3 Basic Image Processing Operations

3.1 Overview

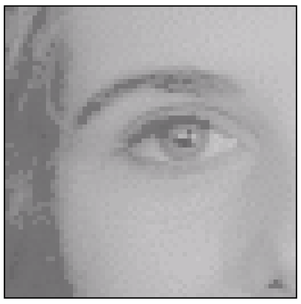
We shall now start to process digital images. First, we shall describe the brightness variation in an image using its histogram. We shall then look at operations which manipulate the image so as to change the histogram, processes that shift and scale the result (making the image brighter or dimmer, in different ways). We shall also consider thresholding techniques that turn an image from grey level to binary. These are called single point operations. After, we shall move to group operations where the group is those points found inside a template. Some of the most common operations on the groups of points are statistical, providing images where each point is the result of, say, averaging the neighbourhood of each point in the original image. We shall see how the statistical operations can reduce noise in the image, which is of benefit to the feature extraction techniques to be considered later. As such, these basic operations are usually for pre-processing for later feature extraction or to improve display quality.

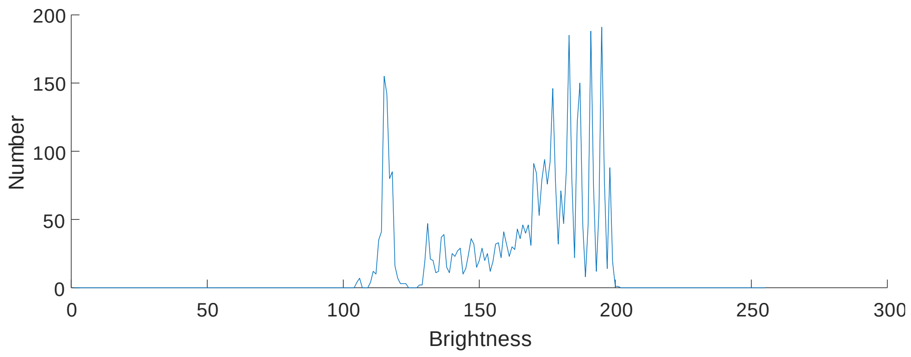
Main topic	Sub topics	Main points
Image Description	Portray **variation** in image brightness content as a graph/**histogram**.	*Histograms*, image *contrast*.
Point Operations	Calculate **new** image points as a **function** of the point at the same place in the original image. The functions can be **mathematical**, or can be computed from the image itself and will change the image's histogram. Finally, **thresholding** turns an image from **grey** level to a **binary** (black and white) representation.	*Histogram* manipulation; *intensity mapping*: addition, inversion, scaling, logarithm, exponent. *Intensity normalisation*; *histogram equalisation*. *Thresholding* and *optimal thresholding*.
Group Operations	Calculate new image points as a function of **neighbourhood** of the point at the same place in the original image. The functions can be **statistical** including: mean (average); median and mode. **Advanced** filtering techniques, including feature preservation.	*Template convolution* (inc. frequency domain implementation). *Statistical operators*: *direct averaging*, *median* filter, and *mode* filter. *Non-local means*, a*nisotropic diffusion*, and *bilateral filter* for image smoothing. Other operators: *force field* and *image ray* transforms.
Image morphology and operators	**Morphological** operators process an image according to **shape**, starting with binary and moving to grey level operations.	*Mathematical morphology*: *hit or miss transform*, *erosion*, *dilation* (inc. grey level operators) and *Minkowski* operators.

Table 3.1 Overview of Chapter 3

3.2 Histograms

The intensity *histogram* shows how individual brightness levels are occupied in an image; the *image contrast* is measured by the range of brightness levels. The histogram plots the number of pixels with a particular brightness level against the brightness level. For 8-bit pixels, the brightness ranges from zero (black) to 255 (white). Figure 3.1 shows an image of an eye and its histogram. The histogram, Figure 3.1(b), shows that not all the grey levels are used and the lowest and highest intensity levels are close together, reflecting moderate **contrast**. The histogram has a region between 100 and 120 brightness values that contains the dark portions of the image, such as the hair (including the eyebrow) and the eye's iris. The brighter points relate mainly to the skin. If the image was darker, overall, the histogram would be concentrated towards black. If the image was brighter, but with lower contrast, then the histogram would be thinner and concentrated near the whiter brightness levels.


(a) image of an eye



(b) histogram of eye image

Figure 3.1 An Image and its Histogram

This histogram shows us that we have not used all available grey levels. Accordingly, we can stretch the image to use them all, and the image would become clearer. This is essentially cosmetic attention to make the image's appearance better. Making the appearance better, especially in view of later processing, is the focus of many basic image processing operations, as will be covered in this Chapter. The histogram can also reveal if there is much noise in the image, if the ideal histogram is known. We might want to remove this noise, not only to improve the appearance of the image, but to ease the task of (and to present the target better for) later feature extraction techniques. This Chapter concerns these basic operations which can improve the appearance and quality of images.

To implement a histogram operator, count up the number of image points that have an intensity at a particular value and store them in a vector (which initially is null). In Matlab the operator `histogram` returns the histogram in Figure 3.1(b).

3.3 Point Operators

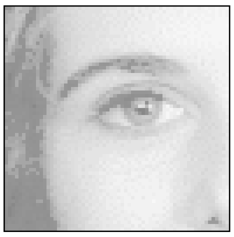
3.3.1 Basic Point Operations

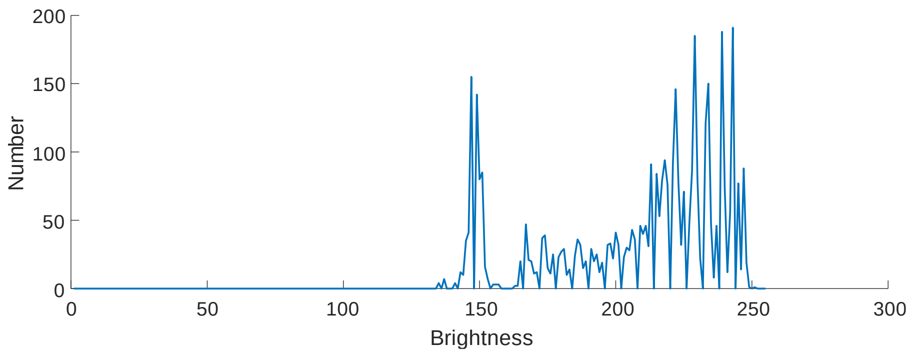
The most basic operations in image processing are point operations where each pixel value is replaced with a new value obtained from the old one. If we want to increase the brightness to stretch the contrast we can simply multiply all pixel values by a scalar, say by 2 to double the range.

Conversely, to reduce the contrast (though this is not usual) we can divide all point values by a scalar. If the overall brightness is controlled by a *level*, *l*, (e.g. the brightness of global light) and the range is controlled by a *gain*, *k*, the brightness of the points in a new picture, **N**, can be related to the brightness in old picture, **O**, by:

$$\mathbf{N}_{x,y} = \exp\left(\mathbf{O}_{x,y}\right) + l \qquad \forall x, y \in 1, N \tag{3.1}$$

This is a point operator that replaces the brightness at points in the picture according to a linear brightness relation. The level controls overall brightness and is the minimum value of the output picture. The gain controls the contrast, or range, and if the gain is greater than unity, the output range will be increased, this process is illustrated in Figure 3.2. So the image of the eye, processed by k = 1.2 and l = 10 will become brighter, Figure 3.2(a), and with better contrast, though in this case the brighter points are mostly set near to white (255). These factors can be seen in its histogram, Figure 3.2(b).

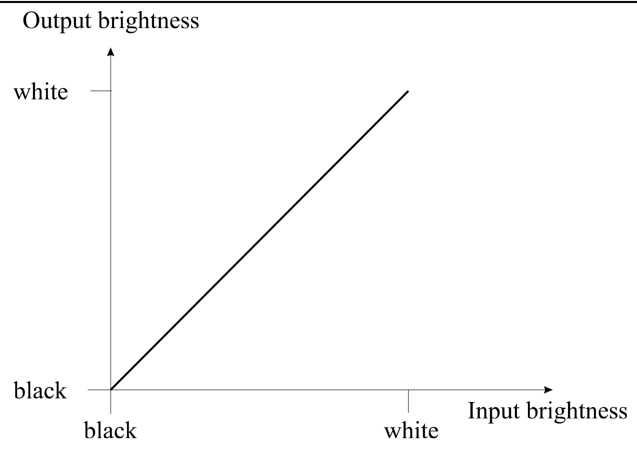

(a) image of brighter eye



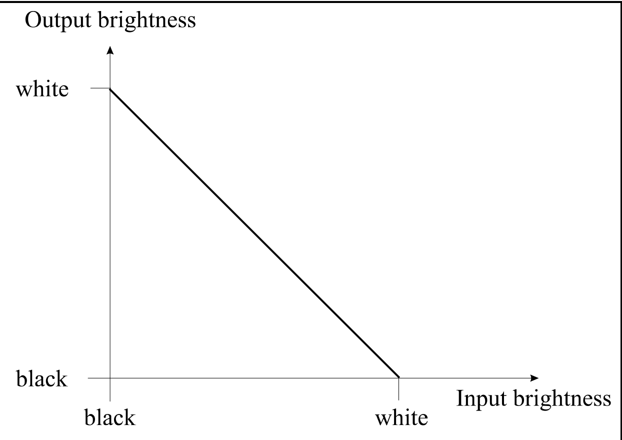
(b) histogram of brighter eye

Figure 3.2 Brightening an Image

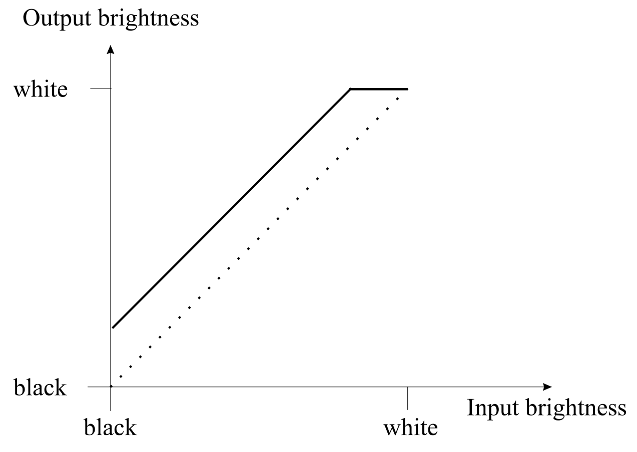
The basis of the implementation of point operators was given earlier, for inversion in Code 1.2. The stretching process can be displayed as a mapping between the input and output ranges, according to the specified relationship, as in Figure 3.3. Figure 3.3(a) is a mapping where the output is a direct copy of the input (this relationship is the dotted line in Figures 3.3(c) and (d)); Figure 3.3(b) is the mapping for brightness **inversion** where dark parts in an image become bright and vice versa. Figure 3.3(c) is the mapping for **addition** and Figure 3.3(d) is the mapping for **multiplication** (or **division,** if the slope was less than that of the input). In these mappings, if the mapping produces values that are smaller than the expected minimum (say negative when zero represents black), or larger than a specified maximum, then a *clipping* process can be used to set the output values to a chosen level. For example, if the relationship between input and output aims to produce output points with intensity value greater than 255, as used for white, the output value can be set to white for these points, as it is in Figure 3.3(c).



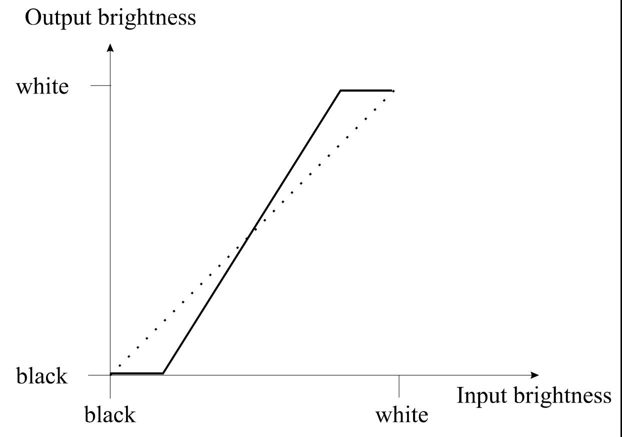
(a) copy



(b) brightness inversion



(c) brightness addition



(d) brightness scaling by multiplication

Figure 3.3 Intensity Mappings

Finally, rather than simple multiplication we can use arithmetic functions such as logarithm to reduce the range or exponent to increase it. This can be used, say, to equalise the response of a camera, or to compress the range of displayed brightness levels. If the camera has a known exponential performance, and outputs a value for brightness which is proportional to the exponential of the brightness of the corresponding point in the scene of view, the application of a *logarithmic point operator* will restore the original range of brightness levels. The effect of replacing brightness by a scaled version of its natural logarithm (implemented as $\mathbf{N}_{x,y} = 20\ln(100\mathbf{O}_{x,y})$) is shown in Figure 3.4(a); the effect of a scaled version of the exponent (implemented as $\mathbf{N}_{x,y} = 20\exp(\mathbf{O}_{x,y}/100)$) is shown in Figure 3.4(b). The scaling factors were chosen to ensure that the resulting image can be displayed since the logarithm or exponent greatly reduce or magnify, pixel values, respectively. This can be seen in the results: Figure 3.4(a) is dark with a small range of brightness levels whereas Figure 3.4(b) is much brighter, with greater contrast. Naturally, application of the logarithmic point operator will change any **multiplicative** changes in brightness to become **additive**. As such, the logarithmic operator can find application in reducing the effects of multiplicative intensity change. The logarithm operator is often used to compress Fourier transforms, for display purposes. This is because the d.c. component can be very large, or the contrast too large, to allow the other points to be seen.

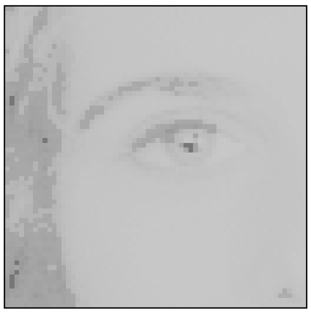
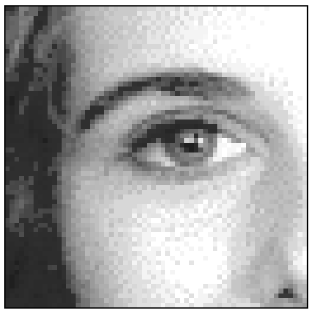
	
(a) logarithmic compression	**(b)** exponential expansion

Figure 3.4 Applying Exponential and Logarithmic Point Operators

In hardware, point operators can be implemented using look-up-tables (LUTs). LUTs give an output that is programmed, and stored, in a table entry that corresponds to a particular input value. If the brightness response of the camera is known, it is possible to pre-program a LUT to make the camera response equivalent to a uniform or flat response across the range of brightness levels (in software, this can be implemented as a CASE function).

3.3.2 Histogram Normalisation

Popular techniques to stretch the range of intensities include *histogram* (*intensity*) *normalisation*. Here, the original histogram is stretched, and shifted, to cover all the 256 available levels. If the original histogram of old picture **O** starts at $\mathbf{O}min$ and extends up to $\mathbf{O}max$ brightness levels, then we can scale up the image so that the pixels in the new picture **N** lie between a minimum output level $\mathbf{N}min$ and a maximum level $\mathbf{N}max$, simply by scaling up the input intensity levels according to:

$$\mathbf{N}_{x,y} = \frac{\mathbf{N}max - \mathbf{N}min}{\mathbf{O}max - \mathbf{O}min} \times \left(\mathbf{O}_{x,y} - \mathbf{O}min\right) + \mathbf{N}min \qquad \forall x, y \in 1, N \qquad (3.2)$$

Code 3.1 gives a Python implementation of intensity normalisation (`normalise.py`), which appears to mimic Matlab's `imagesc` function. The function uses an output ranging from $\mathbf{N}min = 0$ to $\mathbf{N}max = 255$. This is scaled by the input range that is determined from the maximum and minimum values are returned by the `imageMaxMin` operator. Each point in the picture is then scaled as in Equation 3.2 and the `int` function ensures an integer output.

```
# Set utility functions
from ImageSupport import imageReadL, showImageL, createImageL, imageMaxMin
from PlotSupport import plotHistogram

# Set utility to compute histogram
from ImageOperatorsUtilities import computeHistogram

# Iteration
from timeit import itertools

'''Parameters:
    pathToDir = Input image directory
    imageName = Input image name'''
pathToDir = "../../Images/Chapter3/Input/"
imageName = "Horse.png"

# Read image into array
```

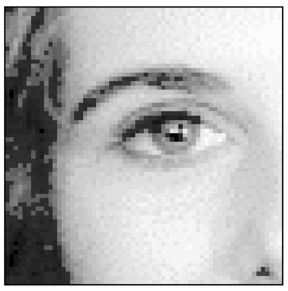
```
ipImage, width, height  = imageReadL(pathToDir + imageName)

# Show input image
showImageL(inputImage)

# Create image to store the normalization
opNormalizedImage = createImageL(width, height)

# Maximum and range
maximum, minimum = imageMaxMin(inputImage)
brightRange = float(maximum - minimum)

# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):

    ipValue = int(ipImage[y,x])

    # Normalize the pixel value according to the range
    opNormalizedImage[y,x] = ((ipValue - minimum) * 255.0 / brightRange)

# Compute histogram
histogramNormalizedImage = computeHistogram(opNormalizedImage)

# Show output image and plot histogram
showImageL(opNormalizedImage)
plotHistogram(histogramNormalizedImage)
```

Code 3.1 Intensity Normalisation

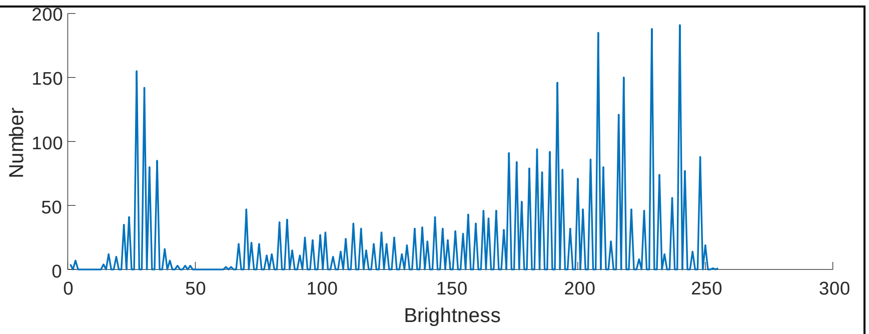
The process is illustrated in Figure 3.5, and can be compared with the original image and histogram in Figure 3.1. An intensity normalised version of the eye image is shown in Figure 3.5(a) which now has better contrast and appears better to the human eye. Its histogram, Figure 3.5(b), shows that the intensity now ranges across all available levels (there is actually one black pixel!).

3.3.3 Histogram Equalisation

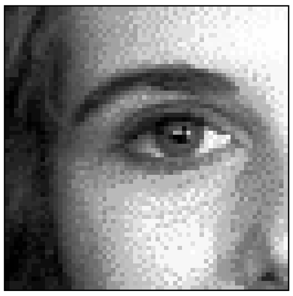
Histogram equalisation is a **non-linear** process aimed to highlight image brightness in a way particularly suited to human visual analysis. Histogram equalisation aims to change a picture in such a way as to produce a picture with a **flatter** histogram, where all levels are equiprobable. In order to develop the operator, we can first inspect the histograms. For a range of M levels then the histogram plots the points per level against level. For the input (old) and the output (new) image, the number of points per level is denoted as $\mathbf{O}(l)$ and $\mathbf{N}(l)$ (for $0<l<M$), respectively. For square images, there are N^2 points in the input and the output image, so the sum of points per level in each should be equal:



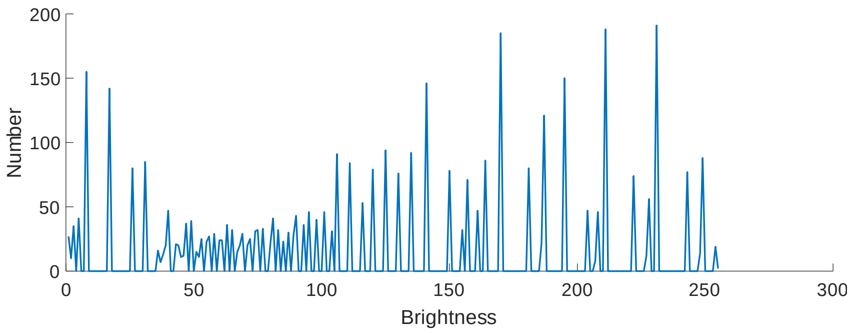
(a) intensity normalised eye



(b) histogram of intensity normalised eye



(c) histogram equalised eye



(d) histogram of histogram equalised eye

Figure 3.5 Illustrating Intensity Normalisation and Histogram Equalisation

$$\sum_{l=0}^{M} \mathbf{O}(l) = \sum_{l=0}^{M} \mathbf{N}(l) \tag{3.3}$$

Also, this should be the same for an arbitrarily chosen level *p*, since we are aiming for an output picture with a uniformly flat histogram. So the cumulative histogram up to level *p* should be transformed to cover up to the level *q* in the new histogram:

$$\sum_{l=0}^{p} \mathbf{O}(l) = \sum_{l=0}^{q} \mathbf{N}(l) \tag{3.4}$$

Since the output histogram is uniformly flat, the cumulative histogram up to level *p* should be a fraction of the overall sum. So the number of points per level in the output picture is the ratio of the number of points to the range of levels in the output image:

$$\mathbf{N}(l) = \frac{N^2}{\mathbf{N}max - \mathbf{N}min} \tag{3.5}$$

So the cumulative histogram of the output picture is:

$$\sum_{l=0}^{q} \mathbf{N}(l) = q \times \frac{N^2}{\mathbf{N}max - \mathbf{N}min} \tag{3.6}$$

By Equation 3.4 this is equal to the cumulative histogram of the input image, so:

$$q \times \frac{N^2}{\mathbf{N}max - \mathbf{N}min} = \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.7}$$

This gives a mapping for the output pixels at level *q*, from the input pixels at level *p* as:

$$q = \frac{\mathbf{N}max - \mathbf{N}min}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.8}$$

This gives a mapping function that provides an output image that has an approximately flat histogram. The mapping function is given by phrasing Equation 3.8 as an equalising function (*E*) of the level (*q)* and the image (**O**) as

$$E(q, \mathbf{O}) = \frac{\mathbf{N}max - \mathbf{N}min}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.9}$$

The output image is then

$$\mathbf{N}_{x,y} = E(\mathbf{O}_{x,y}, \mathbf{O}) \tag{3.10}$$

```
function equalised = equalise(image)
%Nonlinear histogram equalisation
%
```

```
%  Usage: new image = equalise(image)
%
%  Parameters: image      - array of integers

%get dimensions
[rows,cols]=size(image);

%specify range of levels
range=255;

%and the number of points
number=cols*rows;

%initialise the image histogram
for i=1:256
  hist(i)=0;
end;

%work out the histogram
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    hist(image(y,x)+1)=hist(image(y,x)+1)+1;
  end
end;

%evaluate the cumulative histogram
sum=0;
for i=1:256
  sum=sum+hist(i);
  cumhist(i)=floor(sum*range/number);
end

%map using the cumulative histogram
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    equalised(y,x)=cumhist(image(y,x));
  end
end
```

Code 3.2 Histogram Equalisation

The result of equalising the eye image is shown in Figure 3.5. The intensity equalised image, Figure 3.5(c) has much better defined features (especially around the eyes) than in the original version (Figure 3.1). The histogram, Figure 3.5(d), reveals the non-linear mapping process whereby white and black are not assigned equal weight, as they were in intensity normalisation. Accordingly, more pixels are mapped into the darker region and the brighter intensities become better spread, consistent with the aims of histogram equalisation.

Its performance can be very convincing since it is well mapped to the properties of human vision. If a linear brightness transformation is applied to the original image then the equalised histogram will be the same. If we replace pixel values with ones computed according to Equation 3.1, the result of histogram equalisation will not change. An alternative interpretation is that if we equalise images (prior to further processing) then we need not worry about any brightness transformation in the original image. This is to be expected, since the linear operation of the brightness change in Equation 3.2 does not change the overall shape of the histogram, only its size and position. However, noise in the image acquisition process will affect the shape of the original histogram, and hence the equalised version. So the equalised histogram of a picture will not be the same as the equalised histogram of a picture with some noise added to it. You cannot avoid noise in electrical systems, however well you design a system to reduce its effect. Accordingly, histogram equalisation finds little use in generic image processing systems, though it can be potent in **specialised** applications. For these reasons, intensity normalisation is often preferred when a picture's histogram requires manipulation.

In implementation, the function `equalise` in Code 3.2, we shall use an output range where **N***min* = 0 and **N***max* = 255. The implementation first determines the cumulative histogram for each level of the brightness histogram. This is then used as a look up table for the new output brightness at that level. The look up table is used to speed implementation of Equation 3.9, since it can be precomputed from the image to be equalised.

An alternative argument against use of histogram equalisation is that it is a non-linear process and is irreversible. We cannot return to the original picture after equalisation, and we cannot separate the histogram of an unwanted picture. On the other hand, intensity normalisation is a linear process and we can return to the original image, should we need to, or separate pictures, if required. Note that there have been extensions to histogram equalisation, and an adaptive histogram equalization with some extensions [Pizer87] has proved particularly enduring.

3.3.4 Thresholding

The last point operator of major interest is called *thresholding*. This operator selects pixels which have a particular value, or are within a specified range. It can be used to find objects within a picture if their brightness level (or range) is known. This implies that the object's brightness must be known as well. There are two main forms: uniform and adaptive thresholding. In *uniform thresholding*, pixels above a specified level are set to white, those below the specified level are set to black.

$$\mathbf{N}_{x,y} = \left| \begin{array}{lll} 255 & if & \mathbf{N}_{x,y} > threshold \\ 0 & & otherwise \end{array} \right. \quad (3.11)$$

Given the original eye image, Figure 3.6 shows a thresholded image where all pixels **above** 160 brightness levels are set to white, and those below 160 brightness levels are set to black. By this process, the parts pertaining to the facial skin are separated from the background; the cheeks, forehead and other bright areas are separated from the hair and eyes. This can therefore provide a way of isolating points of interest.

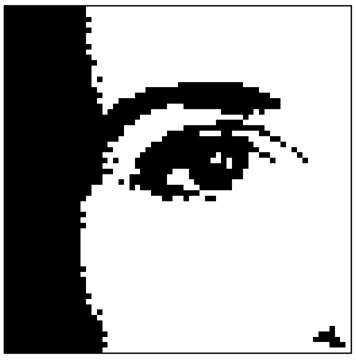


Figure 3.6 Thresholding the Eye Image

Uniform thresholding clearly requires knowledge of the grey level, or the target features might not be selected in the thresholding process. If the level is not known, histogram equalisation or intensity normalisation can be used, but with the restrictions on performance stated earlier. This is, of course, a problem of image interpretation. These problems can only be solved by simple approaches, such as thresholding, for very special cases. In general, it is often prudent to investigate the more sophisticated techniques of feature selection and extraction, to be covered later. Prior to that, we shall investigate group operators, which are a natural counterpart to point operators.

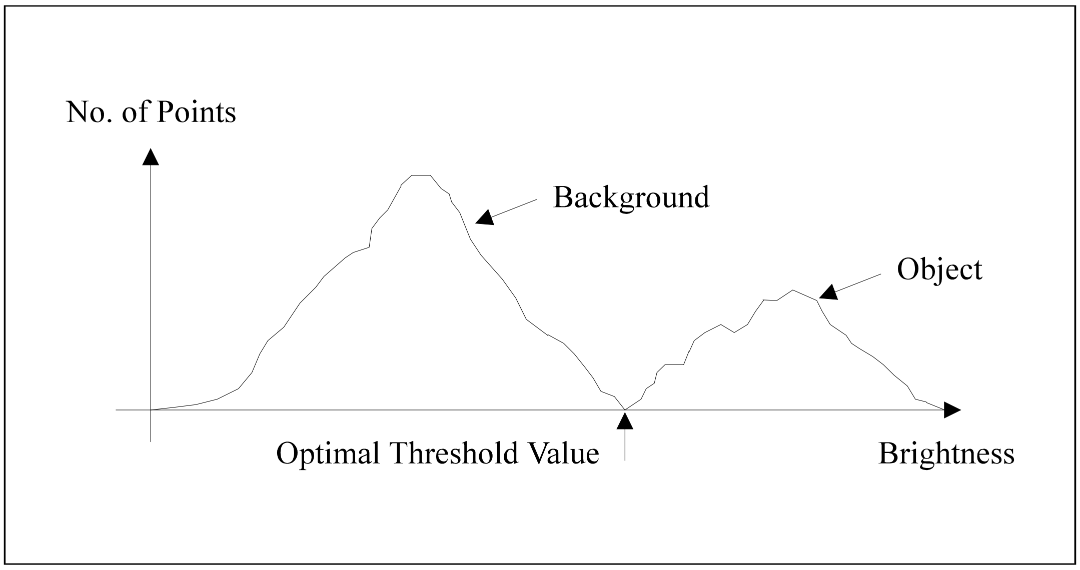


Figure 3.7 Optimal Thresholding

There are more advanced techniques, known as *optimal thresholding*. These usually seek to select a value for the threshold that separates an object from its background. This suggests that the object has a different range of intensities to the background, in order that an appropriate threshold can be chosen, as illustrated in Figure 3.7. Otsu’s method [Otsu79] is one of the most popular techniques of optimal thresholding; there have been surveys [Sahoo88, Lee90, Glasbey93] which compare the performance different methods can achieve. Essentially, Otsu's technique maximises the likelihood that the threshold is chosen so as to split the image between an object and its background. This is achieved by selecting a threshold that gives the best separation of classes, for all pixels in an image. The theory is beyond the scope of this section and we shall merely survey its results and give their implementation. The basis is use of the normalised histogram where the number of points at each level is divided by the total number of points in the image. As such, this represents a probability distribution for the intensity levels as

$$p(l) = \frac{\mathbf{N}(l)}{N^2} \tag{3.12}$$

This can be used to compute then zero- and first-order cumulative moments of the normalised histogram up to the k^{th} level as

$$\omega(k) = \sum_{l=1}^{k} p(l) \tag{3.13}$$

and

$$\mu(k) = \sum_{l=1}^{k} l \cdot p(l) \tag{3.14}$$

The total mean level of the image is given by

$$\mu\mathrm{T} = \sum_{l=1}^{N\max} l \cdot p(l) \tag{3.15}$$

The variance of the class separability is then the ratio

$$\sigma_B^2(k) = \frac{(\mu T \cdot \omega(k) - \mu(k))^2}{\omega(k)(1-\omega(k))} \qquad \forall k \in 1, N\text{max} \tag{3.16}$$

The optimal threshold is the level for which the variance of class separability is at its maximum, namely the optimal threshold T_{opt} is that for which the variance

$$\sigma_B^2(T_{\text{opt}}) = \max_{1 \le k < N\text{max}} \left(\sigma_B^2(k)\right) \tag{3.17}$$

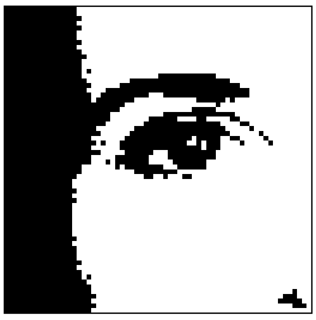
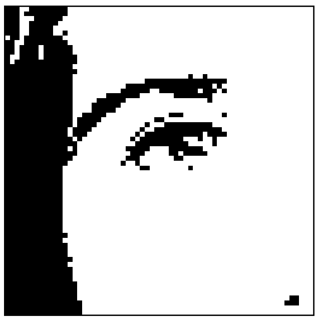
	
(a) thresholding at level 160	**(b)** thresholding by Otsu (level = 127)

Figure 3.8 Thresholding the Eye Image: Manual and Automatic

A comparison of uniform thresholding with optimal thresholding is given in Figure 3.8 for the eye image. The threshold selected by Otsu's operator is actually slightly lower than the value selected manually, and so the thresholded image does omit some detail around the eye, especially in the eyelids. However, the selection by Otsu is **automatic**, as opposed to **manual** and this can be to application advantage in automated vision. Consider for example the need to isolate the human figure in Figure 3.9(a). This can be performed automatically by Otsu as shown in Figure 3.9(b). Note however that there are some extra points, due to illumination, which have appeared in the resulting image together with the human subject. It is easy to remove the isolated points, as we will see later, but more difficult to remove the connected ones. In this instance, the size of the human shape could be used as information to remove the extra points though you might like to suggest other factors that could lead to their removal.

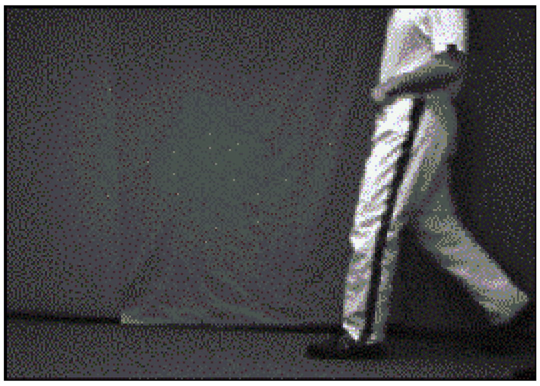
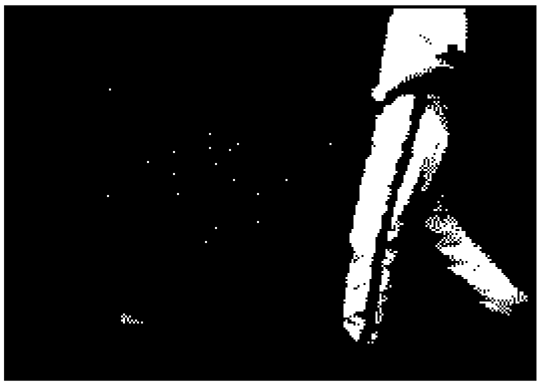
	
(a) walking subject	**(b)** automatic thresholding by Otsu

Figure 3.9 Thresholding an Image of a Walking Subject

Also, we have so far considered **global** techniques, methods that operate on the entire image. There are also **locally adaptive** techniques that are often used to binarise document images prior to character recognition. As mentioned before, surveys of thresholding are available, and one approach [Rosin01] targets thresholding of images whose histogram is unimodal (has a single peak). One survey [Trier95] compares global and local techniques with reference to document image analysis. These techniques are often used in statistical pattern recognition: the thresholded object is classified according to its statistical properties. However, these techniques find less use in image interpretation, where a common paradigm is that there is more than one object in the scene, such as Figure 3.6 where the thresholding operator has selected many objects of potential interest. As such, only uniform thresholding is used in many vision applications, since objects are often occluded (hidden), and many objects have similar ranges of pixel intensity. Accordingly, more sophisticated metrics are required to separate them, by using the uniformly thresholded image, as discussed in later Chapters. Further, the operation to process the thresholded image, say to fill in the holes in the silhouette or to remove the noise on its boundary or outside, is *morphology* which is covered later in Section 3.6.

3.4 Group Operations

3.4.1 Template Convolution

Group operations calculate new pixel values from a pixel's neighbourhood by using a 'grouping' process. The group operation is usually expressed in terms of *template convolution* where the template is a set of weighting coefficients. The template is usually square, and its size is usually odd to ensure that the result positioned precisely on a pixel. The size is usually used to describe the template; a 3×3 template is three pixels wide by three pixels long. New pixel values are calculated by placing the template at the point of interest. Pixel values are multiplied by the corresponding weighting coefficient and added to an overall sum. The sum (usually) evaluates a new value for the centre pixel (where the template is centred) and this becomes the pixel in the output image. If the template's position has not yet reached the end of a line, the template is then moved horizontally by one pixel and the process repeats.

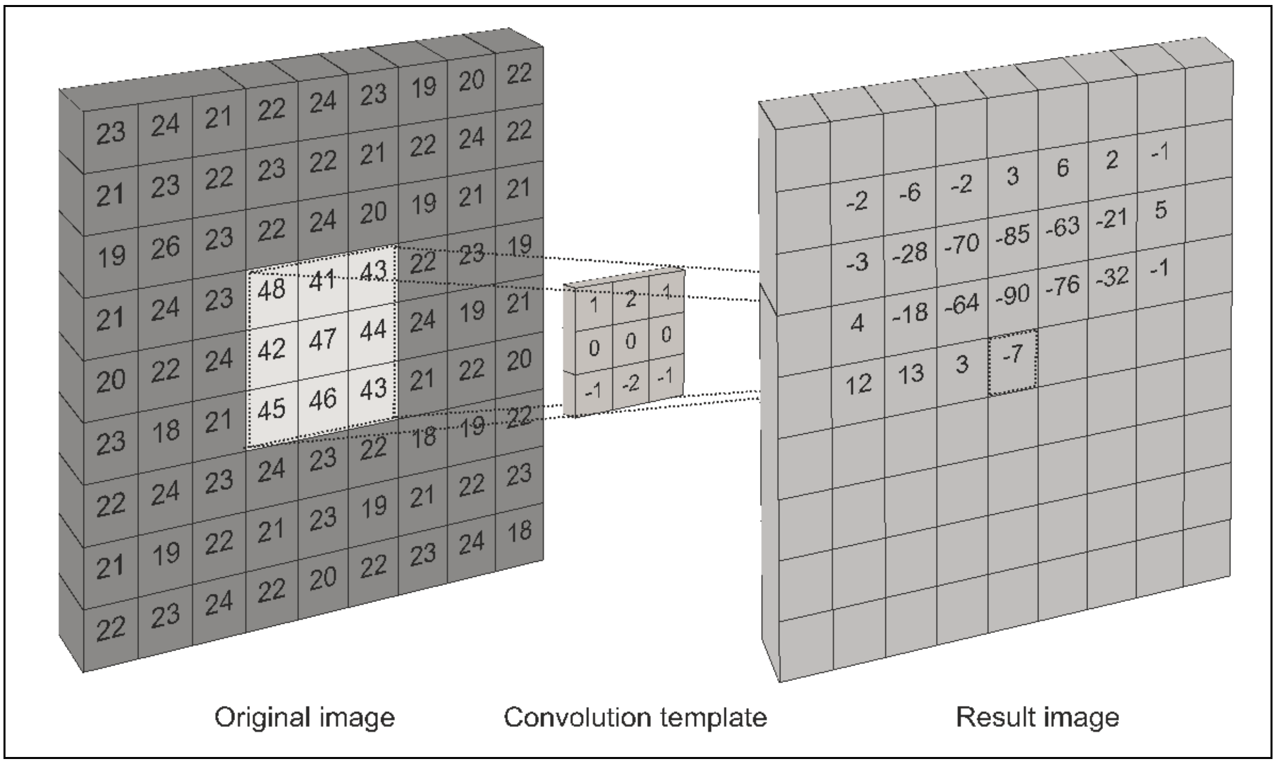


Figure 3.10 Template Convolution Process

This is illustrated in Figure 3.10 where a new image is calculated from an original one, by template convolution. The calculation obtained by template convolution at the centre pixel of the template in the original image becomes the point in the output result image. Since the template cannot extend beyond the original image, a new value cannot be computed for points at the border of the result image. When the template reaches the end of a line, it is repositioned to proceed from the start of the next line. The process is shown part way through the raster scan, the next pixel to be calculated would be derived from the nine points to the right of the current position of the centre point of the template, and stored to the right of the point containing -7. For a 3×3 neighbourhood, Fig. 3.11, nine weighting coefficients w_t are applied to points in the original image to calculate a point in the new image. The position of the new point (at the centre) is shaded in the template.

w_0	w_1	w_2
w_3	w_4	w_5
w_6	w_7	w_8

Figure 3.11 3×3 Template and Weighting Coefficients

To calculate the value in new image, **N**, at point with co-ordinates *x,y*, the template in Figure 3.11 operates on an original image **O** according to:

$$\mathbf{N}_{x,y} = \sum_{i \in \text{template}} \sum_{j \in \text{template}} w_{i,j} \times \mathbf{O}_{x(i),y(j)} \tag{3.18}$$

where the coordinates of the image point $x(i), y(j)$ denote the position of the point that matches the weighting coefficient position. Note that we cannot ascribe values to the picture's borders. This is because when we place the template at the border, parts of the template fall outside the image and have no information from which to calculate the new pixel value. The width of the border equals half the size of the template. In Figure 3.10 the single pixel border points have been left blank. To calculate values for the border pixels, we now have three choices:

1. set the border to black (or deliver a smaller picture);
2. assume (as in Fourier) that the image replicates to infinity along both dimensions and calculate new values by cyclic shift from the far border; or
3. calculate the border pixel value from a smaller area.

None of these approaches is optimal. The results in this book use the first option and set border pixels to black. Note that in many applications the object of interest is imaged centrally or, at least, imaged within the picture. As such, the border information is of little consequence to the remainder of the process. Here, the border points are set to black, by starting functions with a zero function which sets all the points in the picture initially to black (0).

An alternative representation for this process is given by using the convolution notation as

$$\mathbf{N} = \mathbf{W} * \mathbf{O} \tag{3.19}$$

where **N** is the new image which results from convolving the template **W** (of weighting coefficients) with the image **O**.

The Matlab implementation of a template convolution operator `template_convolve` is given in Code 3.3. This function accepts, as arguments, the picture `image` and the template to be convolved with it, `template`. The result of template convolution is an image `convolved`. The operator first initialises the size of the border and the resulting image to black (zero brightness levels). The border gives the range of picture points to be processed in the outer for loops that give the co-ordinates of all points resulting from template convolution. The template is convolved at each picture point by generating a running summation of the pixel values within the template's window multiplied by the respective template weighting coefficient.

```
function convolved = template_convolve(image,template)
%New image point brightness convolution of template with image
%  Usage: [new image] = convolve(image,template of point values)
%
%  Parameters: image      - array of points
%              template   - array of weighting coefficients

%get image dimensions
[rows,cols]=size(image);

%get template dimensions
[trows,tcols]=size(template);

%half of template rows is
tr=floor(trows/2);

%half of template cols is
tc=floor(tcols/2);

%set an output as black
convolved(1:rows,1:cols)=0;

%then convolve the template
for x = tc+1:cols-tc %address all columns except border
    for y = tr+1:rows-tr %address all rows except border
        sum=0; %initialise the sum
        for iwin=1:tcols %address all points in the template
            for jwin=1:trows
                sum=sum+image(y+jwin-tr-1,x+iwin-tc-1)*...
                           template(trows-jwin+1,tcols-iwin+1);
            end
        end
        convolved(y,x)=floor(sum);
    end
end
```

Code 3.3 Template Convolution Operator

Note that according to Eqn. 2.10, for convolution one of the signals is inverted along its principal axis. For images, convolution requires inversion along both axes which is why the template's arguments are inverted in Code 3.3. We shall consider convolution again in the next Section, via the frequency domain, and in Section 5.3.2.

Template convolution can of course be implemented in hardware and requires a two-line store, together with some further latches, for the (input) video data. The output is the result of template convolution, summing the result of multiplying weighting coefficients by pixel values. This is called pipelining, since the pixels are essentially move along a pipeline of information. Note that two line-stores can be used if the video fields only are processed. To process a full frame, one of the fields must be stored if it is presented in interlaced format. Processing can be analog, using operational amplifier circuits and Charge Coupled Device (CCD) for storage along bucket brigade delay lines. Finally, an alternative implementation is to use a parallel architecture: for Multiple Instruction Multiple Data (MIMD) architectures, the picture can be split into blocks (spatial partitioning); Single Instruction Multiple Data (SIMD) architectures can implement template convolution as a combination of shift and add instructions.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 3.12 3×3 Averaging Operator Template Coefficients

3.4.2 Averaging Operator

For an *averaging operator*, the template weighting functions are unity (or 1/9 to ensure that the result of averaging nine white pixels is white, not more than white!). The template for a 3×3 averaging operator, implementing Equation 3.18, is given by the template in Figure 3.12 where the location of the point of interest is again shaded. The averaging operator is then

$$\mathbf{N}_{x,y} = \frac{1}{MN}\sum_{i\in M}\sum_{j\in N}\mathbf{O}_{x(i),y(j)} \tag{3.20}$$

where $x(i), y(j)$ are the coordinates of image points within the template and *M*, *N* are the numbers of columns and rows in the template. The result of averaging the eye image with a 3×3 operator is shown in Figure 3.13. This shows that much of the detail has now disappeared revealing the broad image structure. The eyes and eyebrows are now much clearer from the background, but the fine detail in their structure has been removed.

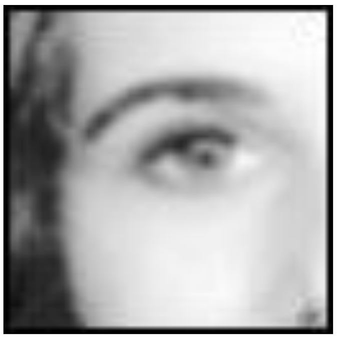


Figure 3.13 Applying Direct Averaging

In order to implement averaging by using the template convolution operator, we need to define a template and then convolve it with the image. (Note also that there is an averaging operator `mean` in Matlab that can be used for this purpose.)

The effect of averaging is to reduce noise, this is its advantage. An associated disadvantage is that averaging causes blurring which reduces detail in an image. It is also a **low pass** filter since its effect is to allow low spatial frequencies to be retained, and to suppress high frequency components. A larger template, say 3×3 or 5×5, will remove more noise (high frequencies) but reduce the level of detail. The size of an averaging operator is then equivalent to the reciprocal of the bandwidth of a low-pass filter it implements

3.4.3 On Different Template Size

Templates can be larger than 3×3. Since they are usually centred on a point of interest, to produce a new output value at that point, they are usually of odd dimension. For reasons of speed, the most common sizes are 3×3, 5×5 and 7×7. Beyond this, say 9×9, many template points are used to calculate a single value for a new point, and this imposes high computational cost, especially for large images. (For example, a 9×9 operator covers 9 times more points than a 3×3 operator.) Square templates have the same properties along both image axes. Some implementations use vector templates (a line), either because their properties are desirable in a particular application, or for reasons of speed.

The effect of larger averaging operators is to smooth the image more, to remove more detail whilst giving greater emphasis to the large structures. This is illustrated in Figure 3.14. A 5×5 operator, Figure 3.14(a), retains more detail than a 7×7 operator, Figure 3.14(b), and much more than a 9×9 operator, Figure 3.14(c). Conversely, the 9×9 operator retains only the largest structures such as the eye region (and virtually removing the iris) whereas this is retained more by the operators of smaller size. Note that the larger operators leave a larger border (since new values cannot be computed in that region) and this can be seen in the increase in border size for the larger operators, in Figures 3.14(b) and (c).

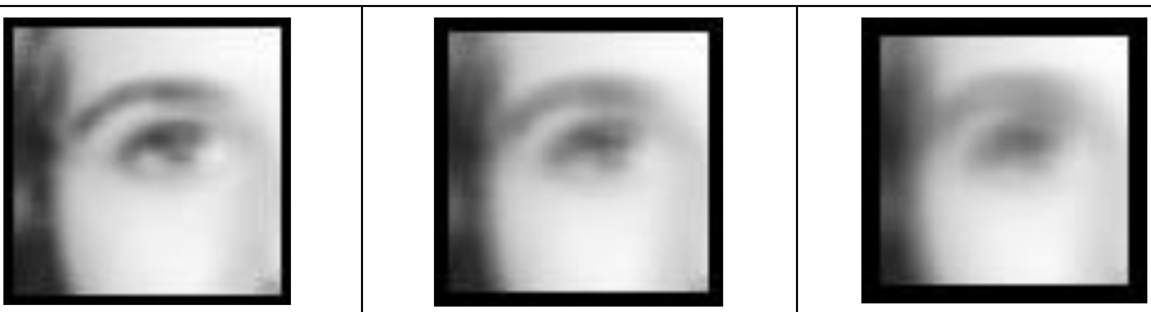

(a) 5×5 **(b)** 7×7 **(c)** 9×9

Figure 3.14 Illustrating the Effect of Window Size

3.4.4 Template Convolution via the Fourier Transform

The Fourier transform actually gives an alternative method to implement template convolution and to speed it up, for larger templates. The question to be answered here is ‘how big?’. In Fourier transforms, the process that is dual to **convolution** is **multiplication** (as in Section 2.3). So template convolution (denoted $*$) can be implemented by multiplying the Fourier transform of the template $\Im(\mathbf{T})$ with the Fourier transform of the picture, $\Im(\mathbf{P})$, to which the template is to be applied. It is perhaps a bit confusing that we appear to be multiplying matrices, but the multiplication is point-by-point in that the result at each point is that of multiplying the (single) points at the same positions in the two matrices. The result needs to be inverse transformed to return to the picture domain.

$$\mathbf{P} * \mathbf{T} = \Im^{-1}\left(\Im(\mathbf{P}) .\times \Im(\mathbf{T})\right) \tag{3.21}$$

The transform of the template and the picture need to be the same size before we can perform the point by point multiplication ($.\times$). Accordingly, the image containing the template is *zero-*

padded prior to its transform which simply means that zeroes are added to the template which lead to a template of the same size as the image. The process is illustrated in Code 3.4 and starts by calculation of the transforms of the image and of the zero-padded template. Then, the transform of the template is multiplied by the transform of the picture point-by-point (using the `.*` operator). (Theoretical study of this process is presented in Section 5.3.2 where we show how the same process can be used to find shapes in images.) Finally, the inverse Fourier transform is used to deliver the result.

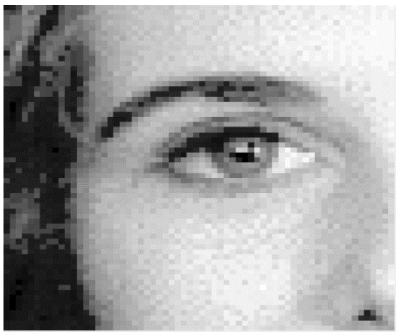
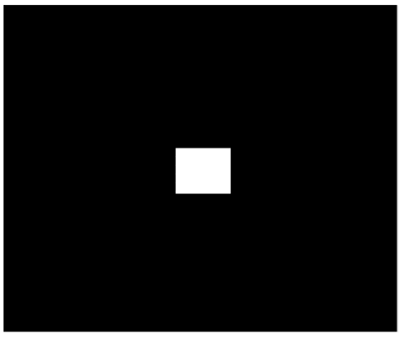
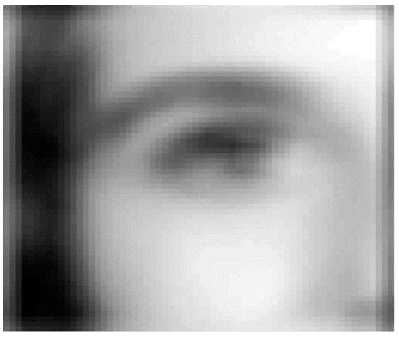
```
image_eye=imread('eye_orig.jpg');
image_eye=double(image_eye(:,:,1));
image_transform=fft2(image_eye);
template_transform=fft2(pad(image_eye, ave_template(7)));
inverted_transform=ifft2(rearrange(image_transform.*template_transform));
```

The transform based implementation of direct averaging can be combined as

```
averaged_image=ifft2(rearrange(fft2(eye).*fft2(pad(eye,ave_template(7)))));
```

Code 3.4 Template Convolution via the Fourier Transform

Code 3.4 is simply a different implementation of direct averaging. It achieves a similar result, but by transform domain calculus. The operation is shown in Fig. 3.15: an image of the eye (a) is transformed to give (d); the averaging template is padded to the same size as the image (b) and transformed (e); the multiplied transforms (f) are inverse transformed to give an averaged version of the eye (c). There is one major difference between the Fourier and the direct implementations: the borders of the images differ (where the border is of width equal to one half of the template's width). This is because for direct averaging the border points are set to zero whereas in the Fourier implementation the image is assumed to replicate to infinity, as in Fig. 2.29 and Eq. 2.26. (The rearrange function, Eq. 2.28, is used since the padding function places the template at the centre of the image). Note that the template transform is a 2D sinc function viewed as an image and that the logarithm of the magnitude (Section 3.3.1) has been used to display all transforms.

		
(a) image of eye	(b) padded averaging template	(c) resulting averaged image

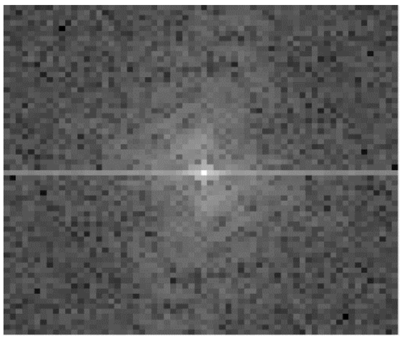
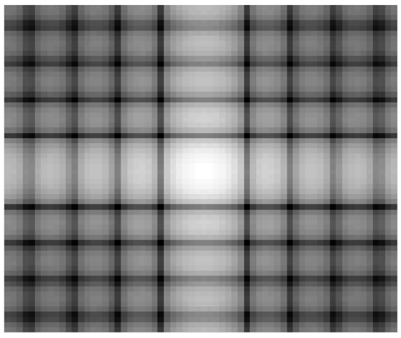
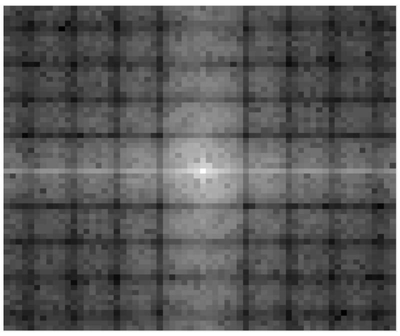
		
(d) image transform	(e) template transform	(f) multiplied transforms

Figure 3.15 Template Convolution via Fourier Transform

It can be **faster** to use the transform-based implementation (Code 3.4) rather than the direct implementation (Code 3.3), depending on the size of the template. For a square template with $N \times N$ points the computational cost of a 2D FFT is of the order of $2N^2\log(N)$. If the transform of the template is pre-computed, there are two transforms required and there is one multiplication for each of the N^2 transformed points. The total cost of the Fourier implementation of template convolution is then of the order of

$$C_{FFT} = 4N^2\log(N) + N^2 \tag{3.22}$$

The cost of the direct implementation for a $m \times m$ template is then m^2 multiplications for each image point, so the cost of the direct implementation is of the order of

$$C_{dir} = N^2 m^2 \tag{3.23}$$

For $C_{dir} < C_{FFT}$, we require:

$$N^2 m^2 < 4N^2\log(N) + N^2 \tag{3.24}$$

If the direct implementation of template matching is faster than its Fourier implementation, we need to choose m so that

$$m^2 < 4\log(N) + 1 \tag{3.25}$$

This implies that, for a 256×256 image, a direct implementation is fastest for 3×3 and 5×5 templates, whereas a transform-based calculation is faster for larger ones. An alternative analysis [Campbell69] has suggested that [Gonzalez17] 'if the number of non-zero terms in (the template) is less than 132 then a direct implementation .... is more efficient than using the FFT approach'. In OpenCV, for some versions the limit appears to 7×7 [OpenCV-TM] for using the FFT and for a kernel size of 5×5 or less a direct version is used. An 11×11 operator is a considerably larger template than our analysis suggests, whereas OpenCV has the same limit as the analysis here. This might be due to higher considerations of complexity than our analysis has included. There are, naturally, further considerations in the use of transform calculus, the most important being the use of windowing (such as Hamming or Hanning) operators to reduce variance in high order spectral estimates. This implies that template convolution by transform calculus should be used when large templates are involved, and when speed is critical. If speed is indeed critical, it might be prudent to implement the operator in dedicated hardware, as described earlier.

3.4.5 Gaussian Averaging Operator

The *Gaussian averaging operator* has been considered to be optimal for image smoothing. The template for the Gaussian operator has values set by the Gaussian relationship. The Gaussian **function** *g* at coordinates *x*,*y* is controlled by the *variance* σ^2 according to:

$$g(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{\frac{-\left(x^2+y^2\right)}{2\sigma^2}} \tag{3.26}$$

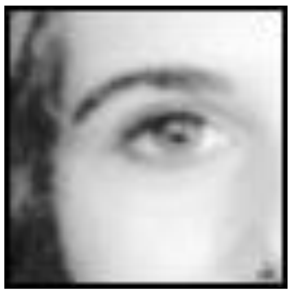
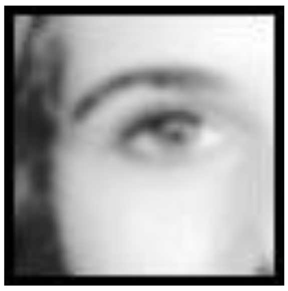
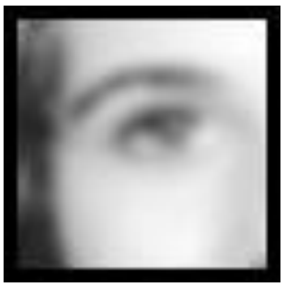
		
(a) 3×3	**(b)** 5×5	**(c)** 7×7

Figure 3.16 Applying Gaussian Averaging

Equation 3.26 gives a way to calculate coefficients for a Gaussian template which is then convolved with an image. The effects of selection of Gaussian templates of differing size are shown in Figure 3.16. The Gaussian function essentially removes the influence of points greater than 3σ in (radial) distance from the centre of the template. The 3×3 operator, Figure 3.16(a), retains many more of the features than those retained by direct averaging (Figure 3.13). The effect of larger size is to remove more detail (and noise) at the expense of losing features. This is reflected in the loss of internal eye component by the 5×5 and the 7×7 operators in Figures 3.16(b) and (c), respectively.

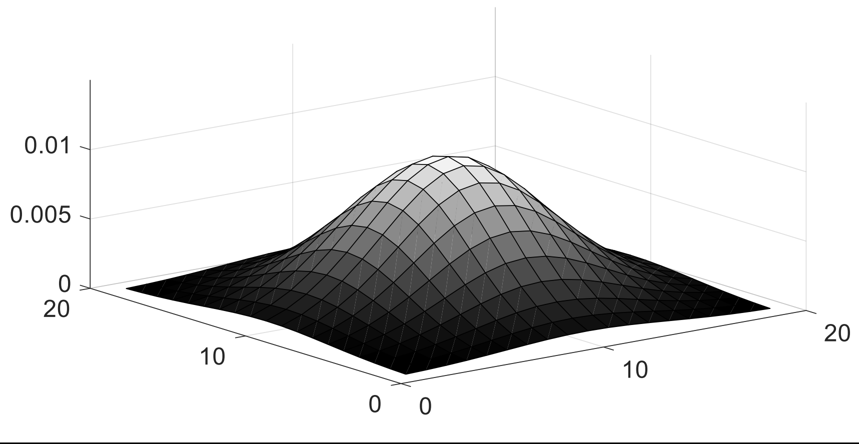


Figure 3.17 Gaussian Function

A surface plot of the 2D Gaussian function of Equation 3.26 has the famous bell-shape, as shown in Figure 3.17 (for a window size of 19×19 and standard deviation of 4.0). The values of the function at discrete points are the values of a Gaussian template. Convolving this template with an image gives Gaussian averaging: the point in the averaged picture is calculated from the sum of a region where the central parts of the picture are weighted to contribute more than the peripheral points.

The size of the template essentially dictates appropriate choice of the variance. The variance is chosen to ensure that template coefficients drop to near zero at the template's edge. The template for size 5×5 with variance unity is shown in Figure 3.18.

0.002	0.013	0.022	0.013	0. 002
0.013	0.060	0. 098	0. 060	0.013
0.022	0. 098	0.162	0. 098	0.022
0.013	0. 060	0.098	0. 060	0.013
0. 002	0.013	0.022	0.013	0. 002

Figure 3.18 Template for the 5×5 Gaussian Averaging Operator ($\sigma = 1.0$).

This template is then convolved with an image to give the Gaussian blurring function. It is actually possible to give the Gaussian blurring function antisymmetric properties by scaling the *x* and *y* co-ordinates. This can find application when an object's shape, and orientation, is known prior to image analysis.

By reference to Figure 3.16 it is clear that the Gaussian filter can offer improved performance compared with direct averaging: more features are retained whilst the noise is removed. This can be understood by Fourier transform theory. In Section 2.5.2 we found that the Fourier transform of a **square** is a two-dimensional **sinc** function. This has a frequency response where the magnitude of the transform does not reduce in a smooth manner and has regions where it becomes negative, called sidelobes. These can have undesirable effects since there are high frequencies that contribute **more** than some lower ones, a bit paradoxical in low-pass filtering which aims to remove noise. In contrast, the Fourier transform of a Gaussian function is another Gaussian function, which decreases smoothly without these sidelobes. This can lead to better performance since the contributions of the frequency components reduce in a controlled manner.

```
function template = gaussian_template(winsize,sigma)
%Template for Gaussian averaging
%
%  Usage: [template] = gaussian_template(number, number)
%
%  Parameters: winsize    - size of template (odd, integer)
%              sigma      - variance of Gaussian function

%centre is half of window size
centre=floor(winsize/2)+1;

%we'll normalise by the total sum
sum=0;

for i=1:winsize
  for j=1:winsize
    template(j,i)=exp(-(((j-centre)*(j-centre))+
                        ((i-centre)*(i-centre)))/(2*sigma*sigma));
    sum=sum+template(j,i);
  end
end

template=template/sum;
```

Code 3.5 Gaussian Template Specification

In a software implementation of the Gaussian operator, we need a function implementing Equation 3.26, the `Gaussian_template` function in Code 3.5. This is used to calculate the coefficients of a template to be centred on an image point. The two arguments are `winsize`, the (square) operator's size, and the standard deviation σ that controls its width, as discussed earlier. The operator coefficients are normalised by the sum of template values, as before. This summation is stored in `sum`, which is initialised to zero. The centre of the square template is then evaluated as half the size of the operator. Then, all template coefficients are calculated by a version of Equation 3.26 which specifies a weight relative to the centre co-ordinates. Finally, the normalised template coefficients are returned as the Gaussian template. The operator is used in template convolution, via `convolve`, as in direct averaging (Code 3.3).

3.4.6 More on Averaging

There is more than could be discussed on basic smoothing, e.g. smoothing was earlier achieved by low-pass filtering via the Fourier transform (Section 2.8), but we shall move on to other operators. The averaging process is actually a statistical operator since it aims to estimate the mean of a local neighbourhood. The error in the process is naturally high, for a population of N samples, the statistical error is of the order of:

$$error = \frac{mean}{\sqrt{N}} \tag{3.27}$$

Increasing the averaging operator's size improves the error in the estimate of the mean, but at the expense of fine detail in the image. The average is of course an estimate optimal for a signal corrupted by additive *Gaussian noise*. The estimate of the mean maximised the probability that the noise has its mean value, namely zero. According to the *central limit theorem*, the result of adding many noise sources together is a Gaussian distributed noise source. In images, noise arises in sampling, in quantisation, in transmission and in processing. By the central limit theorem, the result of these (independent) noise sources is that image noise can be assumed to be Gaussian. In fact, image noise is **not** necessarily Gaussian-distributed, giving rise to more statistical operators. One of these is the *median* operator which has demonstrated capability to reduce noise whilst retaining feature boundaries (in contrast to smoothing which blurs both noise and the boundaries), and the *mode* operator which can be viewed as optimal for a number of noise sources, including *Rayleigh noise*, but is very difficult to determine for small, discrete, populations.

2	8	7
4	0	6
3	5	7

(a) 3×3 template

2	4	3	8	0	5	7	6	7

(b) unsorted vector

0	2	3	4	5	6	7	7	8

↑ median

(c) sorted vector, giving median

Figure 3.19 Finding the Median from a 3×3 Template.

3.5 Other Image Processing Operators

3.5.1 Median Filter

The *median* is another frequently used statistic; the median is the centre of a rank-ordered distribution. The median is usually taken from a template centred on the point of interest. Given the arrangement of pixels in Figure 3.19(a), the pixel values are arranged into a vector format, Figure 3.19(b). The vector is then sorted into ascending order, Figure 3.19(c). The median is the central component of the sorted vector, this is the fifth component since we have nine values.

The median can of course be taken from **larger** template sizes. The development here has aimed not only to demonstrate how the median operator works, but also to provide a basis for further development. The rank ordering process is computationally demanding (**slow**) and motivates study into the deployment of fast algorithms, such as Quicksort, (e.g. [Huang79] is an early approach), though other approaches abound [Weiss06]. The computational demand also has motivated use of template shapes, other than a square. A selection of alternative shapes is shown in Figure 3.20. Common alternative shapes include a cross or a line (horizontal or vertical), centred on the point of interest, which can afford much faster operation since they cover fewer pixels. The basis of the arrangement presented here could be used for these alternative shapes, if required.

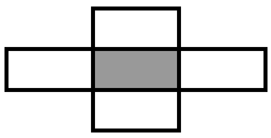
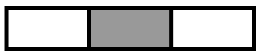
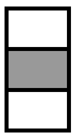
		
(a) cross	**(b)** horizontal line	**(c)** vertical line

Figure 3.20 Alternative Template Shapes for Median Operator

The median has a well-known ability to remove *salt and pepper noise*. This form of noise, arising from say decoding errors in picture transmission systems, can cause isolated white and black points to appear within an image. It can also arise when rotating an image, when points remain unspecified by a standard rotation operator (Chapter 10), as in a texture image, rotated by 10^o in Figure 3.21(a). When a median operator is applied, the salt and pepper noise points will appear at either end of the rank ordered list and are removed by the median process, as shown in Figure 3.21(b). The median operator has practical advantage, due to its ability to retain edges (the boundaries of shapes in images) whilst suppressing the noise contamination. As such, like direct averaging, it remains a worthwhile member of the stock of standard image processing tools. For further details concerning properties and implementation, have a peep at [Hodgson85]. (Note that practical implementation of image rotation is a Computer Graphics issue, and is usually by **texture mapping**; further details can be found in [Hearn97].)

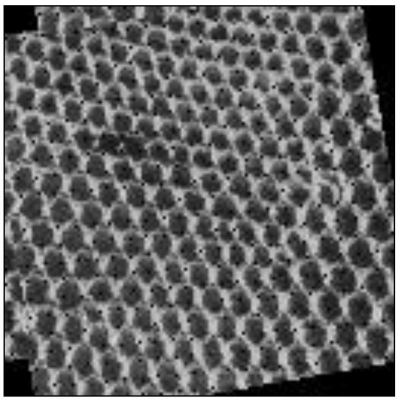
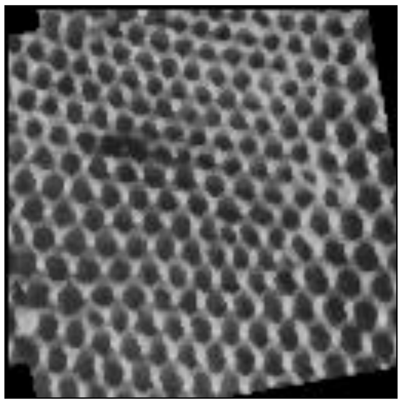
	
(a) rotated fence	**(b)** median filtered

Figure 3.21 Illustrating Median Filtering

3.5.2 Mode Filter

The *mode* is the final statistic of interest, though there are more advanced filtering operators to come. The mode is of course very difficult to determine for small populations and theoretically does not even exist for a continuous distribution. Consider for example determining the mode of the pixels within a square 5×5 template. Naturally, it is possible for all 25 pixels to be different, so each could be considered to be the mode. As such we are forced to estimate the mode: the truncated median filter, as introduced by Davies [Davies88], aims to achieve this. The *truncated median filter* is based on the premise that for many non-Gaussian distributions, the order of the mean, the median and the mode is the same for many images, as illustrated in Figure 3.22. Accordingly, if we truncate the distribution (i.e. remove part of it, where the part selected to be removed in Figure 3.22 is from the region beyond the mean) then the median of the truncated distribution will approach the mode of the original distribution.

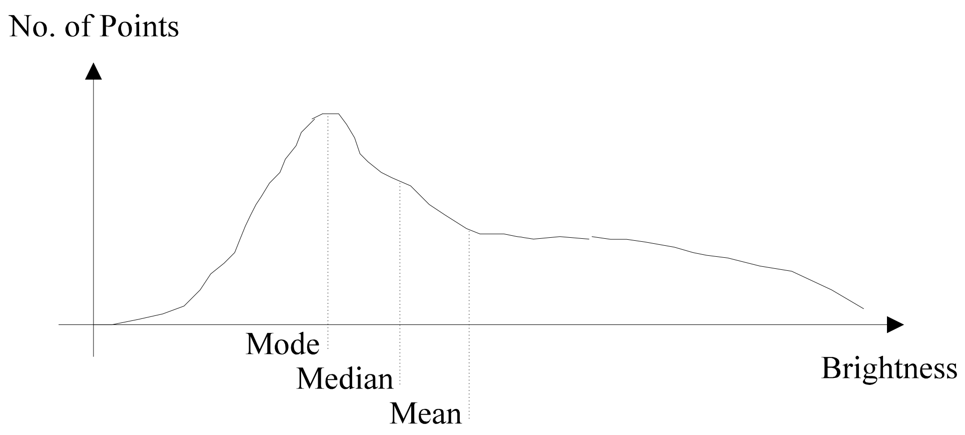


Figure 3.22 Arrangement of Mode, Median and Mean

In implementation the operator first finds the mean and the median of the current window. The distribution of intensity of points within the current window is truncated on the side of the mean so that the median now bisects the distribution of the remaining points (as such not affecting symmetrical distributions). So that the median bisects the remaining distribution, if the median is less than the mean, the point at which the distribution is truncated, *upper*, is

$$\begin{aligned} upper &= median + (median - \min(distribution)) \\ &= 2 \cdot median - \min(distribution) \end{aligned} \tag{3.28}$$

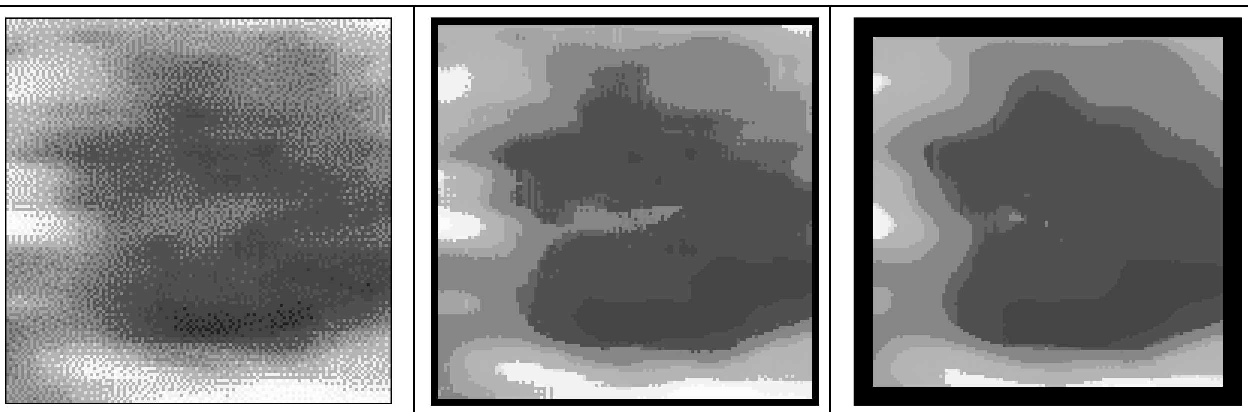
If the median is greater than the mean, then we need to truncate at a lower point (before the mean), *lower*, given by

$$lower = 2 \cdot median - \max(distribution) \tag{3.29}$$

The median of the remaining distribution then approaches the mode. The truncation is performed by storing pixels values in a vector. The median of the truncated vector is then the output of the truncated median filter at that point. Naturally, the window is placed at each possible image point, as in template convolution. However, there can be several iterations at each position to ensure that the mode is approached. In practice only few iterations are usually required for the median to converge to the mode. The window size is usually large, say 7×7 or 9×9 or even more.

The action of the operator is illustrated in Figure 3.23 when applied to a 128×128 part of the ultrasound image (Figure 1.1(c)), from the centre of the image and containing a cross-sectional view of an artery. Ultrasound results in particularly noisy images, in part because the scanner is usually external to the body. The noise is actually multiplicative Rayleigh noise for which the mode is the optimal estimate. This noise obscures the artery which appears in cross-section in Figure 3.23(a); the artery is basically elliptical in shape. The action of the 9×9 truncated median operator, Figure 3.23(b) is to remove noise whilst retaining feature boundaries whilst a larger operator shows better effect, Figure 3.23(c).

Close examination of the result of the truncated median filter is that a selection of boundaries are preserved which are not readily apparent in the original ultrasound image. This is one of the known properties of median filtering: an ability to reduce noise whilst retaining feature boundaries. Indeed, there have actually been many other approaches to speckle filtering; the most popular for ultrasound include direct averaging [Shankar86], median filtering, adaptive (weighted) median filtering [Loupas87] and with basis using nonlocal means [Coupé09] and anisotropic (coherent) diffusion [Abd-Elmoniem 2002] (the latter two to be covered next).



(a) part of ultrasound image | **(b)** 9×9 operator | **(c)** 13×13 operator

Figure 3.23 Applying Truncated Median Filtering

3.5.3 Non-Local Means

The non-local means operator [Buades05] is an extended version of the averaging operator. The basic function of the operator is to assign a point a value that is the mean of an area that is closest to the mean at the value of the point, rather than the mean at that point. As the original paper put it "the denoised value at *x* is a mean of the values of all points whose Gaussian neighbourhood looks like the neighbourhood of *x*". As the former this was rather hard to express and the latter is rather terse, it will be difficult to understand. So let's slow down a bit. By denoting the average (mean) at a point *p* as $\bar{x}(p)$, for an $N \times N$ region, from Eq. 3.20

$$\bar{x}(p) = \frac{1}{N^2} \sum_{i \in M} \sum_{j \in N} \mathbf{O}_{x(i),y(j)} \tag{3.30}$$

where $x(i), y(j)$ are the coordinates of image points within the template. If this operation is performed over the whole image, we have applied the direct averaging operator. The parts of the process are shown in Fig. 3.24.

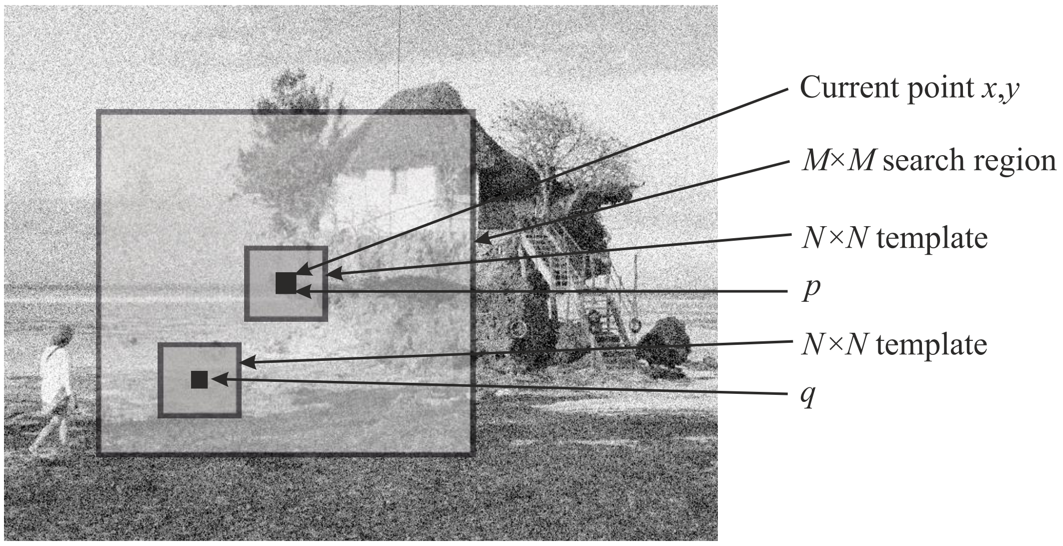


Figure 3.24 Non-Local Means Process

As averaging processes use a weighted sum if we use the Gaussian function, via Eq. 3.26, to calculate a weighting *w* between the average at point *p* and the average at point *q*

$$w(p,q) = g\left(\bar{x}(p) - \bar{x}(q)\right) = \frac{1}{2\pi\sigma^2} e^{\frac{-\left(\bar{x}(p)-\bar{x}(q)\right)^2}{2\sigma^2}} \tag{3.31}$$

Then the weight will be unity when $\bar{x}(p) = \bar{x}(q)$ and much less when the two averages are very different. The product $\bar{x}(p) \times w(p,q)$ will be equal to $\bar{x}(p)$ when the mean of the point of interest *p* is the same as the mean of the region at point *q*. We shall use an $M \times M$ search region. The non-local means operator is then a weighted summation

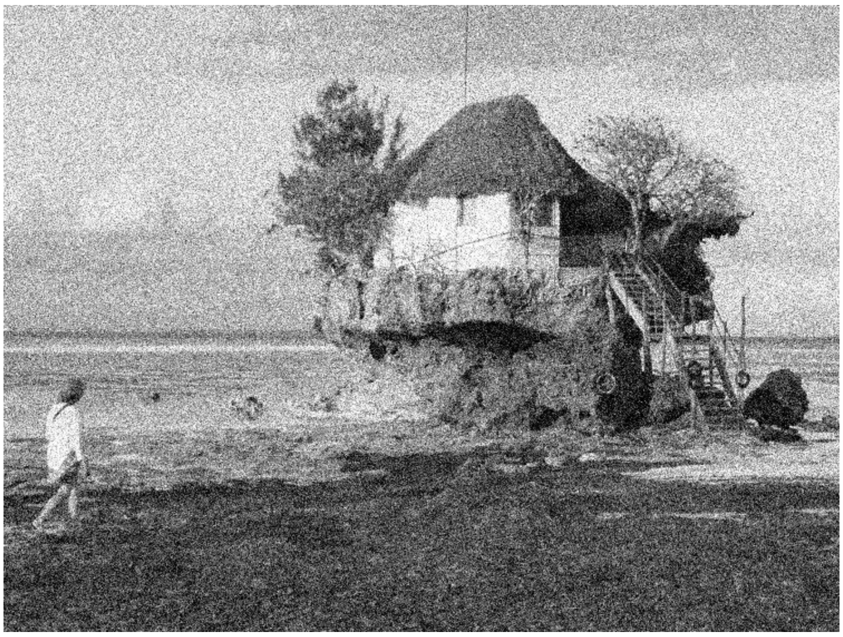
$$\mathbf{N}_{x,y} = \frac{1}{k}\sum_{i\in M}\sum_{j\in M}\mathbf{O}_{x(i),y(j)}\,g\left(\bar{x}(p)-\bar{x}(q)\right) \qquad (3.32)$$
$$= \frac{1}{\sum_{i\in M}\sum_{j\in M} g\left(\bar{x}(p)-\bar{x}(q)\right)}\sum_{i\in M}\sum_{j\in M}\mathbf{O}_{x(i),y(j)}\,g\left(\bar{x}(p)-\bar{x}(q)\right)$$

where k is the normalising function $k=\sum_{i\in M}\sum_{j\in N} g\left(\bar{x}(p)-\bar{x}(q)\right)$. The parameters that must be chosen are the window size of the averaging operator, N, the size of the search region, M, and the standard deviation. A Matlab implementation is given in Code 3.6 which follows the equations above. The border of the resulting image is set to half the window size of the larger of the averaging and range operators.

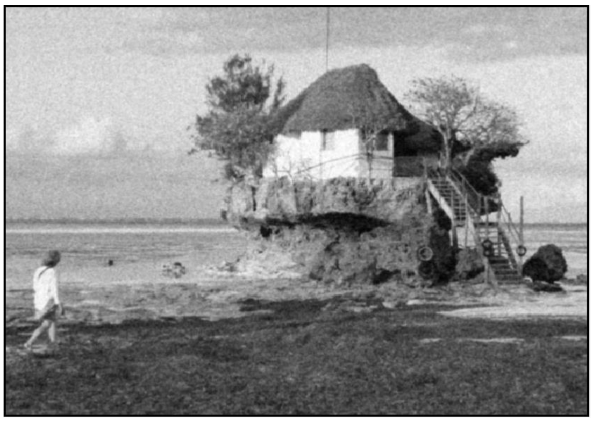
```
function filtered = non_local_means(image,winsize,searchsize,st_dev)

%get image dimensions
[rows,cols]=size(image);

%set the output image to black
filtered(1:rows,1:cols)=0;
local_mean(1:rows,1:cols)=0;

%half of template is
halfwin=floor(winsize/2);
%and half of the search
halfsearch=floor(searchsize/2);

%process points according to largest window function
if winsize>searchsize
    border=halfwin; %normal case
else
    border=halfsearch; %which is possible, but daft
end

%%then form the local averages
local_mean=floor(ave(image,winsize));

%we start by looking at all image points except those in the border
for x = border+1:cols-border
  for y = border+1:rows-border
      %need to total up the weights (for later normalisation)
      weightsum=0;
      %need to total up the weighted points
      productsum=0;
      for iwin = 1:searchsize
        for jwin = 1:searchsize
          %evaluate a Gaussian weight based on the intensity difference
          weightp=exp(-(local_mean(y+jwin-halfsearch-1,x+iwin-halfsearch-1)-...
                      local_mean(y,x))^2/(2*st_dev*st_dev));
          %and add a weighted amount of the image information
          productsum=productsum+weightp*...
                      image(y+jwin-halfsearch-1,x+iwin-halfsearch-1);
          %and add up the weights
          weightsum=weightsum+weightp;
        end
      end
      filtered(y,x)=floor(productsum/weightsum);
  end
end
```

Code 3.6 Non Local Means Operator

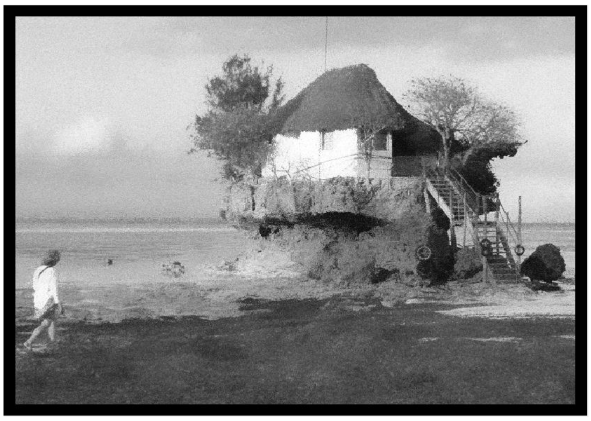
The effect of the non local means operator compared with Gaussian averaging is shown in Fig. 3.25. The image here Fig. 3.25(a) is one to which synthetic noise has been added, and that is clearly seen to be removed by both operators. The function of the non-local means operator is to preserve regions of intensity and that is what is achieved (compare the wall of the hut in Fig. 3.25(b) to that in Fig. 3.25(c)). These images are not intended to be optimal, since optimisation depends on the application image and whether preservation of detail is more important than preservation of regions. This is implicit in the compromise between the choice of the size of the search region and the variance σ^2. A common choice for the window size, *N*, of the averaging operator is 7×7 or 9×9 to preserve local features and larger search sizes, *M*, to reduce noise. The concern the selection on the range and its relationship with the noise has motivated approaches that adapt the range according to image content [Kervrann06]. The image here Fig. 3.25(a) is one to which synthetic noise has been added, and that is clearly seen to be removed by both operators. Note that the image would originally contain noise without the addition of its synthesised form, but the results would be less easy to see. Peformance analysis often depends on a chosen application.



(a) original image



(b) Gaussian averaging



(c) nonlocal means

Figure 3.25 Reducing Noise by Non Local Means and Gaussian Operators

Calling the operator non-local is actually rather misleading, as the originators later noted [Buades11] and a more appropriate name could have been semi-local means. Given that much of its popularity is due to performance, there is a natural interest in speeding the algorithm. One approach concentrates on implementation to improve speed [Dowson11] which, due to computational complexity, is needed when processing volumetric images.

3.5.4 Bilateral filtering

Bilateral filtering is a non-linear filter introduced by Tomasi and Manduchi [Tomasi98]. The filter is based on Gaussian averaging and prevents blurring across feature boundaries by decreasing the filter weight when the intensity difference is too large. By denoting Gaussian averaging as the convolution of the template in Eq 3.26 to form a new point $\mathbf{G}_{x,y}$ as a weighted sum of image points within a template

$$\mathbf{G}_{x,y} = \frac{1}{ks}\sum_{i\in N}\sum_{j\in N}\mathbf{O}_{x(i),y(j)}g\left(i,j,\sigma_d\right) \qquad (3.33)$$
$$= \frac{1}{\sum_{i\in N}\sum_{j\in N}g\left(i,j,\sigma_d\right)}\sum_{i\in N}\sum_{j\in N}\mathbf{O}_{x(i),y(j)}e^{\frac{-\left((x(i)-x)^2+(y(j)-y)^2\right)}{2\sigma_d^2}}$$

where σ_d is the standard deviation of this spatial (domain) operator, $x(i),y(j)$ are again the coordinates of image points within the template, *ks* is the normalising coefficient, and the template is of size $N\times N$. The bilateral filtering process introduces another weighting based on the difference of image intensities. Inevitably, this is formed as a Gaussian (range) function in the manner of Eq. 3.31

$$g\left(p,q,\sigma_r\right) = \frac{1}{2\pi\sigma_r^2}e^{\frac{-(p-q)^2}{2\sigma_r^2}} \qquad (3.34)$$

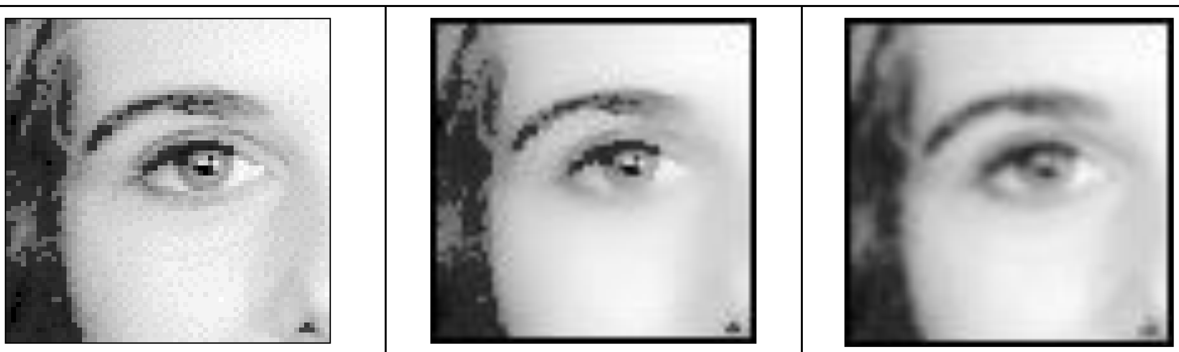
Where p,q are point intensities and σ_r is the standard deviation, controlling width. When the intensity difference is small then this function is large, and the operator preserves feature boundaries, preventing blurring. This is one of the advantages of the median operator and a disadvantage of the Gaussian smoothing operator. The bilateral filter is given by the combination of spatial averaging with the Gaussian weighted difference in brightness as

$$\mathbf{N}_{x,y} = \frac{1}{kb}\sum_{i\in M}\sum_{j\in M}\mathbf{G}_{x(i),y(j)}g\left(\mathbf{O}_{x,y},\mathbf{O}_{x(i),y(j)},\sigma_r\right) \qquad (3.35)$$
$$= \frac{1}{kb}\sum_{i\in M}\sum_{j\in M}\mathbf{G}_{x(i),y(j)}e^{\frac{-\left(\mathbf{O}_{x,y}-\mathbf{O}_{x(i),y(j)}\right)^2}{2\sigma_r^2}}$$

where *kb* is the sum of the template coefficients. The parameters that must be chosen are the window sizes and the two values for standard deviation. Note that the operator reverts to a Gaussian operator when σ_r is large, and a range filter when σ_d is large. The difference from non-local means is that both areas and feature boundaries are preserved, though we shall not show the result here, partly as performance is similar to that of anisotropic diffusion, the following smoothing operator. Optimised versions are available, which do not need parameter selection [Paris08, Weiss06]. Another method is based on the use of the frequency domain and uses linear filtering [Dabov07].

3.5.5 Anisotropic Diffusion

Another form of smoothing that preserves the boundaries of the image features in the smoothing process [Perona90]. The process is called *anisotropic diffusion*, by virtue of its basis. Its result is illustrated in Fig 3.26(b) where the feature boundaries (such as those of the eyebrows or the eyes) in the smoothed image are crisp and the skin is more matte in appearance. This implies that we are filtering within the features and not at their edges. By way of contrast, the Gaussian operator result in Fig. 3.26(c) smooths not just the skin but also the boundaries (the eyebrows in particular seem quite blurred) giving a less pleasing and less useful result. Since we shall later use the boundary information to interpret the image, its preservation is of much interest.



(a) original image	**(b)** anisotropic diffusion	**(c)** Gaussian smoothing

Figure 3.26 Filtering by Anisotropic Diffusion and the Gaussian Operator

As ever, there are some parameters to select to control the operation, so we shall consider the technique's basis so as to guide their selection. Further, it is computationally more complex than Gaussian filtering. The basis of anisotropic diffusion is however rather complex, especially here, and invokes concepts of low-level feature extraction which are covered in the next Chapter. One strategy you might use is to mark this page then go ahead and read Sections 4.1 and 4.2 and return here. Alternatively, you could just plough on, since that is exactly what we shall do. The complexity is because the process not only invokes low-level feature extraction (to preserve feature boundaries) but also as its basis actually invokes concepts of *heat flow*, as well as introducing the concept of *scale space*. So it will certainly be a hard read for many, but comparison of Figure 3.26(b) with Figure 3.26(c) shows that it is well worth the effort.

The essential idea of scale space is that there is a *multiscale* representation of images, from low resolution (a coarsely sampled image) to high resolution (a finely sampled image). This is inherent in the sampling process where the coarse image is the structure and the higher resolution increases the level of detail. As such, we can derive a *scale space* set of images by convolving an original image with a Gaussian function,

$$\mathbf{P}_{x,y}(\sigma)=\mathbf{P}_{x,y}(0)*g(x,y,\sigma) \tag{3.36}$$

where $\mathbf{P}_{x,y}(0)$ is the original image, $g(x,y,\sigma)$ is the Gaussian template derived from Eqn. 3. 26, and $\mathbf{P}_{x,y}(\sigma)$ is the image at level σ. The coarser level corresponds to larger values of the standard deviation σ; conversely the finer detail is given by smaller values. (Scale space will be considered again in Section 4.4.2 as it pervades the more modern operators). We have already seen that the larger values of σ reduce the detail and are then equivalent to an image at a coarser scale, so this is a different view of the same process. The difficult bit is that the family of images derived this way can equivalently be viewed as the solution of the heat equation

$$\partial \mathbf{P} / \partial t = \nabla \mathbf{P}_{x,y}(t) \tag{3.37}$$

where ∇ denotes del, the (directional) gradient operator from vector algebra, and with the initial condition that $\mathbf{P}_0 = \mathbf{P}_{x,y}(0)$. The heat equation itself describes the temperature *T* changing with time *t* as a function of the thermal diffusivity (related to conduction) κ as

$$\partial T / \partial t = \kappa \nabla^2 T \tag{3.38}$$

and in one dimensional form this is

$$\partial T / \partial t = \kappa \frac{\partial^2 T}{\partial x^2} \tag{3.39}$$

so the temperature measured along a line is a function of time, distance, the initial and boundary conditions and the properties of a material. The direct relation of this with image processing is clearly an enormous ouch! There are clear similarities between Eqn. 3.39 and Eqn. 3.37. This is the same functional form and allows for insight, analysis and parameter selection. The heat equation, Eqn. 3.37 is the anisotropic diffusion equation

$$\partial \mathbf{P} / \partial t = \nabla \cdot \left(c_{x,y}(t) \nabla \mathbf{P}_{x,y}(t) \right) \tag{3.40}$$

where $\nabla \cdot$ is the divergence operator (which essentially measures how the density within a region changes), with diffusion coefficient $c_{x,y}$. The diffusion coefficient applies to the local change in the image $\nabla \mathbf{P}_{x,y}(t)$ in different directions. If we have a lot of local change, we seek to retain it since the amount of change is the amount of boundary information. The diffusion coefficient indicates how much importance we give to local change: how much of it is retained. (The equation reduces to isotropic diffusion – Gaussian filtering - if the diffusivity is constant since $\nabla c = 0$.) There is no explicit solution to this equation. By approximating differentiation by differencing (this is explored more in Section 4.2) the rate of change of the image between time step *t* and time step *t*+1 we have

$$\partial \mathbf{P} / \partial t = \mathbf{P}(t+1) - \mathbf{P}(t) \tag{3.41}$$

This implies we have an iterative solution, and for later consistency we shall denote the image $\mathbf{P}$ at time step *t*+1 as $\mathbf{P}^{<t+1>} = \mathbf{P}(t+1)$ so we then have

$$\mathbf{P}^{<t+1>} - \mathbf{P}^{<t>} = \nabla \cdot \left(c_{x,y}(t) \nabla \mathbf{P}_{x,y}^{<t>} \right) \tag{3.42}$$

and again by approximation, using differences evaluated this time over the four compass directions North, South, East and West we have

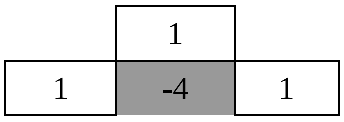
$$\nabla_N(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y-1} - \mathbf{P}_{x,y} \tag{3.43}$$

$$\nabla_S(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y+1} - \mathbf{P}_{x,y} \tag{3.44}$$

$$\nabla_E(\mathbf{P}_{x,y}) = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} \tag{3.45}$$

$$\nabla_W(\mathbf{P}_{x,y}) = \mathbf{P}_{x-1,y} - \mathbf{P}_{x,y} \tag{3.46}$$

The template and weighting coefficients for these are shown in Figure 3.27.



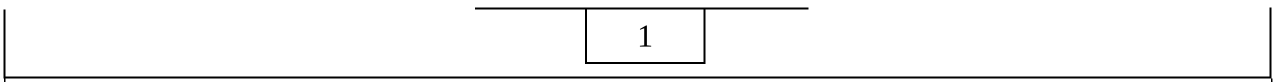


Figure 3.27 Approximations by Spatial Difference in Anisotropic Diffusion

When we use these as an approximation to the right hand side in Eqn. 3.42, we then have $\nabla \cdot \left(c_{x,y}(t)\nabla \mathbf{P}_{x,y}^{<t>}\right) = \lambda\left(cN_{x,y}\nabla_N(\mathbf{P}) + cS_{x,y}\nabla_S(\mathbf{P}) + cE_{x,y}\nabla_E(\mathbf{P}) + cW_{x,y}\nabla_W(\mathbf{P})\right)$ which gives

$$\mathbf{P}^{<t+1>} - \mathbf{P}^{<t>} = \lambda\left(cN_{x,y}\nabla_N(\mathbf{P}) + cS_{x,y}\nabla_S(\mathbf{P}) + cE_{x,y}\nabla_E(\mathbf{P}) + cW_{x,y}\nabla_W(\mathbf{P})\right) \Big| \mathbf{P} = \mathbf{P}_{x,y}^{<t>} \qquad (3.47)$$

where $0 \le \lambda \le 1/4$ and where $cN_{x,y}, cS_{x,y}, cE_{x,y}$ and $cW_{x,y}$ denote the conduction coefficients in the four compass directions. By rearrangement of this we obtain the equation we shall use for the anisotropic diffusion operator

$$\mathbf{P}^{<t+1>} = \mathbf{P}^{<t>} + \lambda\left(cN_{x,y}\nabla_N(\mathbf{P}) + cS_{x,y}\nabla_S(\mathbf{P}) + cE_{x,y}\nabla_E(\mathbf{P}) + cW_{x,y}\nabla_W(\mathbf{P})\right) \Big| \mathbf{P} = \mathbf{P}_{x,y}^{<t>} \qquad (3.48)$$

This shows that the solution is **iterative**: images at **one** time step (denoted by $^{<t+1>}$) are computed from images at the **previous** time step (denoted $^{<t>}$), given the initial condition that the first image is the original (noisy) image. Change (in time and in space) has been approximated as the difference between two adjacent points which gives the iterative equation and shows that the new image is formed by adding a controlled amount of the local change consistent with the main idea: that the smoothing process retains some of the boundary information.

We are not finished yet though, since we need to find values for $cN_{x,y}, cS_{x,y}, cE_{x,y}$ and $cW_{x,y}$. These are chosen to be a function of the difference along the compass directions, so that the boundary (edge) information is preserved. In this way we seek a function that tends to zero with increase in the difference (an edge or boundary with greater contrast) so that diffusion does not take place across the boundaries, keeping the edge information. As such we seek

$$\begin{aligned} cN_{x,y} &= g\left(\left\|\nabla_N(\mathbf{P})\right\|\right) \qquad (3.49)\\ cS_{x,y} &= g\left(\left\|\nabla_S(\mathbf{P})\right\|\right)\\ cE_{x,y} &= g\left(\left\|\nabla_E(\mathbf{P})\right\|\right)\\ cW_{x,y} &= g\left(\left\|\nabla_W(\mathbf{P})\right\|\right) \end{aligned}$$

and one function that can achieve this is

$$g(x,k) = e^{-x^2/k^2} \qquad (3.50)$$

(There is potential confusion with using the same symbol as for the Gaussian function, Eqn. 3.26, but we have followed the original authors' presentation.) This function clearly has the desired properties since when the values of the differences ∇ are large the function *g* is very small, conversely when ∇ is small then *g* tends to unity. *k* is another parameter whose value we have to choose: it controls the rate at which the conduction coefficient decreases with increasing difference magnitude. The effect of this parameter is shown in Figure 3.28. Here, the solid line is for the smaller value of *k* and the dotted one is for a larger value. Evidently, a larger value of *k* means that the contribution of the difference reduces less than for a smaller value of *k*. In both cases, the resulting function is near unity for small differences and near zero for large differences, as required. An alternative to this is to use the function

$$g2(x,k) = \frac{1}{1 + x^2/k^2} \qquad (3.51)$$

which has similar properties to the function in Eqn. 3.50.

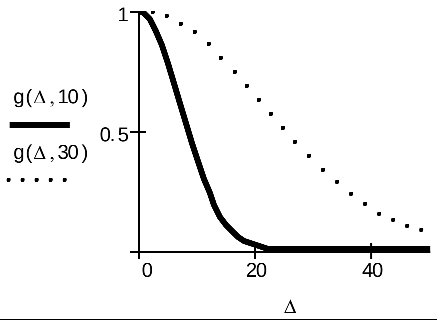


Figure 3.28 Controlling the Conduction Coefficient in Anisotropic Diffusion

This all looks rather complicated, so let's recap. First, we want to filter an image by retaining boundary points. These are retained according to the value of *k* chosen in Eqn. 3. 50. This function is operated in the four compass directions, to weight the brightness difference in each direction, Eqn. 3. 49. These contribute to an iterative equation which calculates a new value for an image point by considering the contribution from its four neighbouring points, Eqn. 3.48. This needs choice of one parameter λ. Further, we need to choose the number of iterations for which calculation proceeds. For information, Figure 3.26(b) was calculated over 20 iterations and we need to use sufficient iterations to ensure that convergence has been achieved. Figure 3.29 shows how we approach this. Figure 3.29 (a) is after a single iteration, (b) after 2, (c) after 5 and (d) after 10 and (e) after 20. Manifestly we could choose to reduce the number of iterations, accepting a different result – or even go further.

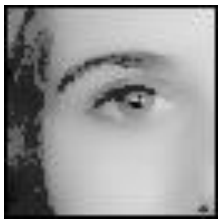
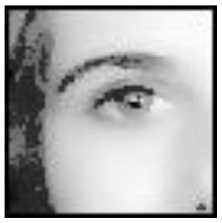
				
(a) one iteration	(b) two iterations	(c) five iterations	(d) ten iterations	(e) final result

Figure 3.29 Iterations of Anisotropic Diffusion

We also need to choose values for *k* and λ. By analogy, *k* is the conduction coefficient and low values preserve edges and high values allow diffusion (conduction) to occur – how much smoothing can take place. The two parameters are naturally inter-related, though λ largely controls the amount of smoothing. Given that low values of either parameter means that no filtering effect is observed, we can investigate their effect by setting one parameter to a high value and varying the other. In Figures 3.30(a)-(c) we use a high value of *k* which means that edges are not preserved and we can observe that different values of λ control the amount of smoothing. (A discussion of how this Gaussian filtering process is achieved can be inferred from Section 4.2.4.) Conversely, we can see how different values for *k* control the level of edge preservation in Figures 3.30(d)-(f) where some structures around the eye are not preserved for larger values of *k*.

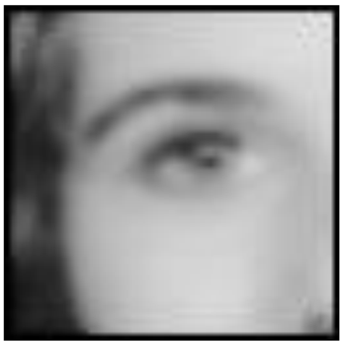
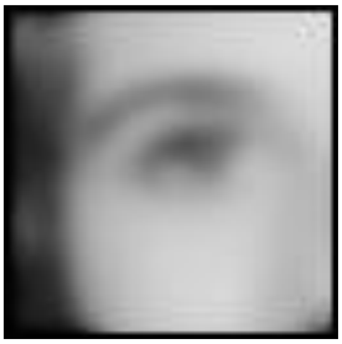
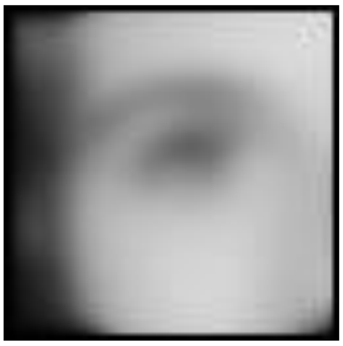
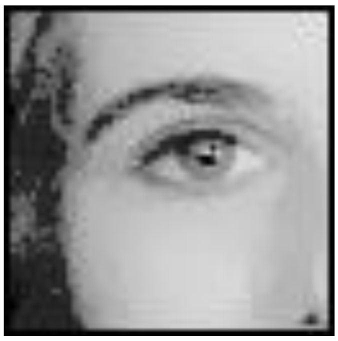
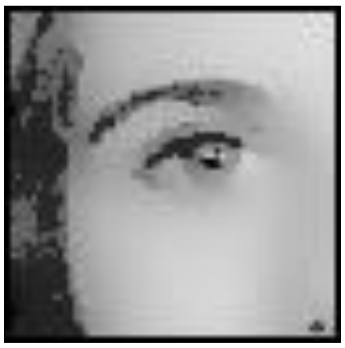
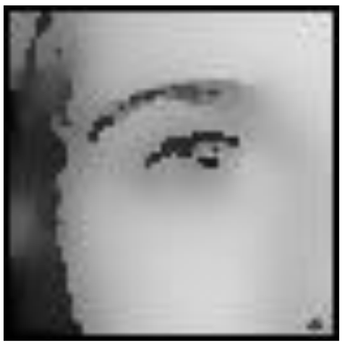
 **(a)** $k = 100$ and $\lambda = 0.05$	 **(b)** $k = 100$ and $\lambda = 0.15$	 **(c)** $k = 100$ and $\lambda = 0.25$
 **(d)** $k = 5$ and $\lambda = 0.25$	 **(e)** $k = 15$ and $\lambda = 0.25$	 **(f)** $k = 25$ and $\lambda = 0.25$

Figure 3.30 Applying Anisotropic Diffusion

By design, the result is akin with that of bilateral filtering (which it preceded) – indeed, the relationship has already been established [Barash02]. One of the major disadvantages is that anisotropic diffusion is an iterative process, the advantages include greater control of the filtering process. The original presentation of anisotropic diffusion [Perona90] is extremely lucid and well worth a read if you consider selecting this technique. Naturally, it has greater detail on formulation and on analysis of results, than space here allows for (and is suitable at this stage). Amongst other papers on this topic, one [Black98] studied the choice of conduction coefficient leading to a function which preserves sharper edges and improves automatic termination. A more recent study has considered parameter and function choice [Tsiotsios13]. As ever, with techniques that require much computation there have been approaches which speed implementation, or achieve similar performance faster (e.g. [Fischl99]).

3.5.6 Comparison of Smoothing Operators

3.5.7 Force Field Transform

There are of course many more image filtering operators; we have so far covered those that are amongst the most popular. There are others which offer alternative insight, sometimes developed in the context of a specific application. By way of example, Hurley developed a transform called the force field transform [Hurley02, Hurley05] which uses an analogy to gravitational force. The transform pretends that each pixel exerts a force on its neighbours which is inversely proportional to the square of the distance between them. This generates a force field where the net force at each point is the aggregate of the forces exerted by all the other pixels on a "unit test pixel" at that point.

This very large scale summation affords very powerful averaging which reduces the effect of noise. The approach was developed in the context of ear biometrics, recognising people by their ears, which has unique advantage as a biometric in that the shape of people's ears does not change with age, and of course – unlike a face - ears do not smile! The force field transform of an ear, Figure 3.32(a), is shown in Figure 3.32(b). Here, the averaging process is reflected in the reduction of the effects of hair. The transform itself has highlighted ear structures, especially the top of the ear and the lower 'keyhole' (the notch).

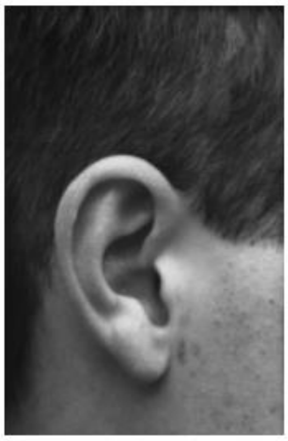
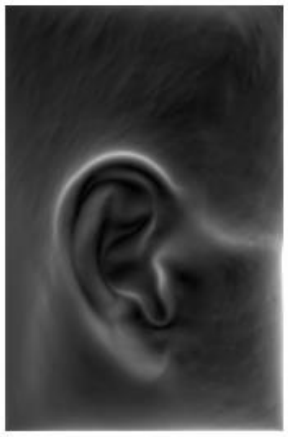
 **(a)** image of ear	 **(b)** magnitude of force field transform

Figure 3.32 Illustrating the Force Field Transform

The image shown is actually the magnitude of the force field. The transform itself is a vector operation, and includes direction [Hurley02]. The transform is expressed as the calculation of the force $\mathbf{F}$ between two points at positions $\mathbf{r}_i$ and $\mathbf{r}_j$ which is dependent on the value of a pixel at point $\mathbf{r}_i$ as

$$\mathbf{F}_i(\mathbf{r}_j) = \mathbf{P}(\mathbf{r}_i)\frac{\mathbf{r}_i - \mathbf{r}_j}{\left|\mathbf{r}_i - \mathbf{r}_j\right|^3} \qquad (3.52)$$

which assumes that the point $\mathbf{r}_j$ is of unit "mass". This is a directional force (which is why the inverse square law is expressed as the ratio of the difference to its magnitude cubed) and the magnitude and directional information has been exploited to determine an ear 'signature' by which people can be recognised. In application, Equation 3.52 can be used to define the coefficients of a template that is convolved with an image (implemented by the FFT to improve speed), as with many of the techniques that have been covered in this Chapter; a Mathcad implementation is also given [Hurley02]. Note that this transform actually exposes low level features (the boundaries of the ears) which are the focus of the next Chapter. How we can determine shapes is a higher level process, and how the processes by which we infer or recognise identity from the low-and the high-level features will be covered in Chapter 12.

3.6 Chapter 3 References

[Abd-Elmoniem 2002] Abd-Elmoniem, K. Z., Youssef, A. B., & Kadah, Y. M. Real-time speckle reduction and coherence enhancement in ultrasound imaging via nonlinear anisotropic diffusion. *IEEE Trans. on Biomed. Eng.*, **49**(9), 997-1014, 2002

[Barash02] D. Barash, A Fundamental Relationship between Bilateral Filtering, Adaptive Smoothing and the Nonlinear Diffusion Equation, *IEEE Trans. on PAMI*, **24**(6), pp 844-849, 2002

[Black98] Black, M. J., Sapiro, G., Marimont, D. H., and Meeger, D., Robust Anisotropic Diffusion, *IEEE Trans. on Image Processing*, **7**(3), pp 421-432, 1998

[Bovik87] Bovik A. C., Huang T. S., Munson D. C., The Effect Of Median Filtering on Edge Estimation and Detection, *IEEE Trans. on PAMI*, **9**(2), pp 181-194, 1987

[Buades05] Buades, A., Coll, B., and Morel, J. M., A non-local algorithm for image denoising. *Proc. CVPR*, 2005, **2**, pp 60-65, 2005

[Buades11] Buades, A., Coll, B., and Morel, J. M. Non-local means denoising. *Image Processing On Line*, **1**, pp 208-212, 2011

[Campbell69] Campbell, J. D., *Edge Structure and the Representation of Pictures*, PhD Thesis, Univ. Missouri Columbia USA, 1969

[Castleman96] Castleman, K. R., *Digital Image Processing*, Prentice Hall Inc., Englewood Cliffs N. J. USA, 1996

[Chan05] Chan, T., and Shen, J., *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, Society for Industrial and Applied Mathematics, 2005

[Chatterjee10] Chatterjee, P., and Milanfar, P., Is Denoising Dead? *IEEE Trans. on IP*, **19**(4), pp 895-911, 2010

[Coupé09] Coupé, P., Hellier, P., Kervrann, C., & Barillot, C. Nonlocal Means-based Speckle Filtering for Ultrasound Images. *IEEE Trans. on IP*, **18**(10), pp 2221-2229, 2009

[Chityala15] Chityala, R., and Pudipeddi, S., *Image Processing and Acquisition using Python*, CRC Press, Boca Raton FL USA, 2015

[Cummings11] Cummings, A. H., Nixon, M. S., Carter, J. N., The image ray transform for structural feature detection, *Pattern Recognition Letters*, **32**(15), pp 2053-2060, 2011

[Dabov07] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K., Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering, *IEEE Trans. on IP*, **16**(8), pp 2080-2095, 2007

[Davies88] Davies, E. R., On the Noise Suppression Characteristics of the Median, Truncated Median and Mode Filters, *Pattern Recog. Lett.*, **7**(2), pp 87-97, 1988

[Davies17] Davies, E. R., *Computer Vision: Principles, Algorithms, Applications, Learning*, Academic Press, 5th Edition, 2017

[Dowson11] Dowson, N., & Salvado, O. Hashed nonlocal means for rapid image filtering. *IEEE Trans. on PAMI*, **33**(3), 485-499. 2011

[Evans95] Evans, A. N., and Nixon M. S., Mode Filtering to Reduce Ultrasound Speckle for Feature Extraction, *Proc. IEE - Vision, Image and Signal Processing*, **142**(2), pp 87-94, 1995

[Evans96] Evans, A. N., and Nixon M. S., Biased Motion-Adaptive Temporal Filtering for Speckle Reduction in Echocardiography, *IEEE Trans. Medical Imaging*, **15**(1), pp 39-50, 1996

[Fischl99] Fischl, B., and Schwartz, E. L. Adaptive Nonlocal Filtering: a Fast Alternative to Anisotropic Diffusion for Image Enhancement, *IEEE Trans. on PAMI*, **21**(1) pp 42-48, 1999

[Glasbey93] Glasbey, C. A., An Analysis of Histogram-Based Thresholding Algorithms, *CVGIP-Graphical Models and Image Processing*, **55** (6), pp 532-537, 1993

[Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4th Edition, Pearson Education, 2017

[Gonzalez09] Gonzalez, R. C., Woods, R. E., and Eddins, S. L., *Digital Image Processing using MATLAB*, Prentice Hall, 2nd Edition, 2009

[Ham19] Ham, B., Cho, M., & Ponce, J. Robust guided image filtering using nonconvex potentials. *IEEE Trans. on PAMI*, **40**(1), pp 192-207, 2018

[Hearn97] Hearn, D., and Baker, M. P., *Computer Graphics C Version*, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, NJ USA, 1997

[Hodgson85] Hodgson, R. M., Bailey, D. G., Naylor, M. J., Ng A., and McNeill S. J., Properties, Implementations and Applications of Rank Filters, *Image and Vision Computing*, **3**(1), pp 3-14, 1985

[Huang79] Huang, T., Yang, G., and Tang, G., A Fast Two-dimensional Median Filtering Algorithm, *IEEE Trans. on ASSP*, **27**(1), pp 13-18, 1979

[Hurley02] Hurley, D. J., Nixon, M. S. and Carter, J. N., Force Field Energy Functionals for Image Feature Extraction, *Image and Vision Computing*, **20**, pp 311-317, 2002

[Hurley05] Hurley, D. J., Nixon, M. S. and Carter, J. N., Force Field Feature Extraction for Ear Biometrics, *Computer Vision and Image Understanding*, **98**(3), pp 491-512, 2005

[Kervrann06] Kervrann, C., and Boulanger, J. Optimal spatial adaptation for patch-based image denoising. *IEEE Trans. on IP*, 15(10), 2866-2878, 2006

[Koenderink84] Koenderink, J., The Structure of Images, *Biological Cybernetics*, **50**, pp 363-370, 1984

[Lee90] Lee, S. A., Chung, S. Y. and Park, R. H., A Comparative Performance Study of Several Global Thresholding Techniques for Segmentation, *CVGIP*, **52**, pp 171-190, 1990

[Levin11] Levin, A., and Nadler, B., Natural image denoising: optimality and inherent bounds, Proc. *CVPR*, 2011

[Loupas87] Loupas, T., and McDicken, W. N., Noise Reduction in Ultrasound Images by Digital Filtering, *British Journal of Radiology*, **60**, pp 389-392, 1987

[Montiel95] Montiel, M. E., Aguado, A. S., Garza, M., and Alarcón, J., Image Manipulation using M-filters in a Pyramidal Computer Model, *IEEE Trans. on PAMI*, **17**(11), pp 1110-1115, 1995.

[OpenCV-TM] OpenCV Template Operator https://github.com/opencv/opencv/blob/master/modules/imgproc/src/filter.cpp

[Otsu79] Otsu, N., A Threshold Selection Method from Gray-Level Histograms, *IEEE Trans. on SMC*, **9**(1), pp 62-66, 1979

[Paris08] Paris, S., and Durand, F., A Fast Approximation of the Bilateral Filter Using a Signal Processing Approach, *International J. Computer Vision* , **81**(1), pp 24 – 52, 2008

[Perona90] Perona, P., and Malik, J., Scale-space and Edge Detection using Anisotropic Diffusion, *IEEE Trans. on PAMI*, **17**(7), pp 620-639, 1990

[Perona90] Perona, P., and Malik, J., Scale-Space and Edge Detection using Anisotropic Diffusion, *IEEE Trans. on PAMI*, **17**(7), pp 629-639, 1990

Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ... and Zuiderveld, K. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, **39**(3), 355-368, 1987

[Rosenfeld82] Rosenfeld, A and Kak A. C., *Digital Picture Processing*, 2nd Edition, Vols. 1 and 2, Academic Press Inc., Orlando FL USA, 1982

[Rosin01] Rosin, P. L., Unimodal Thresholding, *Pattern Recognition*, **34(**11), pp 2083-2096, 2001

[Russ11] Russ, J. C., *The Image Processing Handbook*, 6th Edition, CRC Press (IEEE Press), Boca Raton FL USA, 2011

[Russ17] Russ, J. C., and Russ, J. C. *Introduction to Image Processing and Analysis*, CRC Press, Boca Raton FL USA, 2017

[Seul00] Seul, M., O'Gorman, L., and Sammon, M. J., *Practical Algorithms for Image Analysis: Descriptions, Examples, and Code*, Cambridge University Press, Cambridge UK, 2000

[Sahoo88] Sahoo, P. K., Soltani, S. Wong, A. K. C., and Chen, Y. C., Survey of Thresholding Techniques, *CVGIP*, **41**(2), pp 233-260, 1988

[Serra86] Serra, J., *Introduction to Mathematical Morphology*, Computer Vision, Graphics and Image Processing, **35**, pp 283-305, 1986

[Soille13] Soille, P., *Morphological Image Analysis: Principles and Applications*, Springer Science & Business Media, Heidelberg, Germany, 2013

[Shankar86] Shankar, P. M., Speckle Reduction in Ultrasound B Scans using Weighted Averaging in Spatial Compounding, *IEEE Trans. on Ultrasonics, Ferroelectrics and Frequency Control*, **33**(6), pp754-758, 1986

[Sternberg86] Sternberg, S. R., Gray Scale Morphology, *Computer Vision, Graphics and Image Processing*, **35**, pp 333-355, 1986

[Staal04] Staal, J., Abramoff, M. Niemeijer, M.,Viergever, M., van Ginneken, B., Ridge based vessel segmentation in color images of the retina**,** *IEEE Trans. Med. Imag.*, **23**, pp 501-509, 2004

[Tomasi98] C. Tomasi, and R. Manduchi, Bilateral filtering for Gray and Color images, *Proc ICCV 1998*, pp 839-846, Bombay, India, 1998

[Trier95] Trier, O. D., and Jain, A. K., Goal-Directed Evaluation of Image Binarisation Methods, *IEEE Trans. on PAMI*, **17**(12), pp 1191-1201, 1995

[Tsiotsios13] Tsiotsios, C., and Petrou, M. On the choice of the parameters for anisotropic diffusion in image processing. *Pattern Recog*., **46**(5), 1369-1381, 2013

[Weiss06] Weiss, B., Fast median and bilateral filtering, *Proc. ACM SIGGRAPH 2006*, pp 519-526, 2006

[Witkin83] Witkin, A., Scale-Space Filtering: a New Approach to Multi-Scale Description, *Proc. Int. Joint Conf. Artificial Intelligence*, pp 1019-1021, 1983

[Zhang17] Zhang, K., Zuo, K., Chen, Y., Meng, D., Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising, *IEEE Trans. IP*, 2017