

$$i)\ \ [\mathbf{h}_x] = [\mathbf{h}_y]^t = [1 \quad -1]$$

$$ii)\ \ [\mathbf{h}_x] = [\mathbf{h}_y]^t = [1 \quad 0 \quad -1]$$

Where "^{t}" denotes matrix transpose. These two filters differ significantly in their Fourier magnitude and Fourier phase characteristics. For the frequency range 0 <= Ω <= π, these are given by:

$$i)\ \ [\mathbf{h}] = [1 \quad -1] \quad \overset{F}{\leftrightarrow} \quad |H(\Omega)| = 2|\sin(\Omega/2)|; \quad \varphi(\Omega) = (\pi - \Omega)/2$$

$$ii)\ \ [\mathbf{h}] = [1 \quad 0 \quad -1] \quad \overset{F}{\leftrightarrow} \quad |H(\Omega)| = 2|\sin \Omega|; \quad \varphi(\Omega) = \pi/2$$

The second form (*ii*) gives suppression of high frequency terms (Ω ~ π) while the first form (*i*) does not. The first form leads to a phase shift; the second form does not.

* ***Prewitt gradient filters*** - These filters are specified by:

$$[\mathbf{h}_x] = \frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \bullet [1 \quad 0 \quad -1]$$

$$[\mathbf{h}_y] = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \bullet [1 \quad 1 \quad 1]$$

Both $\mathbf{h}_x$ and $\mathbf{h}_y$ are separable. Beyond the computational implications are the implications for the analysis of the filter. Each filter takes the derivative in one direction using eq. *ii* and smoothes in the orthogonal direction using a one-dimensional version of a *uniform* filter as described in Section 9.4.1.

* ***Sobel gradient filters*** - These filters are specified by:

$$[\mathbf{h}_x] = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \bullet [1 \quad 0 \quad -1]$$

$$[\mathbf{h}_y] = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \bullet [1 \quad 2 \quad 1]$$

Again, $\mathbf{h}_x$ and $\mathbf{h}_y$ are separable. Each filter takes the derivative in one direction using eq. *ii* and smoothes in the orthogonal direction using a one-dimensional version of a *triangular* filter as described in Section 9.4.1.

* ***Alternative gradient filters*** - The variety of techniques available from one-dimensional signal processing for the design of digital filters offers us powerful tools for designing one-dimensional versions of $\mathbf{h}_x$ and $\mathbf{h}_y$. Using the **Parks-McClellan filter design algorithm**, for example, we can choose the frequency bands where we want the derivative to be taken and the frequency bands where we want the noise to be suppressed. The algorithm will then produce a real, odd filter with a minimum length that meets the specifications.

As an example, if we want a filter that has derivative characteristics in a pass band (with weight 1.0) in the frequency range 0.0 <= Ω <= 0.3π and a stop band (with weight 3.0) in the range 0.32π <= Ω <= π, then the algorithm produces the following optimized seven sample filter:

$$[\mathbf{h}_x] = [\mathbf{h}_y]^t = \frac{1}{16348}[-3571 \quad 8212 \quad -15580 \quad 0 \quad 15580 \quad -8212 \quad 3571]$$

The gradient can then be calculated as in equation

* ***Gaussian gradient filters*** - In modern digital image processing one of the most common techniques is to use a Gaussian filter (see Section 9.4.1) to accomplish the required smoothing and one of the derivatives listed in eq. . Thus, we might first apply the recursive Gaussian in eq. followed by eq. *ii* to achieve the desired, smoothed derivative filters $\mathbf{h}_x$ and $\mathbf{h}_y$. Further, for computational efficiency, we can combine these two steps as:

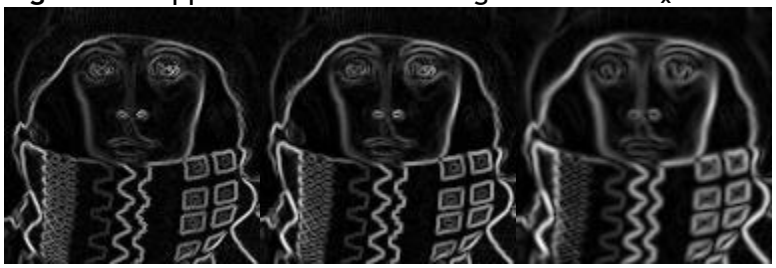
$$w[n] = \left(\frac{B}{2}\right)\left(a[n+1] - a[n-1]\right) + \left(b_1 w[n-1] + b_2 w[n-2] + b_3 w[n-3]\right) / b_0$$

$$c[n] = Bw[n] + \left(b_1 c[n+1] + b_2 c[n+2] + b_3 c[n+3]\right) / b_0$$

Where the various coefficients are defined in eq. . The first (forward) equation is applied from n = 0 *up* to n = N - 1 while the second (backward) equation is applied from n = N - 1 *down* to n = 0.

* ***Summary*** - Examples of the effect of various *derivative algorithms* on a noisy version of Figure 30a (*SNR* = 29 dB) are shown in Figure 31a-c. The effect of various *magnitude gradient algorithms* on Figure 30a is shown in Figure 32a-c. After processing, all images are contrast stretched as in eq. for display purposes.

(a) (b) (c) Simple Derivative - eq. *ii* Sobel - eq. Gaussian (σ =1.5) & eq. *ii*
Figure 31: Application of various algorithms for $\mathbf{h}_x$ - the horizontal derivative.



(a) (b) (c) Simple Derivative - eq. *ii* Sobel - eq. Gaussian (σ =1.5) & eq. *ii*
Figure 32: Various algorithms for the magnitude gradient, $|\nabla a|$.
The magnitude gradient takes on large values where there are strong edges in the image. Appropriate choice of σ in the Gaussian-based derivative (Figure 31c) or gradient (Figure 32c) permits computation of virtually any of the other forms - simple, Prewitt, Sobel, etc. In that sense, the Gaussian derivative represents a superset of derivative filters.

Second Derivatives

It is, of course, possible to compute higher-order derivatives of functions of two variables. In image processing, as we shall see in Sections 10.2.1 and 10.3.2, the second derivatives or Laplacian play an important role.

The **Laplacian** is defined as:

$$\nabla^2 a = \frac{\partial^2 a}{\partial x^2} + \frac{\partial^2 a}{\partial y^2} = (h_{2x} \otimes a) + (h_{2y} \otimes a)$$

Where $\mathbf{h}_{2x}$ and $\mathbf{h}_{2y}$ are second derivative filters. In the frequency domain we have for the

Laplacian filter (from eq.):

$$\nabla^2 a \quad \overset{F}{\leftrightarrow} \quad -(u^2 + v^2)A(u,v)$$

The transfer function of a Laplacian corresponds to a parabola $(u,v) = -(u^2 + v^2)$.

* ***Basic second derivative filter*** - This filter is specified by:

$$[\mathbf{h}_{2x}] = [\mathbf{h}_{2y}]^t = [1 \quad -2 \quad 1]$$

and the frequency spectrum of this filter, in each direction, is given by:

$$H(\Omega) = F\{1 \quad -2 \quad 1\} = -2(1 - \cos\Omega)$$

Over the frequency range $-\pi <= \Omega <= \pi$. The two, one-dimensional filters can be used in the manner suggested by eq. or combined into one, two-dimensional filter as:

$$[\mathbf{h}] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and used as in eq. .

* ***Frequency domain Laplacian*** - This filter is the implementation of the general recipe given in eq. and for the Laplacian filter takes the form:

$$c[m,n] = F^{-1}\{-(\Omega^2 + \Psi^2)A(\Omega, \Psi)\}$$

* ***Gaussian second derivative filter*** - This is the straightforward extension of the Gaussian first derivative filter described above and can be applied independently in each dimension. We first apply

Gaussian smoothing with a σ chosen on the basis of the problem specification. We then apply the desired second derivative filter eq. or eq. . Again there is the choice among the various Gaussian smoothing algorithms.
For efficiency, we can use the recursive implementation and combine the two steps--smoothing and derivative operation--as follows:

$$w[n] = B(a[n] - a[n-1]) + (b_1 w[n-1] + b_2 w[n-2] + b_3 w[n-3]) / b_0$$

$$c[n] = B(w[n+1] - w[n]) + (b_1 c[n+1] + b_2 c[n+2] + b_3 c[n+3]) / b_0$$

Where the various coefficients are defined in eq. . Again, the first (forward) equation is applied from $n = 0$ *up* to $n = N - 1$ while the second (backward) equation is applied from $n = N - 1$ *down* to $n = 0$.

* ***Alternative Laplacian filters*** - Again one-dimensional digital filter design techniques offer us powerful methods to create filters that are optimized for a specific problem. Using the Parks-McClellan design algorithm, we can choose the frequency bands where we want the second derivative to be taken and the frequency bands where we want the noise to be suppressed. The algorithm will then produce a real, even filter with a minimum length that meets the specifications.
As an example, if we want a filter that has second derivative characteristics in a pass band (with weight 1.0) in the frequency range 0.0 <= Ω <= 0.3π and a stop band (with weight 3.0) in the range 0.32π <= Ω <= π, then the algorithm produces the following optimized seven sample filter:

$$[\mathbf{h}_x] = [\mathbf{h}_y]^t = \frac{1}{11043}[-3448 \quad 10145 \quad 1495 \quad -16383 \quad 1495 \quad 10145 \quad -3448]$$

The Laplacian can then be calculated as in eq.

* ***SDGD filter*** - A filter that is especially useful in edge finding and object measurement is the *Second-Derivative-in-the-Gradient-Direction* (*SDGD*) filter. This filter uses five partial derivatives:

$$A_{xx} = \frac{\partial^2 a}{\partial x^2} \quad A_{xy} = \frac{\partial^2 a}{\partial x \partial y} \quad A_x = \frac{\partial a}{\partial x}$$

$$A_{yx} = \frac{\partial^2 a}{\partial x \partial y} \quad A_{yy} = \frac{\partial^2 a}{\partial y^2} \quad A_y = \frac{\partial a}{\partial y}$$

Note that $A_{xy} = A_{yx}$ which accounts for the five derivatives.
This *SDGD* combines the different partial derivatives as follows:

$$SDGD(a) = \frac{A_{xx} A_x^2 + 2 A_{xy} A_x A_y + A_{yy} A_y^2}{A_x^2 + A_y^2}$$

As one might expect, the large number of derivatives involved in this filter implies that noise suppression is important and that Gaussian derivative filters--both first and second order--are highly recommended if not required. It is also necessary that the first and second derivative filters have essentially the same pass bands and stop bands. This means that if the first derivative filter h_{1x} is given by [1 0 -1] (eq. *ii*) then the second derivative filter should be given by $h_{1x} \otimes h_{1x} = h_{2x}$ = [1 0 -2 0 1].

* ***Summary*** - The effects of the various second derivative filters are illustrated in Figure 33a-e. All images were contrast stretched for display purposes using eq. and the parameters 1% and 99%.

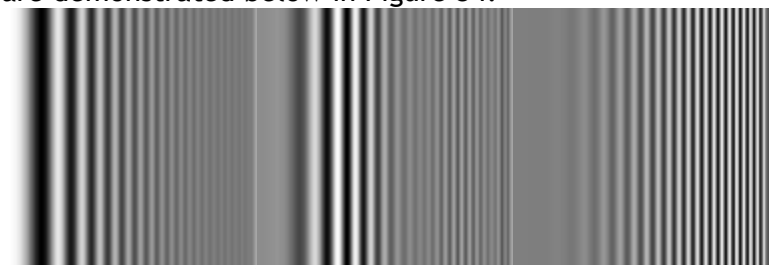
(a) (b) (c) Laplacian - eq. Fourier parabola - eq. Gaussian (σ =1.0) & eq.

(d) (e) "Designer" - eq. *SDGD* (σ =1.0) - eq.

Figure 33: Various algorithms for the Laplacian and Laplacian-related filters.

Other Filters

An infinite number of filters, both linear and non-linear, are possible for image processing. It is therefore impossible to describe more than the basic types in this section. The description of others can be found be in the reference literature (see Section 11) as well as in the applications literature. It is important to use a small consistent set of test images that are relevant to the application area to understand the effect of a given filter or class of filters. The effect of filters on images can be frequently understood by the use of images that have pronounced regions of varying sizes to visualize the effect on edges or by the use of test patterns such as sinusoidal sweeps to visualize the effects in the frequency domain. The former have been used above (Figures 21, 23, and 30-33) and the latter are demonstrated below in Figure 34.



(a) Lowpass filter **(b)** Bandpass filter **(c)** ighpass filter

Figure 34: Various convolution algorithms applied to sinusoidal test image.

Segmentation

In the analysis of the objects in images it is essential that we can distinguish between the **objects of interest** and "the rest." This latter group is also referred to as the background. The techniques that are used to find the objects of interest are usually referred to as ***segmentation techniques* - segmenting the foreground from background**. In this section we will two of the most common techniques, ***thresholding*** and ***edge finding***. And we will present techniques for improving the quality of the segmentation result. It is important to understand that:
* There is no universally applicable segmentation technique that will work for all images, and,
* No segmentation technique is perfect.

Thresholding

This technique is based upon a simple concept. A parameter ϑ called the brightness threshold is chosen and applied to the image a[m,n] as follows:

$$\textbf{If} \quad a[m,n] \geq \theta \qquad a[m,n] = object \ = 1$$
$$\textbf{Else} \qquad a[m,n] = background \quad = 0$$

This version of the algorithm assumes that we are interested in light objects on a dark background. For dark objects on a light background we would use:

$$\textbf{If} \quad a[m,n] < \theta \qquad a[m,n] = object \ = 1$$
$$\textbf{Else} \qquad a[m,n] = background \quad = 0$$

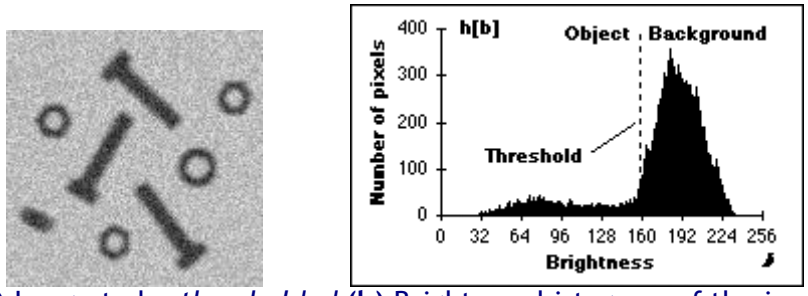
The output is the label "object" or "background" which, due to its dichotomous nature, can be represented as a Boolean variable "1" or "0". In principle, the test condition could be based upon some other property than simple brightness (for example, *If* (*Redness* {*a* [*m*, *n*]} >= ϑ_{red}), but the concept is clear.

The central question in thresholding then becomes: how do we choose the threshold ϑ? While there is no universal procedure for threshold selection that is guaranteed to work on all images, there are a variety of alternatives.

* ***Fixed threshold*** - One alternative is to use a threshold that is chosen independently of the image data. If it is known that one is dealing with very high-contrast images where the objects are very dark and the background is homogeneous (Section 10.1) and very light, then a constant threshold of 128 on a scale of 0 to 255 might be sufficiently accurate. By accuracy we mean that the number of falsely classified pixels should be kept to a minimum.

* ***Histogram-derived thresholds*** - In most cases the threshold is chosen from the brightness histogram of the region or image that we wish to segment. (See Sections 3.5.2 and 9.1.) An image and its associated brightness histogram are shown in Figure 51.
A variety of techniques have been devised to automatically choose a threshold starting from the gray-value histogram, $\{h\ [b] \mid b = 0, 1, \ldots, 2^B\text{-}1\}$. Some of the most common ones are presented below. Many of these algorithms can benefit from a smoothing of the raw histogram data to remove small fluctuations but the smoothing algorithm must not shift the peak positions. This translates into a zero-phase smoothing algorithm given below where typical values for *W* are 3 or 5:

$$h_{smooth}[b] = \frac{1}{W} \sum_{w=-(W-1)/2}^{(W-1)/2} h_{raw}[b-w] \qquad W \text{ odd}$$



(a) Image to be *thresholded* **(b)** Brightness histogram of the image

Figure 51: Pixels below the threshold ($a[m,n] < \vartheta$) will be labeled as object pixels; those above the threshold will be labeled as background pixels.

* ***Isodata algorithm*** - This iterative technique for choosing a threshold was developed by Ridler and Calvard. The histogram is initially segmented into two parts using a starting threshold value such as $\vartheta_0 = 2^{B-1}$, half the maximum dynamic range. The sample mean ($m_{f,0}$) of the gray values associated with the foreground pixels and the sample mean ($m_{b,0}$) of the gray values associated with the background pixels are computed. A new threshold value ϑ_1 is now computed as the average of these two sample means. The process is repeated, based upon the new threshold, until the threshold value does not change any more. In formula:

$$\theta_k = \left(m_{f,k-1} + m_{b,k-1}\right)/2 \;\; until \;\; \theta_k = \theta_{k-1}$$

* ***Background-symmetry algorithm*** - This technique assumes a distinct and dominant peak for the background that is symmetric about its maximum. The technique can benefit from smoothing as described in eq. The maximum peak (*maxp*) is found by searching for the maximum value in the histogram. The algorithm then searches on the *non-object pixel side* of that maximum to find a *p*% point as in eq. (39).

In Figure 51b, where the object pixels are located to the *left* of the background peak at brightness 183, this means searching to the right of that peak to locate, as an example, the 95% value. At this brightness value, 5% of the pixels lie to the *right* (are above) that value. This occurs at brightness 216 in Figure 51b. Because of the assumed symmetry, we use as a threshold a displacement to the *left* of the maximum that is equal to the displacement to the right where the *p*% is found. For Figure 51b this means a threshold value given by 183 - (216 - 183) = 150. In formula:

$$\theta = maxp - \left(p\% - maxp\right)$$

This technique can be adapted easily to the case where we have light objects on a dark, dominant background. Further, it can be used if the object peak dominates and we have reason to assume that the brightness distribution around the object peak is symmetric. An additional variation on this symmetry theme is to use an estimate of the sample standard deviation (*s* in eq. (37)) based on one side of the dominant peak and then use a threshold based on $\vartheta = maxp$ +/- 1.96*s* (at the 5% level) or $\vartheta = maxp$ +/- 2.57*s* (at the 1% level). The choice of "+" or "-" depends on which direction from *maxp* is being defined as the object/background threshold. Should the distributions be approximately Gaussian around *maxp*, then the values 1.96 and 2.57 will, in fact, correspond to the 5% and 1 % level.

* ***Triangle algorithm*** - This technique due to Zack [36] is illustrated in Figure 52. A line is constructed between the maximum of the histogram at brightness b_{max} and the lowest value b_{min} = (p=0)% in the image. The distance **d** between the line and the histogram $h[b]$ is computed for all values of b from $b = b_{min}$ to $b = b_{max}$. The brightness value b_o where the distance between h $[b_o]$ and the line is maximal is the threshold value, that is, $\vartheta = b_o$. This technique is particularly effective when the object pixels produce a weak peak in the histogram.

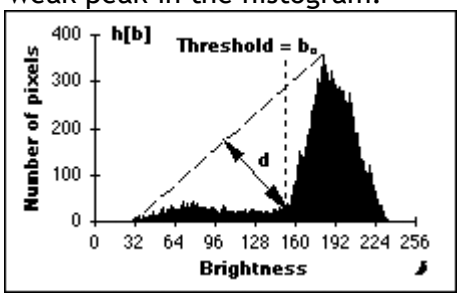


Figure 52: The triangle algorithm is based on finding the value of b that gives the maximum distance **d**.

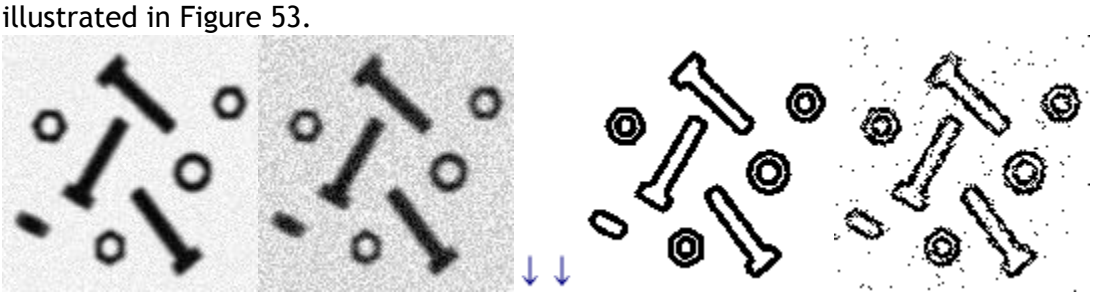
The three procedures described above give the values ϑ = 139 for the Isodata algorithm, ϑ = 150 for the background symmetry algorithm at the 5% level, and ϑ = 152 for the triangle algorithm for the image in Figure 51a.

Thresholding does not have to be applied to entire images but can be used on a region-by-region basis. Chow and Kaneko developed a variation in which the M x N image is divided into non-overlapping regions. In each region a threshold is calculated and the resulting threshold values are put together (interpolated) to form a thresholding surface for the entire image. The regions should be of "reasonable" size so that there are a sufficient number of pixels in each region to make an estimate of the histogram and the threshold. The utility of this procedure--like so many others--depends on the application at hand.

Edge finding

Thresholding produces a segmentation that yields all the pixels that, in principle, belong to the object or objects of interest in an image. An alternative to this is to find those pixels that belong to the borders of the objects. Techniques that are directed to this goal are termed *edge*-finding *techniques*. From our discussion, in Section 9.6, on mathematical morphology, specifically eqs. And, we see that there is an intimate relationship between edges and regions.

* ***Gradient-based procedure*** - The central challenge to edge finding techniques is to find procedures that produce *closed* contours around the objects of interest. For objects of particularly high *SNR*, and this can be achieved by calculating the gradient and then using a suitable threshold. This is illustrated in Figure 53.



(a) *SNR* = 30 dB **(b)** *SNR* = 20 dB
Figure 53: Edge finding based on the Sobel gradient, eq. (110), combined with the Isodata thresholding algorithm eq.

While the technique works well for the 30 dB image in Figure 53a, it fails to provide an accurate determination of those pixels associated with the object edges for the 20 dB images in Figure 53b. A variety of smoothing techniques as described in Section 9.4 and in eq. can be used to reduce the noise effects before the gradient operator is applied.

* ***Zero-crossing based procedure*** - A more modern view to handling the problem of edges in noisy images is to use the zero crossings generated in the Laplacian of an image (Section 9.5.2). The rationale starts from the model of an ideal edge, a step function that has been blurred by an *OTF*, such as Table 4 T.3 (out-of-focus), T.5 (diffraction-limited), or T.6 (general model) to produce the result shown in Figure 54.

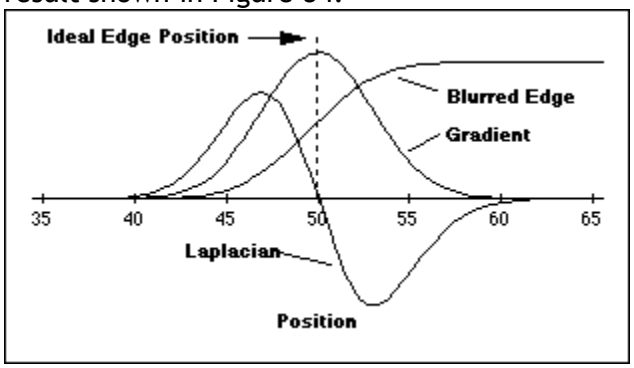


Figure 54: Edge finding based on the zero crossing as determined by the second derivative, the Laplacian. The curves are not to scale.

The edge location is, according to the model, at that place in the image where the Laplacian changes sign, the zero crossing. As the Laplacian operation involves a second derivative, this means a potential enhancement of noise in the image at high spatial frequencies; see eq. (114). To prevent enhanced noise from dominating the search for zero crossings, a smoothing is necessary.

The appropriate smoothing filter, from among the many possibilities described in Section 9.4, should according to Canny have the following properties:
* In the frequency domain, (u,v) or (Ω, Ψ), the filter should be as narrow as possible to provide suppression of high frequency noise, and;
* In the spatial domain, (x,y) or $[m,n]$, the filter should be as narrow as possible to provide good localization of the edge. A too wide filter generates uncertainty as to precisely where, within the filter width, the edge is located.

The smoothing filter that simultaneously satisfies both these properties--minimum bandwidth and minimum spatial width--is the Gaussian filter described in Section 9.4. This means that the image should be smoothed with a Gaussian of an appropriate σ followed by application of the Laplacian. In formula:

$$ZeroCrossing\{a(x,y)\} = \{(x,y) | \nabla^2 \{g_{2D}(x,y) \otimes a(x,y)\} = 0\}$$

where $g_{2D}(x,y)$ is defined in eq. (93). The derivative operation is linear and shift-invariant as defined in eqs. (85) and (86). This means that the order of the operators can be exchanged (eq. (4)) or

combined into one single filter (eq. (5)). This second approach leads to the Marr-ildreth formulation of the "Laplacian-of-Gaussians" (*LoG*) filter:

$$ZeroCrossing\{a(x,y)\} = \{(x,y) | LoG(x,y) \otimes a(x,y) = 0\}$$

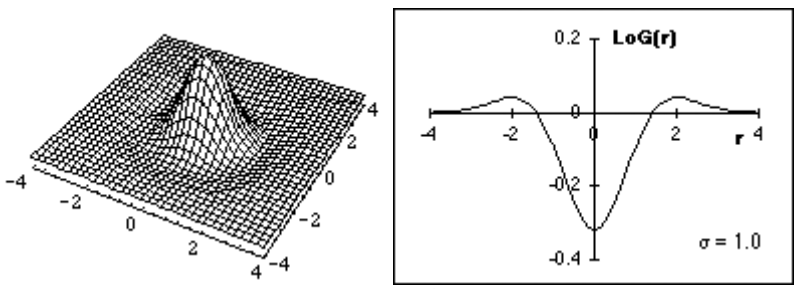
Where

$$LoG(x,y) = \frac{x^2+y^2}{\sigma^4} g_{2D}(x,y) - \frac{2}{\sigma^2} g_{2D}(x,y)$$

Given the circular symmetry this can also be written as:

$$LoG(r) = \left(\frac{r^2 - 2\sigma^2}{2\pi\sigma^6}\right) e^{-(r^2/2\sigma^2)}$$

This two-dimensional convolution kernel, which is sometimes referred to as a "Mexican hat filter", is illustrated in Figure 55.



(a) -*LoG*(*x*,*y*) **(b)** *LoG*(*r*)

Figure 55: *LoG* filter with σ = 1.0.

**PLUS-based procedure* - Among the zero crossing procedures for edge detection, perhaps the most accurate is the *PLUS* filter as developed by Verbeek and Van Vliet . The filter is defined, using eqs. (121) And (122), as:

$$PLUS(a) = SDGD(a) + Laplace(a)$$

$$= \left(\frac{A_{xx}A_x^2 + 2A_{xy}A_xA_y + A_{yy}A_y^2}{A_x^2 + A_y^2}\right) + \left(A_{xx} + A_{yy}\right)$$

Neither the derivation of the *PLUS*'s properties nor an evaluation of its accuracy are within the scope of this section. Suffice it to say that, for positively curved edges in gray value images, the Laplacian-based zero crossing procedure *overestimates* the position of the edge and the *SDGD*-based procedure *underestimates* the position. This is true in both two-dimensional and three-dimensional images with an error on the order of $(\sigma/R)^2$ where *R* is the radius of curvature of the edge. The *PLUS* operator has an error on the order of $(\sigma/R)^4$ if the image is sampled at, at least, 3x the usual Nyquist sampling frequency as in eq. (56) *or* if we choose σ >= 2.7 and sample at the usual Nyquist frequency.

All of the methods based on zero crossings in the Laplacian must be able to distinguish between **zero *crossings*** and **zero *values***. While the former represent edge positions, the latter can be generated by regions that are no more complex than bilinear surfaces, that is, $a(x,y) = a_0 + a_1*x + a_2*y + a_3*x*y$. To distinguish between these two situations, we first find the zero crossing positions and label them as "1" and all other pixels as "0". We then multiply the resulting image by a measure of the *edge strength* at each pixel. There are various measures for the edge strength that is all based on the gradient as described in Section 9.5.1 and eq. This last possibility, use of a morphological gradient as an edge strength measure, was first described by Lee, aralick, and Shapiro and is particularly

effective. After multiplication the image is then thresholded (as above) to produce the final result. The procedure is thus as follows:

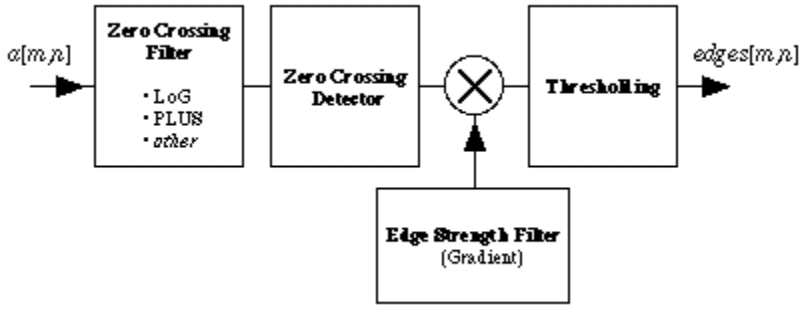


Figure 56: General strategy for edges based on zero crossings.

The results of these two edge finding techniques based on zero crossings, *LoG* filtering and *PLUS* filtering, are shown in Figure 57 for images with a 20 dB *SNR*.

a) Image *SNR* = 20 dB ↑↓ **b)** *LoG* filter ↑↓ **c)** *PLUS* filter ↑↓

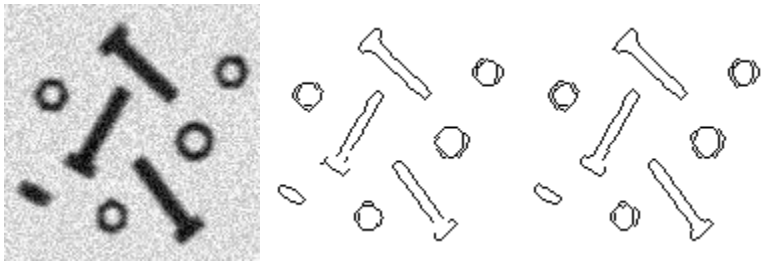


Figure 57: Edge finding using zero crossing algorithms *LoG* and *PLUS*. In both algorithms σ = 1.5.

Edge finding techniques provide, as the name suggests, an image that contains a collection of edge pixels. Should the edge pixels correspond to objects, as opposed to say simple lines in the image, and then a region-filling technique such as eq. may be required to provide the complete objects.

Binary mathematical morphology

The various algorithms that we have described for mathematical morphology in Section 9.6 can be put together to form powerful techniques for the processing of binary images and gray level images. As binary images frequently result from segmentation processes on gray level images, the morphological processing of the binary result permits the improvement of the segmentation result.

* ***Salt-or-pepper filtering*** - Segmentation procedures frequently result in isolated "1" pixels in a "0" neighborhood (salt) or isolated "0" pixels in a "1" neighborhood (pepper). The appropriate

neighborhood definition must be chosen as in Figure 3. Using the lookup table formulation for Boolean operations in a 3 x 3 neighborhood that was described in association with Figure 43, *salt filtering* and *pepper filtering* are straightforward to implement. We weight the different positions in the 3 x 3 neighborhood as follows:

$$Weights = \begin{bmatrix} w_4 = 16 & w_3 = 8 & w_2 = 4 \\ w_5 = 32 & w_0 = 1 & w_1 = 2 \\ w_6 = 64 & w_7 = 128 & w_8 = 256 \end{bmatrix}$$

For a 3 x 3 window in *a*[*m*,*n*] with values "0" or "1" we then compute:

$$\begin{aligned} sum \;=\; & w_0 a[m,n] + w_1 a[m+1,n] + w_2 a[m+1,n-1] + \\ & w_3 a[m,n-1] + w_4 a[m-1,n-1] + w_5 a[m-1,n] + \\ & w_6 a[m-1,n+1] + w_7 a[m,n+1] + w_8 a[m+1,n-1] \end{aligned}$$

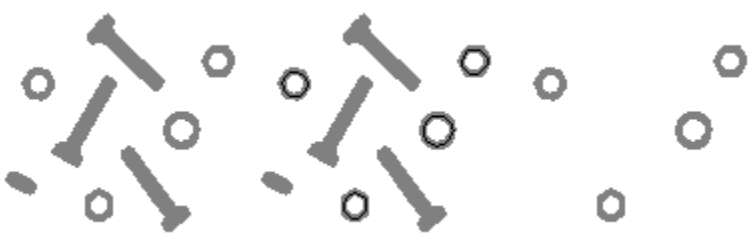
The result, *sum*, is a number bounded by 0 <= *sum* <= 511.

* ***Salt Filter*** - The 4-connected and 8-connected versions of this filter are the same and are given by the following procedure:
Compute *sum*.
If ((*sum* == 1) *c* [*m*,*n*] = 0 **Else** *c*[*m*,*n*] = *a*[*m*,*n*].

* ***Pepper Filter*** - The 4-connected and 8-connected versions of this filter are the following procedures:
4-connected 8-connected i) Compute *sum* i) Compute *sum* ii) **If** ((*sum* == 170) ii) **If** ((*sum* == 510) *c*[*m*,*n*] = 1 *c*[*m*,*n*] = 1 **Else Else** *c*[*m*,*n*] = *a*[*m*,*n*] *c*[*m*,*n*] = *a*[*m*,*n*]

* ***Isolate objects with holes*** - To find objects with holes we can use the following procedure, which is illustrated in Figure 58.

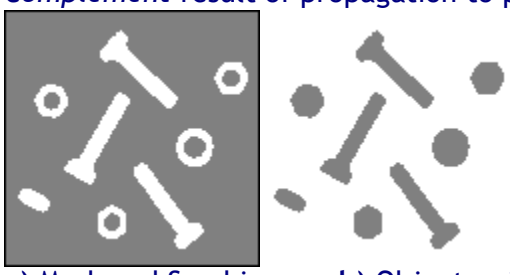
Segment image to produce binary mask representation.
Compute *skeleton* without end pixels - eq.
Use *salt* filter to remove single skeleton pixels.
Propagate remaining skeleton pixels into original binary mask - eq.



a) Binary image **b)** Skeleton after salt filter **c)** Objects with holes
Figure 58: Isolation of objects with holes using morphological operations.
The binary objects are shown in gray and the skeletons, after application of the salt filter, are shown as a black overlay on the binary objects. Note that this procedure uses no parameters other then the fundamental choice of connectivity; it is free from "magic numbers." In the example shown in Figure 58, the 8-connected definition was used as well as the structuring element $\boldsymbol{B} = \boldsymbol{N_8}$.

* ***Filling holes in objects*** - To fill holes in objects we use the following procedure, which is illustrated in Figure 59.

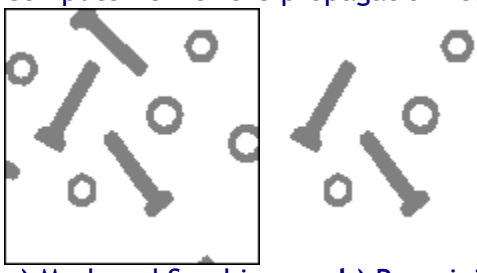
Segment image to produce binary representation of objects.
Compute *complement* of binary image as a *mask image*.
Generate a *seed image* as the border of the image.
Propagate the *seed* into the *mask* - eq.
Complement result of propagation to produce final result.



a) Mask and Seed images **b)** Objects with holes filled
Figure 59: Filling holes in objects.

The *mask image* is illustrated in *gray* in Figure 59a and the *seed image* is shown in *black* in that same illustration. When the object pixels are specified with a connectivity of C = 8, then the propagation into the mask (background) image should be performed with a connectivity of C = 4, that is, dilations with the structuring element $B = N_4$. This procedure is also free of "magic numbers."

* ***Removing border-touching objects*** - Objects that are connected to the image border are not suitable for analysis. To eliminate them we can use a series of morphological operations that are illustrated in Figure 60.
Segment image to produce binary *mask image* of objects.
Generate a *seed image* as the border of the image.
Propagate the *seed* into the *mask* - eq.
Compute *XOR* of the propagation result and the *mask image* as final result



a) Mask and Seed images **b)** Remaining objects
Figure 60: Removing objects touching borders.

The *mask image* is illustrated in *gray* in Figure 60a and the *seed image* is shown in *black* in that same illustration. If the structuring element used in the propagation is $B = N_4$, then objects are removed that are 4-connected with the image boundary. If $B = N_8$ is used then objects that 8-connected with the boundary are removed.
* *Exo-skeleton* - The *exo-skeleton* of a set of objects is the skeleton of the background that contains the objects. The exo-skeleton produces a partition of the image into regions each of which contains one object. The actual skeletonization (eq.) is performed without the preservation of end pixels and with the border set to "0." The procedure is described below and the result is illustrated in Figure 61.
Segment image to produce binary image.
Compute *complement* of binary image.
Compute *skeleton* using eq. *i+ii* with border set to "0".

Figure 61: Exo-skeleton.

* ***Touching objects*** - Segmentation procedures frequently have difficulty separating slightly touching, yet distinct, objects. The following procedure provides a mechanism to separate these objects and makes minimal use of "magic numbers." The exo-skeleton produces a partition of the image into regions each of which contains one object. The actual skeletonization is performed without the preservation of end pixels and with the border set to "0." The procedure is illustrated in Figure 62.

Segment image to produce binary image.
Compute a "small number" of *erosions* with $B = N_4$.
Compute *exo-skeleton* of eroded result.
Complement *exo-skeleton* result.
Compute *AND* of original binary image and the complemented exo-skeleton.

a) Eroded and exo-skeleton images **b)** Objects separated (detail)
Figure 62: Separation of touching objects.

The ***eroded binary image*** is illustrated in *gray* in Figure 62a and the *exo-skeleton image* is shown in *black* in that same illustration. An enlarged section of the final result is shown in Figure 62b and the separation is easily seen. This procedure involves choosing a small, minimum number of erosions but the number is not critical as long as it initiates a coarse separation of the desired objects. The actual separation is performed by the exo-skeleton which, itself, is free of "magic numbers." If the exo-skeleton is 8-connected than the background separating the objects will be 8-connected. The objects, themselves, will be disconnected according to the 4-connected criterion. (See Section 9.6 and Figure 36.)

Gray-value mathematical morphology

As we have seen in Section 10.1.2, gray-value morphological processing techniques can be used for practical problems such as shading correction. In this section several other techniques will be presented.

* ***Top-hat transform*** - The isolation of gray-value objects that are convex can be accomplished with the *top-hat transform* as developed by Meyer. Depending upon whether we are dealing with light objects on a dark background or dark objects on a light background, the transform is defined as:

Light objects - $TopHat(A,B) = A - (A \circ B) = A - \max_B\left(\min_B(A)\right)$

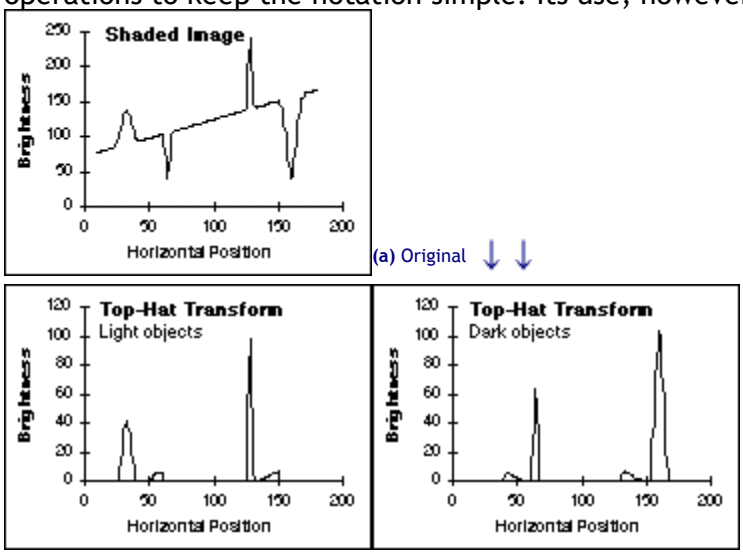
Dark objects - $TopHat(A,B) = (A \bullet B) - A = \min_B\left(\max_B(A)\right) - A$

Where, the structuring element ***B*** is chosen to be bigger than the objects in question, and if possible, to have a convex shape. Because of the properties given in eqs. And, *Topat*(***A***,***B***) >= 0. An example of this technique is shown in Figure 63.
The original image including shading is processed by a 15 x 1 structuring element as described in eqs. and to produce the desired result. Note that the transform for dark objects has been defined in such a way as to yield "positive" objects as opposed to "negative" objects. Other definitions are, of course, possible.

* ***Thresholding*** - A simple estimate of a locally varying threshold surface can be derived from morphological processing as follows:

Threshold surface - $\theta[m,n] = \frac{1}{2}\left(\max(A) + \min(A)\right)$

Once again, we suppress the notation for the structuring element ***B*** under the *max* and *min* operations to keep the notation simple. Its use, however, is understood.



(a) Light object transform **(b)** Dark object transform
Figure 63: Top-hat transforms.

* ***Local contrast stretching*** - Using morphological operations we can implement a technique for *local contrast stretching*. That is, the amount of stretching that will be applied in a neighborhood will be controlled by the original contrast in that neighborhood. The morphological gradient defined in eq. may also be seen as related to a measure of the local contrast in the window defined by the structuring element ***B***:

$LocalContrast(A,B) = \max(A) - \min(A)$

The procedure for local contrast stretching is given by:

$$c[m,n] = scale \bullet \frac{\mathrm{A} - \min(\mathrm{A})}{\max(\mathrm{A}) - \min(\mathrm{A})}$$

The *max* and *min* operations are taken over the structuring element ***B***. The effect of this procedure is illustrated in Figure 64. It is clear that this *local* operation is an extended version of the *point* operation for contrast stretching presented in eq. (77).

↑ before after ↑ ↑ before after ↑ ↑ before after ↑

Figure 64: Local contrast stretching.

Using standard test images (as we have seen in so many examples) illustrates the power of this local morphological filtering approach.

Mathematics-based Operations

We distinguish in this section between binary arithmetic and ordinary arithmetic. In the binary case there are two brightness values "0" and "1". In the ordinary case we begin with 2^B brightness values or levels but the processing of the image can easily generate many more levels. For this reason many *software* systems provide 16 or 32 bit representations for pixel brightness in order to avoid problems with arithmetic overflow.

Binary operations

Operations based on binary (Boolean) arithmetic form the basis for a powerful set of tools that will be described here and extended in Section 9.6, mathematical morphology. The operations described below are point operations and thus admit a variety of efficient implementations including simple look-up tables. The standard notation for the basic set of binary operations is:

$$NOT \qquad c = \bar{a}$$

$$OR \qquad c = a + b$$

$$AND \qquad c = a \bullet b$$

$$XOR \qquad c = a \oplus b = a \bullet \bar{b} + \bar{a} \bullet b$$

$$SUB \qquad c = a \backslash b = a - b = a \bullet \bar{b}$$

The implication is that each operation is applied on a pixel-by-pixel basis. For example, $c[m,n] = a[m,n] \bullet \bar{b}[m,n] \quad \forall m,n$. The definition of each operation is:

NOT	
a	
0	1
1	0
↑ input	↑ output

OR	b	
a	0	1
0	0	1
1	1	1

AND	b	
a	0	1
0	0	0
1	0	1

XOR	b	
a	0	1
0	0	1
1	1	0

SUB	b	
a	0	1
0	0	0
1	1	0

These operations are illustrated in Figure 22 where the binary value "1" is shown in black and the value "0" in white.

a) Image *a* b) Image *b*

c) NOT $(b) = \bar{b}$

d) OR $(a,b) = a + b$

e) AND$(a,b) = a * b$

f) XOR $(a,b) = a \oplus b$

g) SUB$(a,b) = a \setminus b$

Figure 22: Examples of the various binary point operations.

The SUB (*) operation can be particularly useful when the image *a* represents a region-of-interest that we want to analyze systematically and the image *b* represents objects that, having been analyzed, can now be discarded, that is subtracted, from the region.

Arithmetic-based operations

The gray-value point operations that form the basis for image processing are based on ordinary mathematics and include:

Operation	*Definition*	*Preferred data type*
ADD	c = a + b	Integer
SUB	c = a - b	Integer
MUL	c = a * b	Integer or floating point
DIV	c = a / b	Floating point
LOG	c = log(a)	Floating point
EXP	c = exp(a)	Floating point
SQRT	c = sqrt(a)	Floating point
TRIG.	c = sin/cos/tan(a)	Floating point
INVERT	c = $(2^B - 1) - a$	Integer

Convolution-based Operations

Convolution, the mathematical, *local* operation defined in Section 3.1 is central to modern image processing. The basic idea is that a window of some finite size and shape--the *support*--is scanned across the image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the filter assigned to every pixel of the window itself. The window with its weights is called the *convolution kernel*. This leads directly to the following variation on eq. . If the filter $h[j,k]$ is zero outside the (rectangular) window $\{j=0,1,\ldots,J-1;\ k=0,1,\ldots,K-1\}$, then, using eq. , the convolution can be written as the following finite sum:

$$c[m,n] = a[m,n] \otimes h[m,n] = \sum_{j=0}^{J-1}\sum_{k=0}^{K-1} h[j,k]a[m-j,n-k]$$

This equation can be viewed as more than just a pragmatic mechanism for smoothing or sharpening an image. Further, while eq. illustrates the local character of this operation, eqs. and suggest that the operation can be implemented through the use of the Fourier domain which requires a global operation, the Fourier transform. Both of these aspects will be discussed below.

Background

In a variety of image-forming systems an appropriate model for the transformation of the physical signal a (x,y) into an electronic signal c(x,y) is the convolution of the input signal with the impulse response of the sensor system. This system might consist of both an optical as well as an electrical sub-system. If each of these systems can be treated as a linear, shift-invariant (LSI) system then the convolution model is appropriate. The definitions of these two, possible, system properties are given below:

Linearity - $\begin{array}{lll} If & a_1 \rightarrow c_1 \quad and & a_2 \rightarrow c_2 \\ Then & w_1 \bullet a_1 + w_2 \bullet a_2 & \rightarrow w_1 \bullet c_1 + w_2 \bullet c_2 \end{array}$

Shift-Invariance - $\begin{array}{ll} If & a(x,y) \rightarrow c(x,y) \\ Then & a(x-x_o, y-y_o) \rightarrow c(x-x_o, y-y_o) \end{array}$

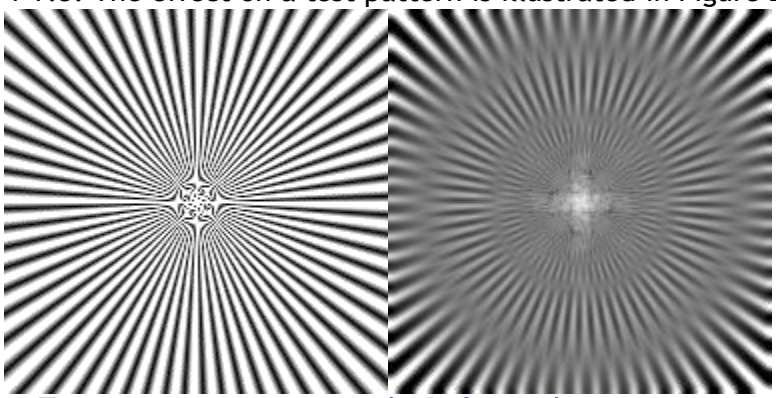
where w_1 and w_2 are arbitrary complex constants and x_o and y_o are coordinates corresponding to arbitrary spatial translations.

Two remarks are appropriate at this point. First, linearity implies (by choosing $w_1 = w_2 = 0$) that "zero in" gives "zero out". The offset described in eq. means that such camera signals are not the output of a linear system and thus (strictly speaking) the convolution result is not applicable. Fortunately, it is straightforward to correct for this non-linear effect. (See Section 10.1.)

Second, optical lenses with a magnification, *M*, other than 1x are not shift invariant; a translation of 1 unit in the input image $a(x,y)$ produces a translation of *M* units in the output image $c(x,y)$. Due to the Fourier property described in eq. this case can still be handled by linear system theory.

If an impulse point of light $d(x,y)$ is imaged through an LSI system then the impulse response of that system is called the *point spread function* (*PSF*). The output image then becomes the convolution of the input image with the *PSF*. The Fourier transform of the *PSF* is called the *optical transfer function* (*OTF*). For optical systems that are circularly symmetric, aberration-free, and diffraction-limited the *PSF* is given by the Airy disk, shown in Table 4-T.5. The *OTF* of the Airy disk is also presented in Table 4-T.5.

If the convolution window is not the diffraction-limited PSF of the lens but rather the effect of defocusing a lens then an appropriate model for *h* (*x*,*y*) is a pill box of radius *a* as described in Table 4-T.3. The effect on a test pattern is illustrated in Figure 23.



a) Test pattern b) Defocused image

Figure 23: Convolution of test pattern with a pill box of radius *a*=4.5 pixels.

The effect of the defocusing is more than just simple blurring or smoothing. The almost periodic negative lobes in the transfer function in Table 4-T.3 produce a 180deg. phase shift in which black turns to white and vice-versa. The phase shift is clearly visible in Figure 23b.

Convolution in the spatial domain

In describing filters based on convolution we will use the following convention. Given a filter h[j,k] of dimensions J x K, we will consider the coordinate [j=0,k=0] to be in the center of the filter matrix, h. This is illustrated in Figure 24. The "center" is well-defined when J and K are odd; for the case where they are even, we will use the approximations (J/2, K/2) for the "center" of the matrix.

$$\mathbf{h} = \begin{bmatrix} h\left[-\left(J-\tfrac{1}{2}\right),-\left(K-\tfrac{1}{2}\right)\right] & \cdots & \cdots & h\left[0,-\left(K-\tfrac{1}{2}\right)\right] & \cdots & \cdots & h\left[\left(J-\tfrac{1}{2}\right),-\left(K-\tfrac{1}{2}\right)\right] \\ \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots \\ \vdots & \cdots & h[-1,-1] & h[0,-1] & h[1,-1] & \cdots & \vdots \\ h\left[-\left(J-\tfrac{1}{2}\right),0\right] & \cdots & h[-1,0] & h[0,0] & h[1,0] & \cdots & h\left[\left(J-\tfrac{1}{2}\right),0\right] \\ \vdots & \cdots & h[-1,1] & h[0,1] & h[1,+1] & \cdots & \vdots \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h\left[-\left(J-\tfrac{1}{2}\right),\left(K-\tfrac{1}{2}\right)\right] & \cdots & \cdots & h\left[0,\left(K-\tfrac{1}{2}\right)\right] & \cdots & \cdots & h\left[\left(J-\tfrac{1}{2}\right),\left(K-\tfrac{1}{2}\right)\right] \end{bmatrix}$$

Figure 24: Coordinate system for describing *h*[*j*,*k*]

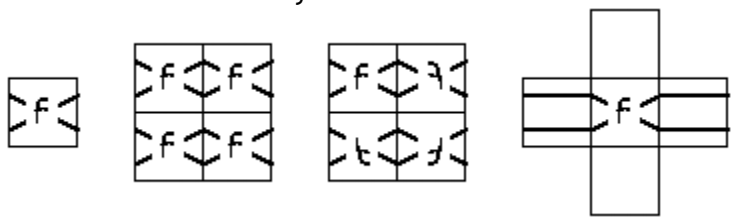
When we examine the convolution sum (eq.) closely, several issues become evident.

* **Evaluation of formula** for *m*=*n*=0 while rewriting the limits of the convolution sum based on the "centering" of *h*[*j*,*k*] shows that values of *a*[*j*,*k*] can be required that are outside the image boundaries:

$$c[0,0] = \sum_{j=-J_o}^{+J_o} \sum_{k=-K_o}^{+K_o} h[j,k]a[-j,-k] \qquad J_o = \frac{(J-1)}{2},\ K_o = \frac{(K-1)}{2}$$

The question arises - what values should we assign to the image *a*[*m*,*n*] for *m*<0, *m*>=*M*, *n*<0, and *n*>=*N*? There is no "answer" to this question. There are only alternatives among which we are free to choose assuming we understand the possible consequences of our choice. The standard alternatives

are a) extend the images with a constant (possibly zero) brightness value, b) extend the image periodically, c) extend the image by mirroring it at its boundaries, or d) extend the values at the boundaries indefinitely. These alternatives are illustrated in Figure 25.



(a) (b) (c) (d)

Figure 25: Examples of various alternatives to extend an image outside its formal boundaries. See text for explanation.

* When the convolution sum is written in the standard form (eq.) for an image *a*[*m*,*n*] of size *M* x *N*:

$$c[m,n] = \sum_{j=0}^{M-1}\sum_{k=0}^{N-1} a[j,k]h[m-j,n-k]$$

we see that the convolution kernel *h*[*j*,*k*] is mirrored around *j*=*k*=0 to produce *h*[-*j*,-*k*] before it is translated by [*m*,*n*] as indicated in eq. . While some convolution kernels in common use are symmetric in this respect, *h*[*j*,*k*]= *h*[-*j*,-*k*], many are not. (See Section 9.5.) Care must therefore be taken in the implementation of filters with respect to the mirroring requirements.

* The computational complexity for a *K* x *K* convolution kernel implemented in the spatial domain on an image of *N* x *N* is $O(K^2)$ where the complexity is measured *per pixel* on the basis of the number of multiplies-and-adds (MADDs).

* The value computed by a convolution that begins with integer brightness for *a*[*m*,*n*] may produce a rational number or a floating point number in the result *c*[*m*,*n*]. Working exclusively with integer brightness values will, therefore, cause roundoff errors.

* Inspection of eq. reveals another possibility for efficient implementation of convolution. If the convolution kernel *h*[*j*,*k*] is *separable*, that is, if the kernel can be written as:

$$h[j,k] = h_{row}[k] \bullet h_{col}[j]$$

then the filtering can be performed as follows:

$$c[m,n] = \sum_{j=0}^{J-1}\left\{\sum_{k=0}^{K-1} h_{row}[k]a[m-j,n-k]\right\}h_{col}[j]$$

This means that instead of applying one, two-dimensional filter it is possible to apply two, one-dimensional filters, the first one in the *k* direction and the second one in the *j* direction. For an *N* x *N* image this, in general, reduces the computational complexity per pixel from *O*(*J** *K*) to *O*(*J*+*K*).
An alternative way of writing separability is to note that the convolution kernel (Figure 24) is a matrix **h** and, if separable, **h** can be written as:

$$\underset{(J\times K)}{[\mathbf{h}]} = \underset{(J\times 1)}{[\mathbf{h}_{col}]} \bullet \underset{(1\times K)}{[\mathbf{h}_{row}]^t}$$

where "*t*" denotes the matrix transpose operation. In other words, **h** can be expressed as the *outer product* of a column vector [$\mathbf{h}_{col}$] and a row vector [$\mathbf{h}_{row}$].

* For certain filters it is possible to find an *incremental implementation* for a convolution. As the convolution window moves over the image (see eq.), the leftmost column of image data under the window is shifted out as a new column of image data is shifted in from the right. Efficient algorithms can take advantage of this and, when combined with separable filters as described above, this can lead to algorithms where the computational complexity per pixel is *O*(*constant*).

Convolution in the frequency domain

In Section 3.4 we indicated that there was an alternative method to implement the filtering of images through convolution. Based on eq. it appears possible to achieve the same result as in eq. by the following sequence of operations:

i) Compute $A(\Omega, \Psi) = F\{a[m,n]\}$

ii) Multiply $A(\Omega, \Psi)$ by the *precomputed* $(\Omega, \Psi) = F\{h[m,n]\}$

iii) Compute the result $c[m,n] = F^{-1}\{A(\Omega, \Psi)*(\Omega, \Psi)\}$

* While it might seem that the "recipe" given above in eq. circumvents the problems associated with direct convolution in the spatial domain--specifically, determining values for the image outside the boundaries of the image--the Fourier domain approach, in fact, simply "assumes" that the image is repeated periodically outside its boundaries as illustrated in Figure 25b. This phenomenon is referred to as *circular convolution*.

If circular convolution is not acceptable then the other possibilities illustrated in Figure 25 can be realized by embedding the image $a[m,n]$ and the filter (Ω, Ψ) in larger matrices with the desired image extension mechanism for $a[m,n]$ being explicitly implemented.

* The computational complexity per pixel of the Fourier approach for an image of $N \times N$ and for a convolution kernel of $K \times K$ is $O(\log N)$ *complex* MADDs *independent of K*. ere we assume that $N > K$ and that N is a highly composite number such as a power of two. (See also 2.1.) This latter assumption permits use of the computationally efficient Fast Fourier Transform (*FFT*) algorithm. Surprisingly then, the indirect route described by eq. can be faster than the direct route given in eq. . This requires, in general, that $K^2 >> \log N$. The range of K and N for which this holds depends on the specifics of the implementation. For the machine on which this manuscript is being written and the specific image processing package that is being used, for an image of $N = 256$ the Fourier approach is faster than the convolution approach when $K >= 15$. (It should be noted that in this comparison the direct convolution involves only integer arithmetic while the Fourier domain approach requires complex floating point arithmetic.)

Smoothing Operations

These algorithms are applied in order to reduce noise and/or to prepare images for further processing such as segmentation. We distinguish between linear and non- linear algorithms where the former are amenable to analysis in the Fourier domain and the latter are not. We also distinguish between implementations based on a rectangular support for the filter and implementations based on a circular support for the filter.

Linear Filters

Several filtering algorithms will be presented together with the most useful supports.

* ***Uniform filter*** - The output image is based on a local averaging of the input filter where all of the values within the filter support have the same weight. In the continuous spatial domain (x,y) the *PSF* and transfer function are given in Table 4-T.1 for the rectangular case and in Table 4-T.3 for the circular (pill box) case. For the discrete spatial domain [m,n] the filter values are the samples of the continuous domain case. Examples for the rectangular case (J=K=5) and the circular case (R=2.5) are shown in Figure 26.

$$h_{rect}[j,k] = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad h_{circ}[j,k] = \frac{1}{21}\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

(a) Rectangular filter (J=K=5) (b) Circular filter (R=2.5)

Figure 26: Uniform filters for image smoothing

Note that in both cases the filter is normalized so that $\Sigma\, h[j,k] = 1$. This is done so that if the input $a[m,n]$ is a constant then the output image $c[m,n]$ is the same constant. The justification can be found in the Fourier transform property described in eq. . As can be seen from Table 4, both of these filters have transfer functions that have negative lobes and can, therefore, lead to phase reversal as seen in Figure 23. The square implementation of the filter is separable and incremental; the circular implementation is incremental.

* ***Triangular filter*** - The output image is based on a local averaging of the input filter where the values within the filter support have differing weights. In general, the filter can be seen as the convolution of two (identical) uniform filters either rectangular or circular and this has direct consequences for the computational complexity. (See Table 13.) In the continuous spatial domain the *PSF* and transfer function are given in Table 4-T.2 for the rectangular support case and in Table 4-T.4 for the circular (pill box) support case. As seen in Table 4 the transfer functions of these filters do not have negative lobes and thus do not exhibit phase reversal.
Examples for the rectangular support case (J=K=5) and the circular support case (R=2.5) are shown in Figure 27. The filter is again normalized so that $\Sigma\, h[j,k]$=1.

$$h_{rect}[j,k] = \frac{1}{81}\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix} h_{circ}[j,k] = \frac{1}{25}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 2 & 0 \\ 1 & 2 & 5 & 2 & 1 \\ 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(a) Pyramidal filter (*J*=*K*=5) **(b)** Cone filter (*R*=2.5)
Figure 27: Triangular filters for image smoothing

* ***Gaussian filter*** - The use of the Gaussian kernel for smoothing has become extremely popular. This has to do with certain properties of the Gaussian (e.g. the central limit theorem, minimum space-bandwidth product) as well as several application areas such as edge finding and scale space analysis. The *PSF* and transfer function for the continuous space Gaussian are given in Table 4-T6. The Gaussian filter is separable:

$$h(x,y) = g_{2D}(x,y) = \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\left(x^2/2\sigma^2\right)}\right) \bullet \left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\left(y^2/2\sigma^2\right)}\right)$$

$$= g_{1D}(x) \bullet g_{1D}(y)$$

There are four distinct ways to implement the Gaussian:
- Convolution using a finite number of samples (N_o) of the Gaussian as the convolution kernel. It is common to choose $N_o = \lceil 3\sigma \rceil$ or $\lceil 5\sigma \rceil$.

$$g_{1D}[n] = \begin{cases} \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(n^2/2\sigma^2\right)} & |n| \le N_o \\ 0 & |n| > N_o \end{cases}$$

- Repetitive convolution using a uniform filter as the convolution kernel.

$$g_{1D}[n] \approx u[n] \otimes u[n] \otimes u[n]$$

$$u[n] = \begin{cases} 1/(2N_o+1) & |n| \le N_o \\ 0 & |n| > N_o \end{cases}$$

The actual implementation (in each dimension) is usually of the form:

$$c[n] = ((a[n] \otimes u[n]) \otimes u[n]) \otimes u[n]$$

This implementation makes use of the approximation afforded by the central limit theorem. For a desired σ with eq. , we use $N_o = \lceil \sigma \rceil$ although this severely restricts our choice of σ's to integer values.
- Multiplication in the frequency domain. As the Fourier transform of a Gaussian *is* a Gaussian (see Table -T.6), this means that it is straightforward to prepare a filter $(\Omega, \Psi) = G_{2D}(\Omega, \Psi)$ for use with eq. . To avoid truncation effects in the frequency domain due to the infinite extent of the Gaussian it is important to choose a σ that is sufficiently large. Choosing $\sigma > k/\pi$ where k = 3 or 4 will usually be sufficient.
- Use of a recursive filter implementation. A recursive filter has an infinite impulse response and thus an infinite support. The separable Gaussian filter can also be implemented by applying the following recipe in each dimension when σ >= 0.5.

i) Choose the σ based on the desired goal of the filtering; *ii)* Determine the parameter *q* based on eq. ; *iii)* Use eq. to determine the filter coefficients $\{b_0,b_1,b_2,b_3,B\}$; *iv)* Apply the forward difference equation, eq. ; *v)* Apply the backward difference equation, eq. ;
The relation between the desired σ and *q* is given by:

$$q = \begin{cases} .98711\sigma - 0.96330 & \sigma \geq 2.5 \\ 3.97156 - 4.14554\sqrt{1-.26891\sigma} & 0.5 \leq \sigma \leq 2.5 \end{cases}$$

The *filter coefficients* $\{b_0, b1, b2, b3, B\}$ are defined by:

$$b_0 = 1.57825 + (2.44413q) + (1.4281q^2) + (0.422205q^3)$$

$$b_1 = (2.44413q) + (2.85619q^2) + (1.26661q^3)$$

$$b_2 = -(1.4281q^2) - (1.26661q^3)$$

$$b_3 = 0.422205q^3$$

$$B = 1 - (b_1 + b_2 + b_3)/b_0$$

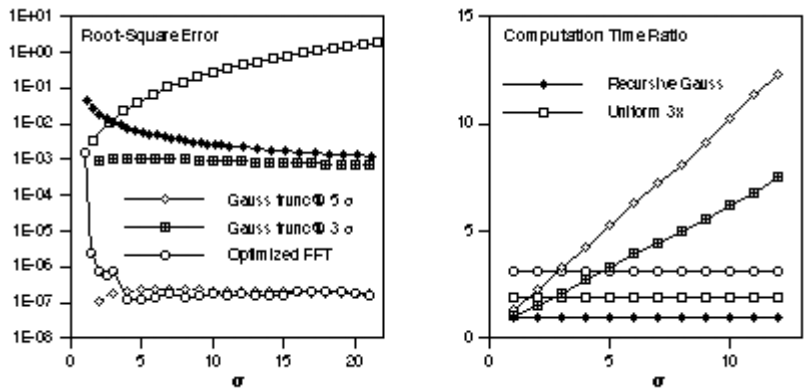
The one-dimensional *forward difference equation* takes an input row (or column) *a*[*n*] and produces an intermediate output result *w*[*n*] given by:

$$w[n] = Ba[n] + (b_1 w[n-1] + b_2 w[n-2] + b_3 w[n-3])/b_0$$

The one-dimensional *backward difference equation* takes the intermediate result *w*[*n*] and produces the output *c*[*n*] given by:

$$c[n] = Bw[n] + (b_1 c[n+1] + b_2 c[n+2] + b_3 c[n+3])/b_0$$

The forward equation is applied from *n* = 0 *up* to *n* = *N* - 1 while the backward equation is applied from *n* = *N* - 1 *down* to *n* = 0.
The relative performance of these, various implementation of the Gaussian filter can be described as follows. Using the *root-square error* $\sqrt{\sum_{n=-\infty}^{+\infty} |g[n|\sigma] - h[n]|^2}$ between a true, infinite-extent Gaussian, $g[n|\sigma]$, and an approximated Gaussian, *h*[*n*], as a measure of accuracy, the various algorithms described above give the results shown in Figure 28a. The relative speed of the various algorithms in shown in Figure 28b.
The root-square error measure is extremely conservative and thus all filters, with the exception of "Uniform 3x" for large σ, are sufficiently accurate. The recursive implementation is the fastest independent of σ; the other implementations can be significantly slower. The FFT implementation, for example, is 3.1 times slower for *N*=256 . Further, the FFT requires that *N* be a highly composite number.



a) Accuracy comparison **b)** Speed comparison

Figure 28: Comparison of various Gaussian algorithms with *N*=256. The legend is spread across both graphs

* ***Other*** - The Fourier domain approach offers the opportunity to implement a variety of smoothing algorithms. The smoothing filters will then be *lowpass filters*. In general it is desirable to use a lowpass filter that has zero phase so as not to produce phase distortion when filtering the image. The importance of phase was illustrated in Figures 5 and 23. When the frequency domain characteristics can be represented in an analytic form, then this can lead to relatively straightforward implementations of (Ω, Ψ). Possible candidates include the lowpass filters "Airy" and "Exponential Decay" found in Table 4-T.5 and Table 4-T.8, respectively.

Non-Linear Filters

A variety of smoothing filters have been developed that are not linear. While they cannot, in general, be submitted to Fourier analysis, their properties and domains of application have been studied extensively.

* ***Median filter*** - The median statistic was described in Section 3.5.2. A median filter is based upon moving a window over an image (as in a convolution) and computing the output pixel as the median value of the brightness within the input window. If the window is *J* x *K* in size we can order the *J***K* pixels in brightness value from smallest to largest. If *J***K* is odd then the median will be the (*J***K*+1)/2 entry in the list of ordered brightness. Note that the value selected will be exactly equal to one of the existing brightness so that no roundoff error will be involved if we want to work exclusively with integer brightness values. The algorithm as it is described above has a generic complexity per pixel of *O*(*J***K**log(*J***K*)). Fortunately, a fast algorithm (due to uang et al.) exists that reduces the complexity to *O*(*K*) assuming *J* >= *K*.

A useful variation on the theme of the median filter is the *percentile filter*. ere the center pixel in the window is replaced not by the 50% (median) brightness value but rather by the *p*% brightness value where *p*% ranges from 0% (the *minimum filter*) to 100% (the *maximum filter*). Values other then (*p*=50)% do not, in general, correspond to smoothing filters.

* ***Kuwahara filter*** - Edges play an important role in our perception of images (see Figure 15) as well as in the analysis of images. As such, it is important to be able to smooth images without disturbing the sharpness and, if possible, the position of edges. A filter that accomplishes this goal is termed an *edge-preserving filter* and one particular example is the Kuwahara filter. Although this filter can be implemented for a variety of different window shapes, the algorithm will be described for a square window of size $J = K = 4L + 1$ where L is an integer. The window is partitioned into four regions as shown in Figure 29.

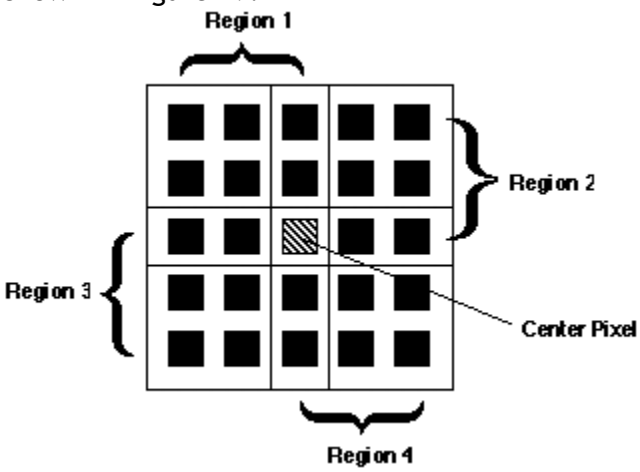


Figure 29: Four, square regions defined for the Kuwahara filter. In this example L=1 and thus J=K=5. Each region is [(J+1)/2] x [(K+1)/2].
In each of the four regions (i=1,2,3,4), the mean brightness, m_i in eq. , and the *variance*$_i$, s_i^2 in eq. , are measured. The output value of the center pixel in the window is the mean value of that region that has the smallest variance.

Summary of Smoothing Algorithms

The following table summarizes the various properties of the smoothing algorithms presented above. The filter size is assumed to be bounded by a rectangle of J x K where, without loss of generality, J >= K. The image size is N x N.

Algorithm	*Domain*	*Type*	*Support*	*Separable / Incremental*	*Complexity/pixel*
Uniform	Space	Linear	Square	Y / Y	O(*constant*)
Uniform	Space	Linear	Circular	N / Y	$O(K)$
Triangle	Space	Linear	Square	Y / N	O(*constant*) [a]
Triangle	Space	Linear	Circular	N / N	$O(K)$ [a]
Gaussian	Space	Linear	∞ [a]	Y / N	O(*constant*) [a]
Median	Space	Non-Linear	Square	N / Y	$O(K)$ [a]
Kuwahara	Space	Non-Linear	Square [a]	N / N	$O(J^* K)$
Other	Frequency	Linear	--	-- / --	$O(\log N)$

Table 13: Characteristics of smoothing filters. [a]See text for additional explanation.
Examples of the effect of various smoothing algorithms are shown in Figure 30.

a) Original **b)** Uniform 5 x 5 **c)** Gaussian (σ = 2.5)

d) Median 5 x 5 **e)** Kuwahara 5 x 5

Figure 30: Illustration of various linear and non-linear smoothing filters

Shading Correction

The method by which images are produced--the interaction between objects in real space, the illumination, and the camera--frequently leads to situations where the image exhibits significant shading across the field-of-view. In some cases the image might be bright in the center and decrease in brightness as one goes to the edge of the field-of-view. In other cases the image might be darker on the left side and lighter on the right side. The shading might be caused by non-uniform illumination, non-uniform camera sensitivity, or even dirt and dust on glass (lens) surfaces. In general this shading effect is undesirable. Eliminating it is frequently necessary for subsequent processing and especially when image analysis or image understanding is the final goal.

Model of shading

In general we begin with a model for the shading effect. The illumination $I_{ill}(x,y)$ usually interacts in a multiplicative with the object $a(x,y)$ to produce the image $b(x,y)$:

$$b(x,y) = I_{ill}(x,y) \bullet a(x,y)$$

with the object representing various imaging modalities such as:

$$a(x,y) = \begin{cases} r(x,y) & \text{reflectance model} \\ 10^{-OD(x,y)} & \text{absorption model} \\ c(x,y) & \text{fluorescence model} \end{cases}$$

where at position (x,y), $r(x,y)$ is the *reflectance*, $OD(x,y)$ is the *optical density*, and $c(x,y)$ is the concentration of fluorescent material. Parenthetically, we note that the fluorescence model only holds for low concentrations. The camera may then contribute *gain* and *offset* terms, as in eq. (74), so that:

Total shading -

$$\begin{aligned} c[m,n] &= gain[m,n] \bullet b[m,n] + offset[m,n] \\ &= gain[m,n] \bullet I_{ill}[m,n] \bullet a[m,n] + offset[m,n] \end{aligned}$$

In general we assume that $I_{ill}[m,n]$ is slowly varying compared to $a[m,n]$.

Estimate of shading

We distinguish between two cases for the determination of $a[m,n]$ starting from $c[m,n]$. In both cases we intend to estimate the shading terms $\{gain[m,n]*I_{ill}[m,n]\}$ and $\{offset[m,n]\}$. While in the first case we assume that we have only the recorded image $c[m,n]$ with which to work, in the second case we assume that we can record two, additional, calibration images.

* *A posteriori estimate* - In this case we attempt to extract the shading estimate from $c[m,n]$. The most common possibilities are the following.

Lowpass filtering - We compute a smoothed version of $c[m,n]$ where the smoothing is large compared to the size of the objects in the image. This smoothed version is intended to be an estimate of the background of the image. We then subtract the smoothed version from $c[m,n]$ and then restore the desired DC value. In formula:

Lowpass - $\hat{a}[m,n] = c[m,n] - LowPass\{c[m,n]\} + constant$

where $\hat{a}[m,n]$ is the estimate of $a[m,n]$. Choosing the appropriate lowpass filter means knowing the appropriate spatial frequencies in the Fourier domain where the shading terms dominate.

omomorphic filtering - We note that, if the *offset*[*m*,*n*] = 0, then *c*[*m*,*n*] consists solely of multiplicative terms. Further, the term {*gain*[*m*,*n*]*I_{ill}[*m*,*n*]} is slowly varying while *a*[*m*,*n*] presumably is not. We therefore take the logarithm of *c*[*m*,*n*] to produce two terms one of which is low frequency and one of which is high frequency. We suppress the shading by high pass filtering the logarithm of *c*[*m*,*n*] and then take the exponent (inverse logarithm) to restore the image. This procedure is based on *homomorphic filtering* as developed by Oppenheim, Schafer and Stockham . In formula:

$$i)\quad c[m,n] = gain[m,n] \bullet I_{ill}[m,n] \bullet a[m,n]$$

$$ii)\quad \ln\{c[m,n]\} = \ln\left\{\underbrace{gain[m,n] \bullet I_{ill}[m,n]}_{slowly\ varying}\right\} + \ln\left\{\underbrace{a[m,n]}_{rapidly\ varying}\right\}$$

$$iii)\quad HighPass\{\ln\{c[m,n]\}\} \approx \ln\{a[m,n]\}$$

$$iv)\quad \hat{a}[m,n] = \exp\{HighPass\{\ln\{c[m,n]\}\}\}$$

Morphological filtering - We again compute a smoothed version of *c*[*m*,*n*] where the smoothing is large compared to the size of the objects in the image but this time using morphological smoothing as in eq. . This smoothed version is the estimate of the background of the image. We then subtract the smoothed version from *c*[*m*,*n*] and then restore the desired DC value. In formula:

$$\hat{a}[m,n] = c[m,n] - MorphSmooth\{c[m,n]\} + constant$$

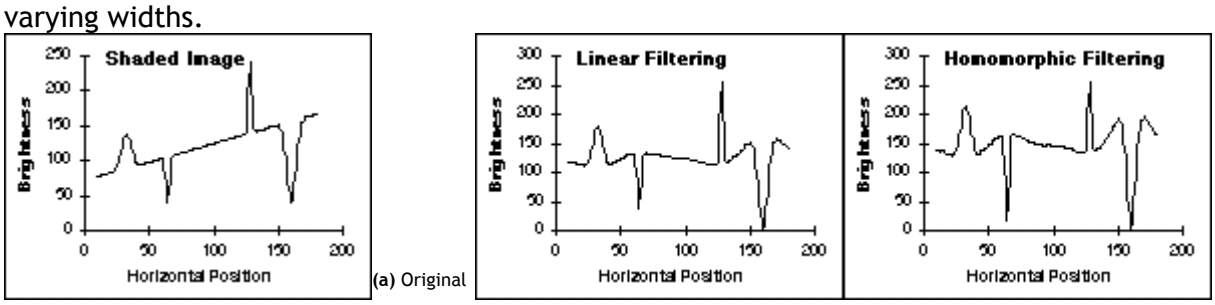
Choosing the appropriate morphological filter window means knowing (or estimating) the size of the largest objects of interest.

* *A priori estimate* - If it is possible to record test (calibration) images through the cameras system, then the most appropriate technique for the removal of shading effects is to record two images - *BLACK*[*m*,*n*] and *WHITE*[*m*,*n*]. The *BLACK* image is generated by covering the lens leading to *b*[*m*,*n*] = 0 which in turn leads to *BLACK*[*m*,*n*] = *offset*[*m*,*n*]. The *WHITE* image is generated by using *a*[*m*,*n*] = 1 which gives *WHITE*[*m*,*n*] = *gain*[*m*,*n*]*I_{ill}[*m*,*n*] + *offset*[*m*,*n*]. The correction then becomes:

$$\hat{a}[m,n] = constant \bullet \frac{c[m,n] - BLACK[m,n]}{WHITE[m,n] - BLACK[m,n]}$$

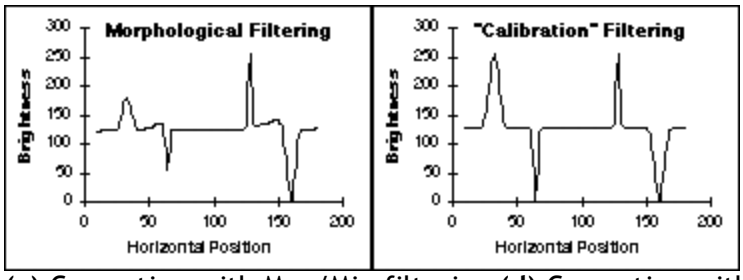
The *constant* term is chosen to produce the desired dynamic range.

The effects of these various techniques on the data from Figure 45 are shown in Figure 47. The shading is a simple, linear ramp increasing from left to right; the objects consist of Gaussian peaks of varying widths.



(a) Original

(b) Correction with Lowpass filtering **(c)** Correction with Logarithmic filtering



(c) Correction with Max/Min filtering **(d)** Correction with Test Images

Figure 47: Comparison of various shading correction algorithms. The final result (d) is identical to the original (not shown).

In summary, if it is possible to obtain *BLACK* and *WHITE* calibration images, then eq. is to be preferred. If this is not possible, then one of the other algorithms will be necessary.

Basic Enhancement and Restoration Techniques

The process of image acquisition frequently leads (inadvertently) to image degradation. Due to mechanical problems, out-of-focus blur, motion, inappropriate illumination, and noise the quality of the digitized image can be inferior to the original. The goal of *enhancement* is-- starting from a recorded image $c[m,n]$--to produce the most visually pleasing image $\hat{a}[m,n]$. The goal of restoration is--starting from a recorded image $c[m,n]$--to produce the best possible estimate $\hat{a}[m,n]$ of the original image $a[m,n]$. The goal of enhancement is beauty; the goal of restoration is truth.
The measure of success in restoration is usually an error measure between the original $a[m,n]$ and the estimate $\hat{a}[m,n]$: $E\{\hat{a}[m,n], a[m,n]\}$. *No mathematical error function is known that corresponds to human perceptual assessment of error*. The mean-square error function is commonly used because:
1. It is easy to compute;
2. It is differentiable implying that a minimum can be sought;
3. It corresponds to "signal energy" in the total error, and;
4. It has nice properties *vis à vis* Parseval's theorem, eqs. (22) and (23).
The *mean-square error* is defined by:

$$E\{\hat{a},a\} = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|\hat{a}[m,n]-a[m,n]\right|^2$$

In some techniques an error measure will not be necessary; in others it will be essential for evaluation and comparative purposes.

Unsharp masking

A well-known technique from photography to improve the visual quality of an image is to enhance the edges of the image. The technique is called *unsharp masking*. Edge enhancement means first isolating the edges in an image, amplifying them, and then adding them back into the image. Examination of Figure 33 shows that the Laplacian is a mechanism for isolating the gray level edges. This leads immediately to the technique:

$$\hat{a}[m,n] = a[m,n] - \left(k \bullet \nabla^2 a[m,n]\right)$$

The term k is the amplifying term and $k > 0$. The effect of this technique is shown in Figure 48.
The Laplacian used to produce Figure 48 is given by eq. (120) and the amplification term $k = 1$.

Original ↑ ↑ Laplacian-enhanced
Figure 48: Edge enhanced compared to original

Noise suppression

The techniques available to suppress noise can be divided into those techniques that are based on temporal information and those that are based on spatial information. By temporal information we mean that a sequence of images $\{a_p[m,n] \mid p=1,2,\ldots,P\}$ are available that contain *exactly* the same objects and that differ only in the sense of independent noise realizations. If this is the case and if the noise is additive, then simple averaging of the sequence:

Temporal averaging -

$$\hat{a}[m,n] = \frac{1}{P}\sum_{p=1}^{P} a_p[m,n]$$

will produce a result where the mean value of each pixel will be unchanged. For each pixel, however, the standard deviation will decrease from σ to $\sigma/\sqrt{P}$.

If temporal averaging is not possible, then spatial averaging can be used to decrease the noise. This generally occurs, however, at a cost to image sharpness. Four obvious choices for spatial averaging are the smoothing algorithms that have been described in Section 9.4 - Gaussian filtering (eq. (93)), median filtering, Kuwahara filtering, and morphological smoothing (eq.).

Within the class of linear filters, the optimal filter for restoration in the presence of noise is given by the *Wiener filter* . The word "optimal" is used here in the sense of minimum mean-square error (*mse*). Because the square root operation is monotonic increasing, the optimal filter also minimizes the root mean-square error (*rms*). The Wiener filter is characterized in the Fourier domain and for additive noise that is independent of the signal it is given by:

$$H_W(u,v) = \frac{S_{aa}(u,v)}{S_{aa}(u,v) + S_{nn}(u,v)}$$

where $S_{aa}(u,v)$ is the power spectral density of an ensemble of random images $\{a[m,n]\}$ and $S_{nn}(u,v)$ is the power spectral density of the random noise. If we have a single image then $S_{aa}(u,v) = |A(u,v)|^2$. In practice it is unlikely that the power spectral density of the uncontaminated image will be available. Because many images have a similar power spectral density that can be modeled by Table 4-T.8, that model can be used as an estimate of $S_{aa}(u,v)$.

A comparison of the five different techniques described above is shown in Figure 49. The Wiener filter was constructed directly from eq. because the image spectrum and the noise spectrum were known. The parameters for the other filters were determined choosing that value (either σ or window size) that led to the minimum *rms*.

a) Noisy image (*SNR*=20 *dB*) **b)** Wiener filter **c)** Gauss filter (σ = 1.0)
rms = 25.7 *rms* = 20.2 *rms* = 21.1

d) Kuwahara filter (5 x 5) **e)** Median filter (3 x 3) **f)** Morph. smoothing (3 x 3)
rms = 22.4 *rms* = 22.6 *rms* = 26.2
Figure 49: Noise suppression using various filtering techniques.
The root mean-square errors (*rms*) associated with the various filters are shown in Figure 49. For this specific comparison, the Wiener filter generates a lower error than any of the other procedures that are examined here. The two linear procedures, Wiener filtering and Gaussian filtering, performed slightly better than the three non-linear alternatives.

Distortion suppression

The model presented above--an image distorted solely by noise--is not, in general, sophisticated enough to describe the true nature of distortion in a digital image. A more realistic model includes not only the noise but also a model for the distortion induced by lenses, finite apertures, possible motion of the camera and/or an object, and so forth. One frequently used model is of an image $a[m,n]$ distorted by a linear, shift-invariant system $h_o[m,n]$ (such as a lens) and then contaminated by noise $\kappa[m,n]$. Various aspects of $h_o[m,n]$ and $\kappa[m,n]$ have been discussed in earlier sections. The most common combination of these is the additive model:

$$c[m,n] = \left(a[m,n] \otimes h_o[m,n]\right) + \kappa[m,n]$$

The restoration procedure that is based on linear filtering coupled to a minimum mean-square error criterion again produces a Wiener filter :

$$H_W(u,v) = \frac{H_o^*(u,v) S_{aa}(u,v)}{\left|H_o(u,v)\right|^2 S_{aa}(u,v) + S_{nn}(u,v)}$$

$$= \frac{H_o^*(u,v)}{\left|H_o(u,v)\right|^2 + \left(S_{nn}(u,v) \big/ S_{aa}(u,v) \right)}$$

Once again $S_{aa}(u,v)$ is the power spectral density of an image, $S_{nn}(u,v)$ is the power spectral density of the noise, and $_o(u,v)$ = F{$h_o[m,n]$}. Examination of this formula for some extreme cases can be useful. For those frequencies where $S_{aa}(u,v) >> S_{nn}(u,v)$, where the signal spectrum dominates the noise spectrum, the Wiener filter is given by $1/_o(u,v)$, the *inverse filter* solution. For those frequencies where $S_{aa}(u,v) << S_{nn}(u,v)$, where the noise spectrum dominates the signal spectrum, the Wiener filter is proportional to $_o^*(u,v)$, the *matched filter* solution. For those frequencies where $_o(u,v) = 0$, the Wiener filter $_W(u,v) = 0$ preventing overflow.
The **Wiener filter** is a solution to the restoration problem based upon the hypothesized use of a linear filter and the minimum mean-square (or *rms*) error criterion. In the example below the image $a[m,n]$ was distorted by a bandpass filter and then white noise was added to achieve an *SNR* = 30 *dB*. The results are shown in Figure 50.

a) Distorted, noisy image **b)** Wiener filter **c)** Median filter (3 x 3)
rms = 108.4 *rms* = 40.9
Figure 50: Noise *and* distortion suppression using the Wiener filter, eq. and the median filter.
The *rms* after Wiener filtering but before contrast stretching was 108.4; after contrast stretching with eq. (77) the final result as shown in Figure 50b has a mean-square error of 27.8. Using a 3 x 3 *median filter* as shown in Figure 50c leads to a *rms* error of 40.9 before contrast stretching and 35.1 after contrast stretching. Although the Wiener filter gives the minimum *rms* error over the set of all *linear* filters, the *non-linear* median filter gives a lower *rms* error. The operation *contrast stretching* is itself a non-linear operation. The "visual quality" of the median filtering result is comparable to the Wiener filtering result. This is due in part to periodic artifacts introduced by the linear filter which are visible in Figure 50b.

Noise

Images acquired through modern sensors may be contaminated by a variety of noise sources. By noise we refer to stochastic variations as opposed to deterministic distortions such as shading or lack of focus. We will assume for this section that we are dealing with images formed from light using modern electro-optics. In particular we will assume the use of modern, charge-coupled device (CCD) cameras where photons produce electrons that are commonly referred to as photoelectrons. Nevertheless, most of the observations we shall make about noise and its various sources hold equally well for other imaging modalities.
While modern technology has made it possible to reduce the noise levels associated with various electro-optical devices to almost negligible levels, one noise source can never be eliminated and thus forms the limiting case when all other noise sources are "eliminated".

Photon Noise

When the physical signal that we observe is based upon light, then the quantum nature of light plays a significant role. A single photon at λ = 500 nm carries an energy of E = h ν = hc/ λ = 3.97 x 10-19 Joules. Modern CCD cameras are sensitive enough to be able to count individual photons. (Camera sensitivity will be discussed in Section 7.2.) The noise problem arises from the fundamentally statistical nature of photon production. We cannot assume that, in a given pixel for two consecutive but independent observation intervals of length T, the same number of photons will be counted. Photon production is governed by the laws of quantum physics which restrict us to talking about an average number of photons within a given observation window. The probability distribution for p photons in an observation window of length T seconds is known to be Poisson:

$$P(p|\rho,T) = \frac{(\rho T)^p e^{-\rho T}}{p!}$$

where ρ is the rate or intensity parameter measured in photons per second. It is critical to understand that even if there were no other noise sources in the imaging chain, the statistical fluctuations associated with photon counting over a finite time interval *T* would still lead to a finite signal-to-noise ratio (*SNR*). If we use the appropriate formula for the *SNR* (eq.), then due to the fact that the average value and the standard deviation are given by:

Poisson process - $average = \rho T$, $\sigma = \sqrt{\rho T}$

we have for the *SNR*:

Photon noise - $SNR = 10\log_{10}(\rho T) \quad dB$

The three traditional assumptions about the relationship between signal and noise do not hold for photon noise:
* photon noise is not independent of the signal;
* photon noise is not Gaussian, and;
* photon noise is not additive.

For very bright signals, where ρT exceeds 10^5, the noise fluctuations due to photon statistics can be ignored if the sensor has a sufficiently high saturation level. This will be discussed further in Section 7.3 and, in particular, eq. .

Thermal Noise

An additional, stochastic source of electrons in a CCD well is thermal energy. Electrons can be freed from the CCD material itself through thermal vibration and then, trapped in the CCD well, be indistinguishable from "true" photoelectrons. By cooling the CCD chip it is possible to reduce significantly the number of "thermal electrons" that give rise to thermal noise or dark current. As the integration time T increases, the number of thermal electrons increases. The probability distribution of thermal electrons is also a Poisson process where the rate parameter is an increasing function of temperature. There are alternative techniques (to cooling) for suppressing dark current and these usually involve estimating the average dark current for the given integration time and then subtracting this value from the CCD pixel values before the A/D converter. While this does reduce the dark current average, it does not reduce the dark current standard deviation and it also reduces the possible dynamic range of the signal.

On-chip Electronic Noise

This noise originates in the process of reading the signal from the sensor, in this case through the field effect transistor (FET) of a CCD chip. The general form of the power spectral density of readout noise is:

Readout noise -

$$S_{nn}(\omega) \propto \begin{cases} \omega^{-\beta} & \omega < \omega_{\min} \quad \beta > 0 \\ k & \omega_{\min} < \omega < \omega_{\max} \\ \omega^{\alpha} & \omega > \omega_{\max} \quad \alpha > 0 \end{cases}$$

where a and β are constants and ω is the (radial) frequency at which the signal is transferred from the CCD chip to the "outside world." At very low readout rates ($\omega < \omega_{\min}$) the noise has a $1/f$ character. Readout noise can be reduced to manageable levels by appropriate readout rates and proper electronics. At very low signal levels (see eq.), however, readout noise can still become a significant component in the overall *SNR* .

KTC Noise

Noise associated with the gate capacitor of an FET is termed *KTC noise* and can be non-negligible. The output RMS value of this noise voltage is given by:

KTC noise (voltage) - $\sigma_{KTC} = \sqrt{\frac{kT}{C}}$

where *C* is the FET gate switch capacitance, *k* is Boltzmann's constant, and *T* is the absolute temperature of the CCD chip measured in K. Using the relationships $Q = C \bullet V = N_{e^-} \bullet e^-$, the output RMS value of the KTC noise expressed in terms of the number of photoelectrons (N_{e^-}) is given by:

KTC noise (electrons) - $\sigma_{N_e} = \frac{\sqrt{kTC}}{e^-}$

where e^- is the electron charge. For *C* = 0.5 pF and *T* = 233 K this gives $N_{e^-} = 252 \text{ electrons}$. This value is a "one time" noise per pixel that occurs during signal readout and is thus independent of the integration time (see Sections 6.1 and 7.7). Proper electronic design that makes use, for example, of correlated double sampling and dual-slope integration can almost completely eliminate KTC noise .

Amplifier Noise

The standard model for this type of noise is additive, Gaussian, and independent of the signal. In modern well-designed electronics, amplifier noise is generally negligible. The most common exception to this is in color cameras where more amplification is used in the blue color channel than in the green channel or red channel leading to more noise in the blue channel. (See also Section 7.6.)

Quantization Noise

Quantization noise is inherent in the amplitude quantization process and occurs in the analog-to-digital converter, ADC. The noise is additive and independent of the signal when the number of levels L >= 16. This is equivalent to B >= 4 bits. (See Section 2.1.) For a signal that has been converted to electrical form and thus has a minimum and maximum electrical value, eq. is the appropriate formula for determining the SNR. If the ADC is adjusted so that 0 corresponds to the minimum electrical value and 2B-1 corresponds to the maximum electrical value then:
Quantization noise - $SNR = 6B + 11 \quad dB$
For *B* >= 8 bits, this means a *SNR* >= 59 dB. Quantization noise can usually be ignored as the total *SNR* of a complete system is typically dominated by the smallest *SNR*. In CCD cameras this is photon noise.

Tools

Certain tools are central to the processing of digital images. These include mathematical tools such as *convolution*, *Fourier analysis*, and *statistical* descriptions, and manipulative tools such as *chain codes* and *run codes*. We will present these tools without any specific motivation. The motivation will follow in later sections.

Convolution

There are several possible notations to indicate the convolution of two (multi-dimensional) signals to produce an output signal. The most common are:

$$c = a \otimes b = a * b$$

We shall use the first form, $c = a \otimes b$, with the following formal definitions.

In 2D continuous space:

$$c(x,y) = a(x,y) \otimes b(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} a(\chi,\zeta)\, b(x-\chi, y-\zeta)\, d\chi\, d\zeta$$

In 2D discrete space:

$$c[m,n] = a[m,n] \otimes b[m,n] = \sum_{j=-\infty}^{+\infty}\sum_{k=-\infty}^{+\infty} a[j,k]\, b[m-j, n-k]$$

Properties of Convolution

There are a number of important mathematical properties associated with convolution.

* Convolution is *commutative*.

$$c = a \otimes b = b \otimes a$$

* Convolution is *associative*.

$$c = a \otimes (b \otimes c) = (a \otimes b) \otimes c = a \otimes b \otimes c$$

* Convolution is *distributive*.

$$c = a \otimes (b + d) = (a \otimes b) + (a \otimes d)$$

where *a*, *b*, *c*, and *d* are all images, either continuous or discrete.

Fourier Transforms

The Fourier transform produces another representation of a signal, specifically a representation as a weighted sum of complex exponentials. Because of Euler's formula:

$$e^{jq} = \cos(q) + j\sin(q)$$

where $j^2 = -1$, we can say that the Fourier transform produces a representation of a (2D) signal as a weighted sum of sines and cosines. The defining formulas for the forward Fourier and the inverse Fourier transforms are as follows. Given an image *a* and its Fourier transform *A*, then the forward transform goes from the spatial domain (either continuous or discrete) to the frequency domain which is always continuous.

Forward - $A = \mathsf{F}\{a\}$

The inverse Fourier transform goes from the frequency domain back to the spatial domain.

Inverse - $a = \mathsf{F}^{-1}\{A\}$

The Fourier transform is a unique and invertible operation so that:

$$a = \mathsf{F}^{-1}\{\mathsf{F}\{a\}\} \qquad and \qquad A = \mathsf{F}\{\mathsf{F}^{-1}\{A\}\}$$

The specific formulas for transforming back and forth between the spatial domain and the frequency domain are given below.

In 2D continuous space:

Forward - $$A(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} a(x,y)e^{-j(ux+vy)}\,dxdy$$

Inverse - $$a(x,y) = \frac{1}{4\pi^2}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} A(u,v)e^{+j(ux+vy)}\,dudv$$

In 2D discrete space:

Forward - $$A(\Omega,\Psi) = \sum_{m=-\infty}^{+\infty}\sum_{n=-\infty}^{+\infty} a[m,n]e^{-j(\Omega m+\Psi n)}$$

Inverse - $$a[m,n] = \frac{1}{4\pi^2}\int_{-\pi}^{+\pi}\int_{-\pi}^{+\pi} A(\Omega,\Psi)e^{+j(\Omega m+\Psi n)}\,d\Omega\,d\Psi$$

Properties of Fourier Transforms

There are a variety of properties associated with the Fourier transform and the inverse Fourier transform. The following are some of the most relevant for digital image processing.

* The Fourier transform is, in general, a complex function of the real frequency variables. As such the transform can be written in terms of its magnitude and phase.

$$A(u,v) = |A(u,v)|e^{j\varphi(u,v)} \qquad A(\Omega,\Psi) = |A(\Omega,\Psi)|e^{j\varphi(\Omega,\Psi)}$$

* A 2D signal can also be complex and thus written in terms of its magnitude and phase.

$$a(x,y) = |a(x,y)|e^{j\vartheta(x,y)} \qquad a[m,n] = |a[m,n]|e^{j\vartheta[m,n]}$$

* If a 2D signal is real, then the Fourier transform has certain symmetries.

$$A(u,v) = A^*(-u,-v) \qquad A(\Omega,\Psi) = A^*(-\Omega,-\Psi)$$

The symbol (*) indicates complex conjugation. For real signals eq. leads directly to:

$$|A(u,v)| = |A(-u,-v)| \qquad \varphi(u,v) = -\varphi(-u,-v)$$
$$|A(\Omega,\Psi)| = |A(-\Omega,-\Psi)| \qquad \varphi(\Omega,\Psi) = -\varphi(-\Omega,-\Psi)$$

* If a 2D signal is real and even, then the Fourier transform is real and even.

$$A(u,v) = A(-u,-v) \qquad A(\Omega,\Psi) = A(-\Omega,-\Psi)$$

* The Fourier and the inverse Fourier transforms are linear operations.

$$\mathcal{F}\{w_1 a + w_2 b\} = \mathcal{F}\{w_1 a\} + \mathcal{F}\{w_2 b\} = w_1 A + w_2 B$$
$$\mathcal{F}^{-1}\{w_1 A + w_2 B\} = \mathcal{F}^{-1}\{w_1 A\} + \mathcal{F}^{-1}\{w_2 B\} = w_1 a + w_2 b$$

Where *a* and *b* are 2D signals (images) and w_1 and w_2 are arbitrary, complex constants.

* The Fourier transform in discrete space, $A(\Omega,\Psi)$, is periodic in both Ω and Ψ. Both periods are 2π.

$$A(\Omega + 2\pi j, \Psi + 2\pi k) = A(\Omega,\Psi) \qquad j,k \text{ integers}$$

* The energy, *E*, in a signal can be measured either in the spatial domain or the frequency domain. For a signal with finite energy:

Parseval's theorem (2D continuous space):

$$E = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} |a(x,y)|^2 \, dxdy = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} |A(u,v)|^2 \, dudv$$

Parseval's theorem (2D discrete space):

$$E = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} |a[m,n]|^2 = \frac{1}{4\pi^2} \int_{-\pi}^{+\pi}\int_{-\pi}^{+\pi} |A(\Omega,\Psi)|^2 \, d\Omega d\Psi$$

This "signal energy" is not to be confused with the physical energy in the phenomenon that produced the signal. If, for example, the value *a*[*m*,*n*] represents a photon count, then the *physical* energy is proportional to the amplitude, *a*, and not the square of the amplitude. This is generally the case in video imaging.

* Given three, multi-dimensional signals *a*, *b*, and *c* and their Fourier transforms *A*, *B*, and *C*:

$$c = a \otimes b \quad \overset{F}{\leftrightarrow} \quad C = A \bullet B$$

and

$$c = a \bullet b \quad \overset{F}{\leftrightarrow} \quad C = \frac{1}{4\pi^2} A \otimes B$$

In words, convolution in the spatial domain is equivalent to multiplication in the Fourier (frequency) domain and vice-versa. This is a central result which provides not only a methodology for the implementation of a convolution but also insight into how two signals interact with each other--under convolution--to produce a third signal. We shall make extensive use of this result later.

* If a two-dimensional signal *a*(*x*,*y*) is scaled in its spatial coordinates then:

$$\begin{aligned} &If \quad a(x,y) \rightarrow a(M_x \bullet x, M_y \bullet y) \\ &Then \quad A(u,v) \rightarrow A\left(u/M_x , v/M_y \right) \Big/ |M_x \bullet M_y| \end{aligned}$$

* If a two-dimensional signal *a*(*x*,*y*) has Fourier spectrum *A*(*u*,*v*) then:

$$A(u=0, v=0) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} a(x,y) dxdy$$

$$a(x=0, y=0) = \frac{1}{4\pi^2} \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} A(u,v) dxdy$$

* If a two-dimensional signal *a*(*x*,*y*) has Fourier spectrum *A*(*u*,*v*) then:

$$\frac{\partial a(x,y)}{\partial x} \overset{F}{\leftrightarrow} juA(u,v) \qquad \frac{\partial a(x,y)}{\partial y} \overset{F}{\leftrightarrow} jvA(u,v)$$

$$\frac{\partial^2 a(x,y)}{\partial x^2} \overset{F}{\leftrightarrow} -u^2 A(u,v) \qquad \frac{\partial^2 a(x,y)}{\partial y^2} \overset{F}{\leftrightarrow} -v^2 A(u,v)$$

Importance of phase and magnitude

Equation indicates that the Fourier transform of an image can be complex. This is illustrated below in Figures 4a-c. Figure 4a shows the original image *a*[*m*,*n*], Figure 4b the magnitude in a scaled form as log(|*A*(Ω, Ψ)|), and Figure 4c the phase φ(Ω, Ψ).

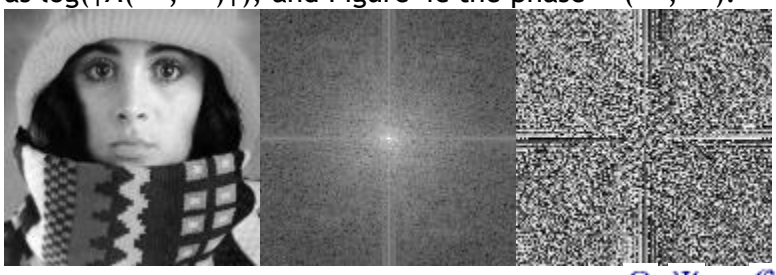


Figure 4a Figure 4b Figure 4c Original log(|A(Ω, Ψ)|) φ(Ω, Ψ)

Both the magnitude and the phase functions are necessary for the complete reconstruction of an image from its Fourier transform. Figure 5a shows what happens when Figure 4a is restored solely on the basis of the magnitude information and Figure 5b shows what happens when Figure 4a is restored solely on the basis of the phase information.

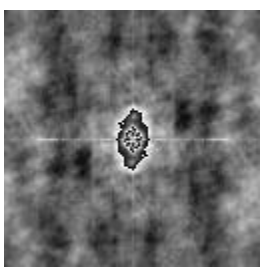


Figure 5a

Figure 5b

φ(Ω, Ψ) = 0 |A(Ω, Ψ)| = *constant*

Neither the magnitude information nor the phase information is sufficient to restore the image. The magnitude-only image (Figure 5a) is unrecognizable and has severe dynamic range problems. The phase-only image (Figure 5b) is barely recognizable, that is, severely degraded in quality.

Circularly symmetric signals

An arbitrary 2D signal *a* (*x*,*y*) can always be written in a polar coordinate system as *a*(*r*, ϑ). When the 2D signal exhibits a circular symmetry this means that:

$$a(x,y) = a(r,\theta) = a(r)$$

Where $r^2 = x^2 + y^2$ and $\tan\vartheta = y/x$. As a number of physical systems such as lenses exhibit circular symmetry, it is useful to be able to compute an appropriate Fourier representation.

The Fourier transform A (u, v) can be written in polar coordinates $A(\omega_r, \xi)$ and then, for a circularly symmetric signal, rewritten as a *ankel transform*:

$$A(u,v) = \mathsf{F}\{a(x,y)\} = 2\pi\int_0^\infty a(r)J_o(\omega_r r)rdr = A(\omega_r)$$

Where $\omega_r^2 = u^2 + v^2$ and $\tan\xi = v/u$ and $J_o(*)$ is a Bessel function of the first kind of order zero.

The inverse *ankel transform* is given by:

$$a(r) = \frac{1}{2\pi}\int_0^\infty A(\omega_r)J_o(\omega_r r)\omega_r d\omega_r$$

The Fourier transform of a circularly symmetric 2D signal is a function of only the radial frequency, ω_r. The dependence on the angular frequency, ξ, has vanished. Further, if $a(x,y) = a(r)$ is real, then it is automatically even due to the circular symmetry. According to equation , $A(\omega_r)$ will then be real and even.

Examples of 2D signals and transforms

Table 4 shows some basic and useful signals and their 2D Fourier transforms. In using the table entries in the remainder of this chapter we will refer to a spatial domain term as the *point spread function* (*PSF*) or the *2D impulse response* and its Fourier transforms as the *optical transfer function* (*OTF*) or simply *transfer function*. Two standard signals used in this table are $u(*)$, the unit step function, and $J_1(*)$, the Bessel function of the first kind. Circularly symmetric signals are treated as functions of r as in eq. .

Statistics

In image processing it is quite common to use simple statistical descriptions of images and sub-images. The notion of a statistic is intimately connected to the concept of a probability distribution, generally the distribution of signal amplitudes. For a given region--which could conceivably be an entire image--we can define the probability *distribution* function of the brightness in that region and the probability *density* function of the brightness in that region. We will assume in the discussion that follows that we are dealing with a digitized image *a*[*m*,*n*].

Probability distribution function of the brightness

The probability distribution function, *P*(*a*), is the probability that a brightness chosen from the region is less than or equal to a given brightness value *a*. As *a* increases from $-\infty$ to $+\infty$, *P*(*a*) increases from 0 to 1. *P*(*a*) is monotonic, non-decreasing in *a* and thus d*P*/d*a* >= 0.

Probability density function of the brightness

The probability that a brightness in a region falls between *a* and *a*+Δ*a*, given the probability distribution function *P*(*a*), can be expressed as *p*(*a*)Δ*a* where *p*(*a*) is the probability density function:

$$p(a)\Delta a = \left(\frac{dP(a)}{da}\right)\Delta a$$

T.1 *Rectangle*	$R_{a,b}(x,y) = \frac{1}{4ab}u(a^2-x^2)u(b^2-y^2)$	$\overset{F}{\leftrightarrow}$	$\left(\frac{\sin(2\pi a f_x)}{\pi a f_x}\right)\left(\frac{\sin(2\pi b f_y)}{\pi b f_y}\right)$
	picture 1		picture 2

T.2 *Pyramid*	$R_{a,b}(x,y) \otimes R_{a,b}(x,y)$	$\overset{F}{\leftrightarrow}$	$\left(\frac{\sin(2\pi a f_x)}{\pi a f_x}\frac{\sin(2\pi b f_y)}{\pi b f_y}\right)^2$
	picture 1		picture 2

T.3 *Pill Box*	$P_a(r) = \frac{u(a^2-r^2)}{\pi a^2}$	$\overset{F}{\leftrightarrow}$	$2\frac{J_1(2\pi a f)}{\pi a f}$
	picture 1		picture 2

T.4 *Cone*	$P_a(r) \otimes P_a(r)$	$\overset{F}{\leftrightarrow}$	$4\left(\frac{J_1(2\pi a f)}{\pi a f}\right)^2$

	picture 1		picture 2

T.5 *Airy PSF*	$PSF(r) = \frac{1}{\pi}\left(\frac{J_1(\omega_c r/2)}{r}\right)^2$	$\overset{F}{\leftrightarrow}$	$\frac{2}{\pi}\left(\cos^{-1}\left(\frac{f}{f_c}\right) - \left(\frac{f}{f_c}\right)\sqrt{1-\left(\frac{f}{f_c}\right)^2}\right) u\left(f_c^2 - f^2\right)$
	picture 1		picture 2

T.6 *Gaussian*	$g_{2D}(r,\sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right)$	$\overset{F}{\leftrightarrow}$	$G_{2D}(f,\sigma) = \exp\left(-2\pi^2 f^2 \sigma^2\right)$
	picture 1		picture 2

T.7 *Peak*	$\frac{1}{r}$	$\overset{F}{\leftrightarrow}$	$\frac{1}{f}$
	picture 1		picture 2

T.8 *Exponential Decay*	e^{-ar}	$\overset{F}{\leftrightarrow}$	$2\pi a / \left(\omega^2 + a^2\right)^{3/2}$
	picture 1		picture 2

Table 4: 2D Images and their Fourier Transforms

Because of the monotonic, non-decreasing character of *P*(*a*) we have that:

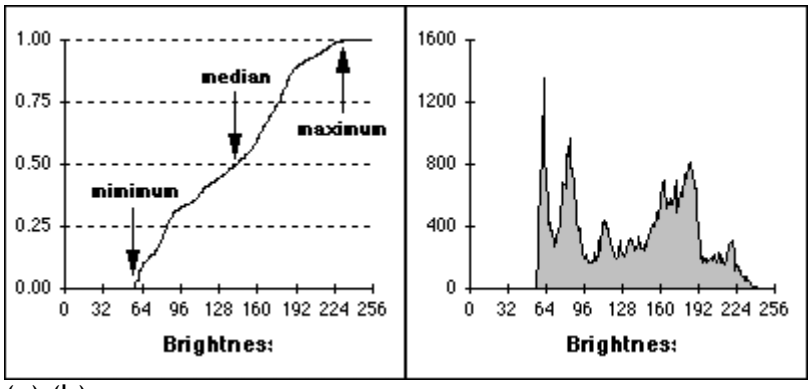
$$p(a) \geq 0 \quad \text{and} \quad \int_{-\infty}^{+\infty} p(a)\,da = 1$$

For an image with quantized (integer) brightness amplitudes, the interpretation of Δa is the width of a brightness interval. We assume constant width intervals. The brightness probability *density* function is frequently estimated by counting the number of times that each brightness occurs in the region to generate a *histogram, h*[*a*]. The histogram can then be normalized so that the total area under the histogram is 1 (eq.). Said another way, the *p*[*a*] for a region is the normalized count of the number of pixels, Λ, in a region that have quantized brightness *a*:

$$p[a] = \frac{1}{\Lambda} h[a] \quad \text{with} \quad \Lambda = \sum_a h[a]$$

The brightness probability *distribution* function for the image shown in Figure 4a is shown in Figure 6a. The (unnormalized) brightness histogram of Figure 4a which is proportional to the estimated brightness

probability density function is shown in Figure 6b. The height in this histogram corresponds to the number of pixels with a given brightness.



(a) (b)

Figure 6: (a) Brightness distribution function of Figure 4a with *minimum*, *median*, and *maximum* indicated. See text for explanation. (b) Brightness histogram of Figure 4a.

Both the distribution function and the histogram as measured from a region are a statistical description of that region. It must be emphasized that both *P*[*a*] and *p*[*a*] should be viewed as *estimates* of true distributions when they are computed from a specific region. That is, we view an image and a specific region as one realization of the various random processes involved in the formation of that image and that region. In the same context, the statistics defined below must be viewed as estimates of the underlying parameters.

Average

The average brightness of a region is defined as the *sample mean* of the pixel brightness within that region. The average, m_a, of the brightness over the Λ pixels within a region ($\Re$) is given by:

$$m_a = \frac{1}{\Lambda} \sum_{(m,n)\in\Re} a[m,n]$$

Alternatively, we can use a formulation based upon the (unnormalized) brightness histogram, $h(a) = \Lambda * p(a)$, with discrete brightness values *a*. This gives:

$$m_a = \frac{1}{\Lambda} \sum_{a} a \bullet h[a]$$

The average brightness, m_a, is an estimate of the mean brightness, u_a, of the underlying brightness probability distribution.

Standard deviation

The *unbiased estimate* of the standard deviation, s_a, of the brightness within a region ($\Re$) with Λ pixels is called the *sample standard deviation* and is given by:

$$s_a = \sqrt{\frac{1}{\Lambda - 1}\sum_{m,n\in\Re}\left(a[m,n] - m_a\right)^2}$$

$$= \sqrt{\frac{\sum_{m,n\in\Re} a^2[m,n] - \Lambda m_a^2}{\Lambda - 1}}$$

Using the histogram formulation gives: $s_a = \sqrt{\frac{\left(\sum_a a^2 \bullet h[a]\right) - \Lambda \bullet m_a^2}{\Lambda - 1}}$

The standard deviation, s_a, is an estimate of σ_a of the underlying brightness probability distribution.

Coefficient-of-variation

The dimensionless *coefficient-of-variation*, *CV*, is defined as: $CV = \frac{s_a}{m_a} \times 100\%$

Percentiles

The percentile, *p%*, of an *unquantized* brightness distribution is defined as that value of the brightness *a* such that: $P(a) = p\%$
Or equivalently

$$\int_{-\infty}^{a} p(\alpha)d\alpha = p\%$$

Three special cases are frequently used in digital image processing.

* 0% the *minimum* value in the region
* 50% the *median* value in the region
* 100% the *maximum* value in the region

All three of these values can be determined from Figure 6a.

Mode

The mode of the distribution is the most frequent brightness value. There is no guarantee that a mode exists or that it is unique.

Signal to Noise ratio

The signal-to-noise ratio, *SNR*, can have several definitions. The noise is characterized by its standard deviation, s_n. The characterization of the signal can differ. If the signal is known to lie between two boundaries, $a_{min} <= a <= a_{max}$, then the *SNR* is defined as:

Bounded signal -

$$SNR = 20\log_{10}\left(\frac{a_{\max} - a_{\min}}{s_n}\right)\ dB$$

If the signal is not bounded but has a statistical distribution then two other definitions are known:

Stochastic signal - S & N inter-dependent

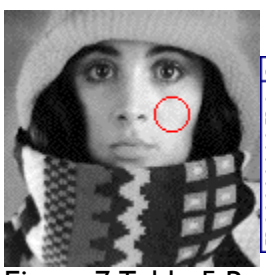
$$SNR = 20\log_{10}\left(\frac{m_a}{s_n}\right)\ dB$$

S & N independent

$$SNR = 20\log_{10}\left(\frac{s_a}{s_n}\right)\ dB$$

where m_a and s_a are defined above.

The various statistics are given in Table 5 for the image and the region shown in Figure 7.



Statistic	*Image*	*ROI*
Average	137.7	219.3
Standard Deviation	49.5	4.0
Minimum	56	202
Median	141	220
Maximum	241	226
Mode	62	220
SNR (db)	*NA*	33.3

Figure 7 Table 5 Region is the interior of the circle. Statistics from Figure 7

A *SNR* calculation for the *entire* image based on eq. is not directly available. The variations in the image brightness that lead to the large value of *s* (=49.5) are not, in general, due to noise but to the variation in local information. With the help of the region there is a way to estimate the *SNR*. We can use the $s^{\Re}$ (=4.0) and the dynamic range, a_{max} - a_{min}, for the image (=241-56) to calculate a global *SNR* (=33.3 dB). The underlying assumptions are that 1) the signal is approximately constant in that region and the variation in the region is therefore due to noise, and, 2) that the noise is the same over the entire image with a standard deviation given by $s_n = s^{\Re}$.

Contour Representations

When dealing with a region or object, several compact representations is available that can facilitate manipulation of and measurements on the object. In each case we assume that we begin with an image representation of the object as shown in Figure 8a,b. Several techniques exist to represent the region or object by describing its contour.

Chain code

This representation is based upon the work of Freeman. We follow the contour in a clockwise manner and keep track of the directions as we go from one contour pixel to the next. For the standard implementation of the chain code we consider a contour pixel to be an object pixel that has a background (non-object) pixel as one or more of its 4-connected neighbors. See Figures 3a and 8c.

The codes associated with eight possible directions are the chain codes and, with *x* as the current contour pixel position, the codes are generally defined as:

$$\text{Chain codes} = \begin{matrix} 3 & 2 & 1 \\ 4 & x & 0 \\ 5 & 6 & 7 \end{matrix}$$

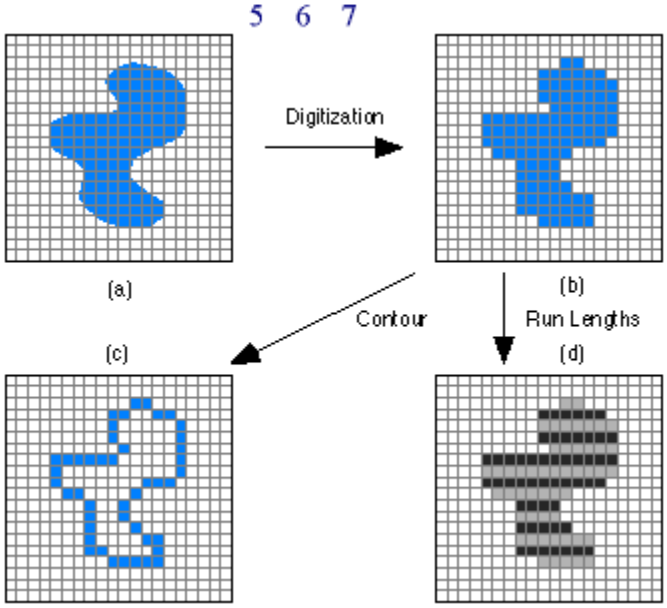


Figure 8: Region (shaded) as it is transformed from (a) continuous to (b) discrete form and then considered as a (c) contour or (d) run lengths illustrated in alternating colors.

Chain code properties

* Even codes {0,2,4,6} correspond to horizontal and vertical directions; odd codes {1,3,5,7} correspond to the diagonal directions.

* Each code can be considered as the angular direction, in multiples of 45deg., that we must move to go from one contour pixel to the next.

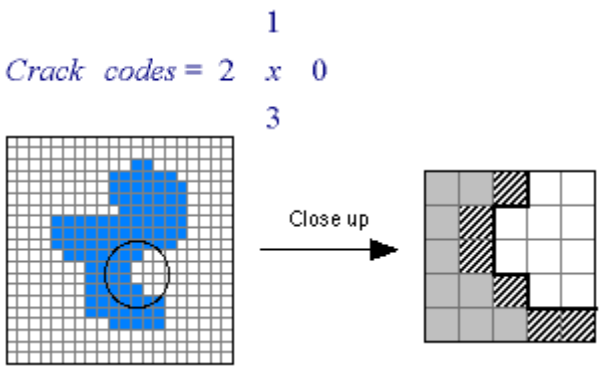
* The absolute coordinates [*m*,*n*] of the first contour pixel (e.g. top, leftmost) together with the chain code of the contour represent a complete description of the discrete region contour.

* When there is a change between two consecutive chain codes, then the contour has changed direction. This point is defined as a *corner*.

Crack code

An alternative to the chain code for contour encoding is to use neither the contour pixels associated with the object nor the contour pixels associated with background but rather the line, the "crack", in between. This is illustrated with an enlargement of a portion of Figure 8 in Figure 9.

The "crack" code can be viewed as a chain code with four possible directions instead of eight.



(a) (b)

Figure 9: (a) Object including part to be studied. (b) Contour pixels as used in the chain code are diagonally shaded. The "crack" is shown with the thick black line.

The chain code for the enlarged section of Figure 9b, from top to bottom, is {5,6,7,7,0}. The crack code is {3,2,3,3,0,3,0,0}.

Run codes

A third representation is based on coding the consecutive pixels along a row--a run--that belong to an object by giving the starting position of the run and the ending position of the run. Such runs are illustrated in Figure 8d. There are a number of alternatives for the precise definition of the positions. Which alternative should be used depends upon the application and thus will not be discussed here.