implemented operation corresponds simply to adding the products of each weight with the corresponding pixels of the image. In order to process the entire image, the window is shifted by positioning its origin onto each of the image pixels, generating the corresponding output pixels (see Figure 3.18(c)).

Clearly, the size of the window can be changed, implying neighborhoods of different sizes to be taken into account. For instance, a neighborhood of size 5×5 would work in an analogous way, except that the weights should be 1/25. Such a larger neighborhood allows more distant pixels to also influence the output value. In other words, the *analyzing spatial scale* becomes larger. The practical effect is that the output image becomes smoother. It is observed that choosing a suitable analyzing scale is not a problem that has a unique or simple solution.

It is important to note that processing the pixels on the border of the image presents a special problem because when the origin of the operator is placed over the border pixels, some positions of the window fall outside the image. In fact, not only the outermost pixels are affected, but those near (but not on) the border also present this problem (the number of outer pixel layers affected by this problem depends on the size of the operator). The following three typical solutions can be considered in such situations:

(1) The outer layers are simply ignored, i.e., the filter operation is carried out only for the inner pixels. As a consequence, the resulting image is typically smaller than the original, although its size can be kept by filling up the borders with zeroes.
(2) The image is augmented with outer layers in order to complement the necessary number of neighboring pixels required by the filtering process. The value of these extra pixels have to be set arbitrarily (e.g., to zero), which can adversely affect the results.
(3) The image is assumed to be periodic (namely a thorus), so that if a position of the operator falls off the image, the corresponding operation is carried out over the pixel at the other side of the image. This type of structure is naturally implemented by linear filtering techniques based on the Fourier transform [Morrison, 1994].

The choice of one of the above solutions depends on each specific problem. Algorithms can be easily altered in order to implement any of the above discussed solutions. In this sense, average filtering is a special case of *linear filtering*, which can be modelled by 2D convolutions:

$$f(p,q) = \frac{1}{MN}\sum_m \sum_n h(m,n)\, g(p-m, q-n)$$

where $M \times N$ is the size of the input image.

In fact, there are many image processing tasks carried out in an analogous way, varying only as far as the filtering function is concerned. The definition of different sets of weights can lead to completely different results. For instance, special operators can be defined in order to differentiate the image or

As in the one-dimensional case, we define the *Fourier pair* as follows:

$$g(x, y) \leftrightarrow G(u, v)$$

As an example, the 2D Dirac delta function is defined as

$$\delta(x, y) = \begin{cases} \text{not defined} & \text{if } x = 0 \text{ and } y = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \boldsymbol{d}(x, y) dx dy = 1$$

and its Fourier transform is calculated as

$$\Im\{\boldsymbol{d}(x, y)\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \boldsymbol{d}(x, y) \exp\{-j2\boldsymbol{p}(ux + vy)\} dx dy = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \boldsymbol{d}(x, y) \exp\{0\} dx dy =$$

$$= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \boldsymbol{d}(x, y) dx dy = 1$$

Since the respective inverse can be verified to exist, we have $\delta(x, y) \leftrightarrow 1$.

The 2D Fourier transform exhibits a series of useful and practical properties in image processing and analysis, most of which are analogous to the 1D Fourier properties explained in Chapter 2. Table 3.2 summarizes some of the most useful 2D Fourier properties (see also [Castleman, 1996; Gonzalez and Woods, 1993]).

Table 3.2: *2D Fourier transform properties assuming* $g(x, y) \leftrightarrow G(u, v)$.

Property	Description
Separability (DFT)	The discrete Fourier transform can be computed in terms of 1D Fourier transforms of the image rows followed by 1D transforms of the columns (or vice-versa).
Spatial Translation (Shifting)	$g(x - x_0, y - y_0) \leftrightarrow \exp[-j2\boldsymbol{p}(ux_0 + vy_0)]G(u, v)$
Frequency Translation (Shifting)	$\exp[j2\boldsymbol{p}(xu_0 + yv_0)]g(x, y) \leftrightarrow G(u - u_0, v - v_0)$
Conjugate Symmetry	If $g(x, y)$ is real, then $G(u, v) = G^*(-u, -v)$
Rotation by θ	$g(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta) \leftrightarrow$ $\leftrightarrow G(u\cos\theta + v\sin\theta, -u\sin\theta + v\cos\theta)$
Linearity – Sum	$g_1(x, y) + g_2(x, y) \leftrightarrow G_1(u, v) + G_2(u, v)$

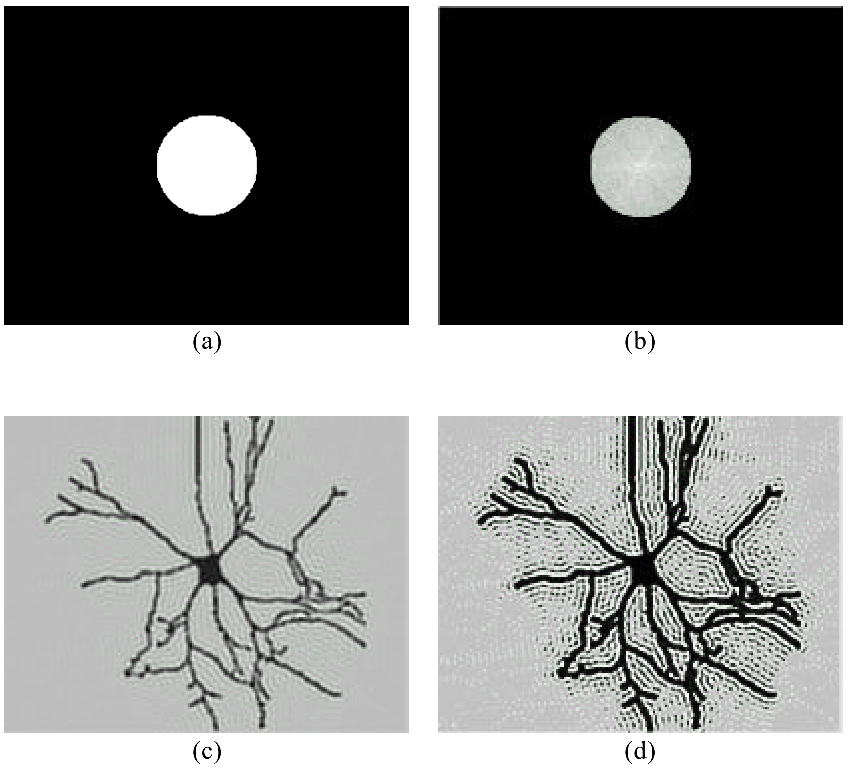

Figure 3.22: *Frequency domain 2D filter defined by Equation (3.1) (a); Fourier transform of the image in Figure 3.21(b) filtered by this filter function (b); the resulting filtered image (c). The histogram equalized version of (c) is shown in (d). It is important to observe that while (b) shows only the filtered Fourier modulus for visualizations sake, the filtering is actually carried out on the complex coefficients. (The neural cell in (c) has been reprinted from Journal of Neuroscience Methods, 27, T. G. Smith Jr., W. B. Marks, G. D. Lange, W. H. Sheriff Jr. and E. A. Neale, A Fractal Analysis of Cell Images, 173-180, Copyright (1989), with permission from Elsevier Science.)*

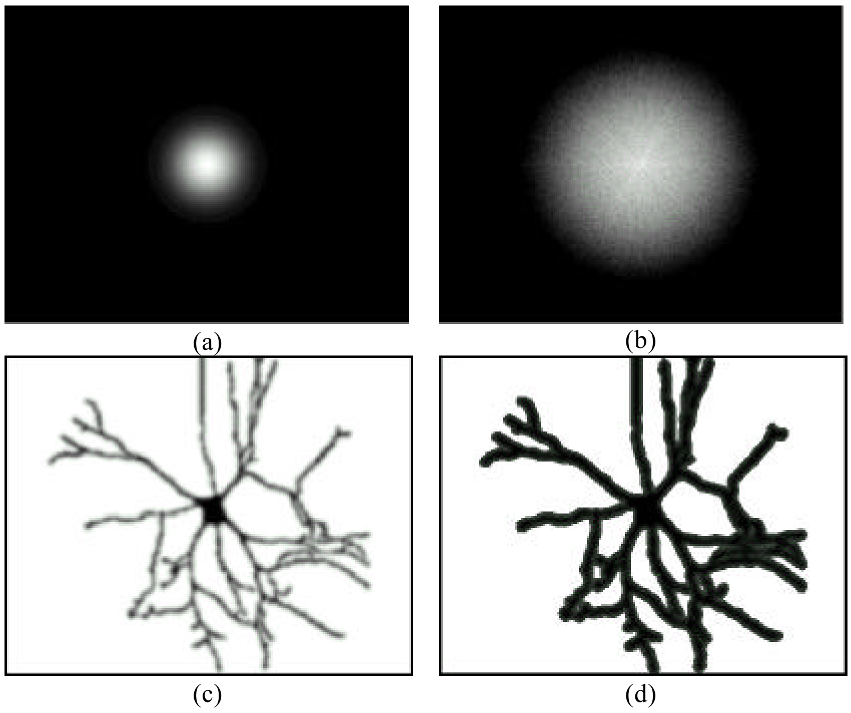

Figure 3.23: *Frequency domain 2D Gaussian filter (a); Fourier transform of Figure 3.21(b) filtered by this Gaussian function (b); and the resulting filtered image(c). The histogram equalized version of (c) is shown in (d). (The neural cell in (c) has been reprinted from Journal of Neuroscience Methods, 27, T. G. Smith Jr., W. B. Marks, G. D. Lange, W. H. Sheriff Jr. and E. A. Neale, A Fractal Analysis of Cell Images, 173-180, Copyright (1989), with permission from Elsevier Science.)*

3.2.6 Median and Other Nonlinear Filters

Another classic technique for image filtering is known as *median filtering*. This technique can be understood similarly to the previously discussed average filtering, though the mathematical implications are different because of its nonlinear nature. Recall that the linear filtering approach is based on placing the origin of an operator at each pixel of the image and carrying out a weighted sum between the mask weights and the respective pixels under the mask. As far as median filtering is concerned, the window operator does not have any weight. Instead, the pixels under the operator are sorted[5], and the middle value (i.e., the median) is selected to substitute the reference pixel. Figure 3.24 illustrates this process. In this example, as in the sorted sequence

[5] Information about efficient sorting algorithms can be found in [Langsam et al., 1996].

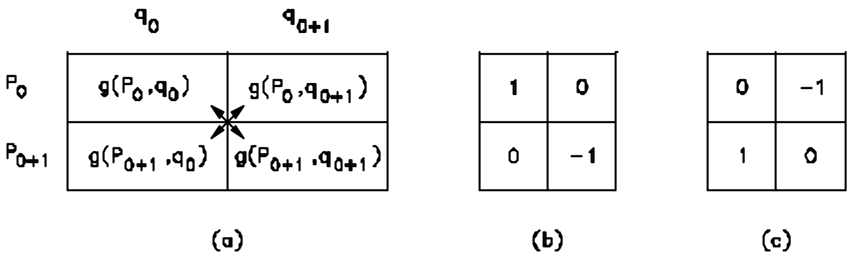

Figure 3.29: *The Roberts operator.*

Let g be a digital image. The value of the *Roberts operator* at (p_0,q_0) is typically defined as

$$r(p_0,q_0)=\sqrt{\left(g(p_0,q_0)-g(p_0+1,q_0+1)\right)^2+\left(g(p_0+1,q_0)-g(p_0,q_0+1)\right)^2}$$

It is observed that a number of alternative definitions for this operator can be found in the related literature. Schalkoff [1989], for example, defines the Roberts operator as

$$r(p_0,q_0)=\max\left\{\left|g(p_0,q_0)-g(p_0+1,q_0+1)\right|,\left|g(p_0+1,q_0)-g(p_0,q_0+1)\right|\right\}$$

On the other hand, Angulo and Madrigal [1986] present the following definition:

$$r(p_0,q_0)=\left|g(p_0,q_0)-g(p_0+1,q_0+1)\right|+\left|g(p_0+1,q_0)-g(p_0,q_0+1)\right|$$

These two definitions may be preferable for real-time applications since they do not involve the square root operation. Finally, Castleman [1996] adopts

$$r=\sqrt{\left(\sqrt{g(p_0,q_0)}-\sqrt{g(p_0+1,q_0+1)}\right)^2+\left(\sqrt{g(p_0+1,q_0)}-\sqrt{g(p_0,q_0+1)}\right)^2}$$

In the latter definition, the square root of each pixel is taken before the calculus of the differences. Castleman argues that this operation makes the Roberts operator more similar to edge detection in the human visual system.

Algorithm: ***Belong***

```
function belong( image, seed_gl, current_pixel )
bel = FALSE;
If ( (current_pixel.p >= 1) AND
      (current_pixel.p <= P) AND
      (current_pixel.q >= 1) AND
      (current_pixel.q <= Q))
   then
      If ( abs( image[current_pixel.p,
           Current_pixel.q] - seed_gl ) < ERROR )
      then
         bel = TRUE;
```

The function `belong` is initialized with `FALSE`, so that if nothing happens in the following `if`, it returns `FALSE`, indicating that the pixel does not belong to the region. The first `if` verifies that the pixel coordinates are valid, assuming that the indexes *p* and *q* of the image vary from 1 to `P` and 1 to `Q`, respectively. The second `if` checks the difference between the gray-levels of the current pixel and that of the seed, where `ERROR` is a constant that must be set *a priori*. Knowledge about each specific problem can be incorporated into the above procedure. For instance, if the objects to be segmented in the image are known not to present a diameter larger than a given value, this condition can be easily incorporated into the second `if` of the function `belong` by testing the distance between the current pixel and the seed (or between the current pixel and the farthest one still belonging to the object).

In addition, a mechanism has to be devised so that the already considered pixels are not revisited. This can be easily accomplished as follows. First, it is important to note that the output of the region growing procedure is a labeled image and, therefore, we must define some labeling representation. This can be achieved by marking the pixels that belong to the growing region with a special value, preferably one that is not a valid gray-level. For instance, if the input gray-levels vary from 0 and 255, we could mark the growing region with, say 1000, in such a way that a simple thresholding operation over the output image with a threshold of 300 would produce a binary image with the segmented region of interest. Furthermore, this labeling scheme also avoids the algorithm to reconsider already visited pixels, because if the label to be assigned to the pixels that belong to the region is large enough, the difference between the labeled pixels and the seed will always exceed the threshold `ERROR`. Therefore, the procedure `label(image, current_pixel)` only has to assign a (large enough) marking value to the `current_pixel` of the `image`.