CHAPTER 11

FREQUENCY-DOMAIN FILTERING

WHAT WILL WE LEARN?

- Which mathematical tools are used to represent an image's contents in the 2D frequency domain?
- What is the Fourier transform, what are its main properties, and how is it used in the context of frequency-domain filtering?
- How are image processing filters designed and implemented in the frequency domain?
- What are the differences between low-pass and high-pass filters (HPFs)?
- What are the differences among ideal, Butterworth, and Gaussian filters?

11.1 INTRODUCTION

This chapter builds upon the ideas introduced in Section 2.4.4, which state that some image processing tasks can be performed by transforming the input images to a different domain, applying selected algorithms in the transform domain, and eventually applying the inverse transformation to the result. In this chapter, we are particularly interested in a special case of operations in the transform domain, which we call *frequency-domain filtering*. Frequency-domain filters work by following a straightforward sequence of steps (Figure 11.1):

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

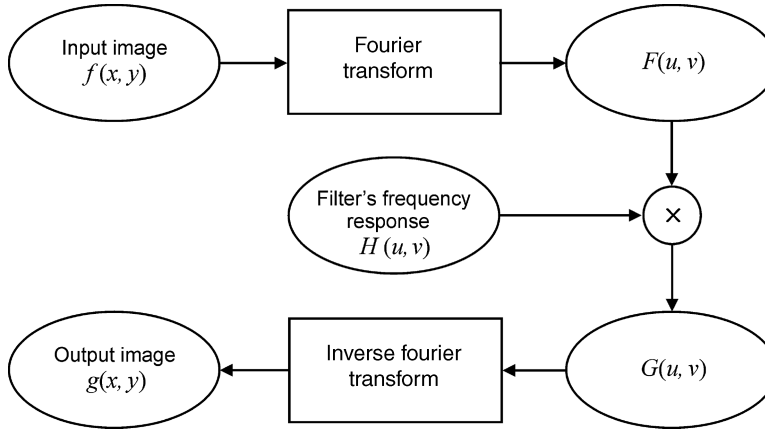


FIGURE 11.1 Frequency-domain operations.

1. The input image is transformed to a 2D frequency-domain representation using the 2D Fourier transform (FT).
2. A filter of specific type (e.g., ideal, Butterworth, Gaussian) and behavior (e.g., low pass, high pass) is specified and applied to the frequency-domain representation of the image.
3. The resulting values are transformed back to the 2D spatial domain by applying the inverse 2D Fourier transform, producing an output (filtered) image.

The mathematical foundation of frequency-domain techniques is the *convolution theorem*. Let $g(x, y)$ be an image obtained by the convolution[1] (denoted by the $*$ symbol) of an image $f(x, y)$ with a linear, position invariant operator $h(x, y)$, that is,

$$g(x, y) = f(x, y) * h(x, y) \tag{11.1}$$

From the convolution theorem, the following frequency-domain relation holds:

$$G(u, v) = F(u, v)H(u, v) \tag{11.2}$$

where G, F, and H are the Fourier transforms of g, f, and h, respectively.

Many image processing problems can be expressed in the form of equation (11.2). In a noise removal application, for instance, given $f(x, y)$, the goal, after computing $F(u, v)$, will be to select $H(u, v)$ such that the desired resulting image,

$$g(x, y) = \mathfrak{F}^{-1}\left[F(u, v)H(u, v)\right] \tag{11.3}$$

where $\mathfrak{F}^{-1}$ is the inverse 2D Fourier transform operation, exhibits a reduction in the noisy contents present in the original image $f(x, y)$. For certain types of noise,

[1] Two-dimensional discrete convolution was introduced in Section 10.2.2.

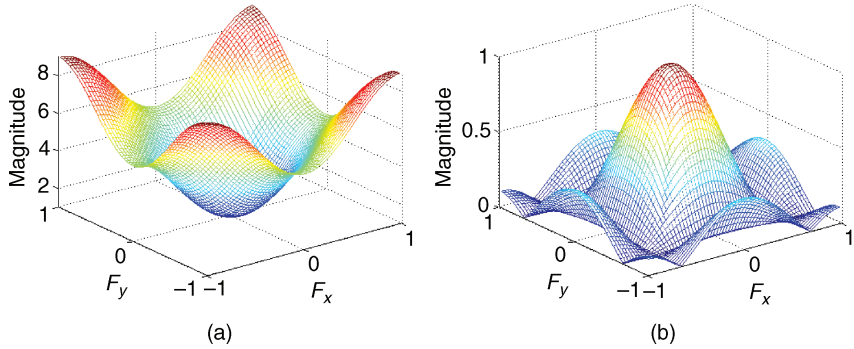


FIGURE 11.2 Two examples of response functions for frequency-domain filters: (a) low-pass filter equivalent to a 3×3 average filter in the spatial domain; (b) high-pass filter equivalent to a 3×3 composite Laplacian sharpening filter in the spatial domain.

this result could be achieved using a low-pass Butterworth filter (Section 11.3.3), for example.

There are two options for designing and implementing image filters in the frequency domain using MATLAB and IPT:

1. Obtain the frequency-domain filter response function from spatial filter convolution mask. The IPT has a function that does exactly that: `freqz2`. Figure 11.2 shows examples of such response functions for the 3×3 average filter described by equation (10.9) and the 3×3 composite Laplacian sharpening filter described by equation (10.16).
2. Generate filters directly in the frequency domain. In this case, a meshgrid array (of the same size as the image) is created using the MATLAB function `meshgrid`. This is the method used in the tutorials of this chapter.

11.2 FOURIER TRANSFORM: THE MATHEMATICAL FOUNDATION

The FT is a fundamental tool in signal and image processing.[2] In this section, we discuss the mathematical aspects of 2D transforms in general and then introduce the 2D FT and its main properties.

11.2.1 Basic Concepts

A *transform* is a mathematical tool that allows the conversion of a set of values to another set of values, creating, therefore, a new way of representing the same

[2] A complete, detailed analysis of the FT and associated concepts for 1D signals is beyond the scope of this book. Refer to the section "Learn More About It" for useful pointers.

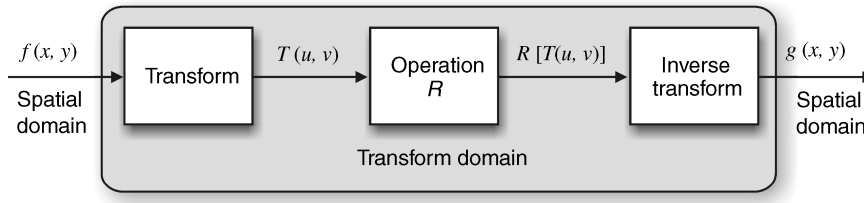


FIGURE 11.3 Operations in a transform domain.

information. In the field of image processing, the original domain is referred to as *spatial domain*, whereas the results are said to lie in the *transform domain*. The motivation for using mathematical transforms in image processing stems from the fact that some tasks are best performed by transforming the input images, applying selected algorithms in the transform domain, and eventually applying the inverse transformation to the result (Figure 11.3).

Linear 2D transforms can be expressed in generic form in terms of a *forward transform* $T(u, v)$ as

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot r(x, y, u, v) \tag{11.4}$$

where u ($u = 0, 1, 2, \ldots, M - 1$) and v ($v = 0, 1, 2, \ldots, N - 1$) are called *transform variables*, $f(x, y)$ is the input image, x and y are the row and column dimensions of $f(x, y)$ (as described in Section 2.1), and $r(x, y, u, v)$ is called the *forward transformation kernel*, sometimes expressed as a collection of *basis images*.

The original image $f(x, y)$ can be recovered by applying the *inverse transform* of $T(u, v)$:

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) \cdot s(x, y, u, v) \tag{11.5}$$

where $x = 0, 1, 2, \ldots, M - 1$, $y = 0, 1, 2, \ldots, N - 1$, and $s(x, y, u, v)$ is called the *inverse transformation kernel*.

The combination of equations (11.4) and (11.5) is usually referred to as a *transform pair*. Different mathematical transforms use different transformation kernels. For the sake of this chapter, we shall limit the discussion to the Fourier Transform.[3]

[3]The interested reader should refer to the section "Learn More About It" for useful references on other mathematical transforms.

11.2.2 The 2D Discrete Fourier Transform: Mathematical Formulation

The 2D Fourier transform of an image is a special case of equations (11.4) and (11.5) where

$$r(x, y, u, v) = \exp[-j2\pi(ux/M + vy/N)] \tag{11.6}$$

and

$$s(x, y, u, v) = \frac{1}{MN} \exp[j2\pi(ux/M + vy/N)] \tag{11.7}$$

where $j = \sqrt{-1}$.

Substituting equations (11.6) and (11.7) into equations (11.4) and (11.5), we have

$$F(u, v) = T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot \exp[-j2\pi(ux/M + vy/N)] \tag{11.8}$$

and

$$f(x, y) = \mathfrak{F}^{-1}[F(u, v)] = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v) \cdot \exp[j2\pi(ux/M + vy/N)] \tag{11.9}$$

Remember from our discussion in Chapter 2 that array indices in MATLAB start at 1; therefore, `F(1,1)` and `f(1,1)` in MATLAB correspond to $F(0, 0)$ and $f(0, 0)$ in equations (11.8) and (11.9).

The value of the 2D FT at the origin of the frequency domain (i.e, $F(0, 0)$) is called the *DC component* of the FT, a terminology borrowed from electrical engineering, where *DC* means *direct current*, that is, alternating current of zero frequency. The *DC* component of the FT is the product of the average intensity value of $f(x, y)$ and a factor (MN). The FT is usually implemented using a computationally efficient algorithm known as *fast Fourier transform* (FFT) [Bri74].

The 2D FT of an image is an array of complex numbers, usually expressed in polar coordinates, whose components are *magnitude* (or *amplitude*) ($|F(u, v)|$) and *phase* ($\phi(u, v)$), which can be expressed as

$$|F(u, v)| = \sqrt{[R(u, v)]^2 + [I(u, v)]^2} \tag{11.10}$$

and

$$\phi(u, v) = \arctan\left[\frac{I(u, v)}{R(u, v)}\right] \tag{11.11}$$

where $R(u, v)$ and $I(u, v)$ are the real and imaginary parts of $F(u, v)$, that is, $F(u, v) = R(u, v) + jI(u, v)$.

In MATLAB

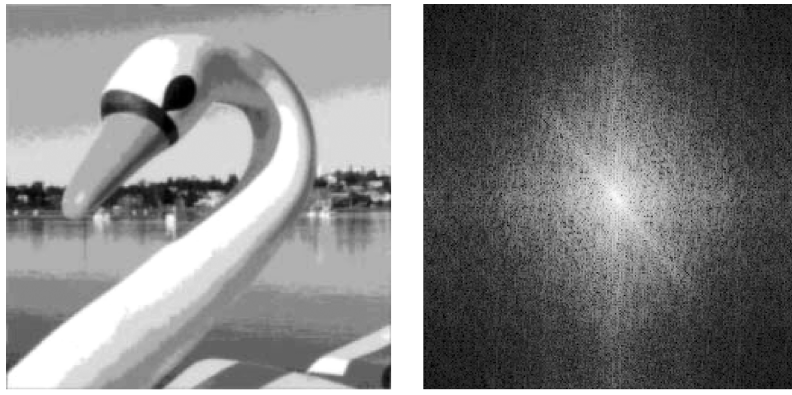
The 2D FT and its inverse are implemented in MATLAB by functions `fft2` and `ifft2`, respectively. 2D FT results are usually shifted for visualization purposes in such a way as to position the zero-frequency component at the center of the figure (exploiting the periodicity property of the 2D FT, described in Section 11.2.3). This can be accomplished by function `fftshift`. These functions are extensively used in the tutorials of this chapter. MATLAB also includes function `ifftshift`, whose basic function is to undo the results of `fftshift`.

■ EXAMPLE 11.1

Figure 11.4 shows an image and its frequency spectrum (the log transformation of the amplitude component of its 2D FT coefficients). The transformed version of the image clearly bears no resemblance to the original version. Moreover, it does not provide any obvious visual hints as to what the original image contents were. Do not be discouraged by these facts, though. As you will soon see, these apparent limitations of the transformed version of the image will not prevent us from designing frequency-domain filters to achieve specific goals. The resulting image—obtained after applying the inverse transform—will allow us to judge whether those goals were successfully achieved or not.

MATLAB code:

```
I = imread('Figure11_04_a.png');
Id = im2double(I);
ft = fft2(Id);
ft_shift = fftshift(ft);
imshow(log(1 + abs(ft_shift)), [])
```



(a) (b)

FIGURE 11.4 (a) Original image (256 × 256 pixels); (b) Fourier spectrum of the image in (a).

11.2.3 Summary of Properties of the Fourier Transform

In this section, we present selected properties of the 2D FT and its inverse that are particularly important in image processing.

Linearity The Fourier transform is a linear operator, that is,

$$\mathfrak{F}[a \cdot f1(x, y) + b \cdot f2(x, y)] = a \cdot F1(u, v) + b \cdot F2(u, v) \tag{11.12}$$

and

$$a \cdot f1(x, y) + b \cdot f2(x, y) = \mathfrak{F}^{-1}[a \cdot F1(u, v) + b \cdot F2(u, v)] \tag{11.13}$$

where a and b are constants.

Translation The translation property of the Fourier transform shows that if an image is moved (translated), the resulting frequency-domain spectrum undergoes a phase shift, but its amplitude remains the same. Mathematically,

$$\mathfrak{F}[f(x - x_0, y - y_0)] = F(u, v) \cdot \exp[-j2\pi(ux_0/M + vy_0/N)] \tag{11.14}$$

and

$$f(x - x_0, y - y_0) = \mathfrak{F}^{-1}F(u, v) \cdot \exp[j2\pi(ux_0/M + vy_0/N)] \tag{11.15}$$

Conjugate Symmetry If $f(x, y)$ is real, its FT is conjugate symmetric about the origin:

$$F(u, v) = F^*(-u, -v) \tag{11.16}$$

where $F^*(u, v)$ is the conjugate of $F(u, v)$; that is, if $F(u, v) = R(u, v) + jI(u, v)$, then $F^*(u, v) = R(u, v) - jI(u, v)$.

Combining equations (11.10) and (11.16), we have

$$|F(u, v)| = |F(-u, -v)| \tag{11.17}$$

Periodicity The FT (and its inverse) are infinitely periodic in both the u and v directions. Mathematically,

$$F(u, v) = F(u + M, v + N) \tag{11.18}$$

and

$$f(x, y) = f(x + M, y + N) \tag{11.19}$$

Separability The Fourier transform is separable; that is, the FT of a 2D image can be computed by two passes of the 1D FT algorithm, one along the rows (columns), and the other along the columns (rows) of the result.

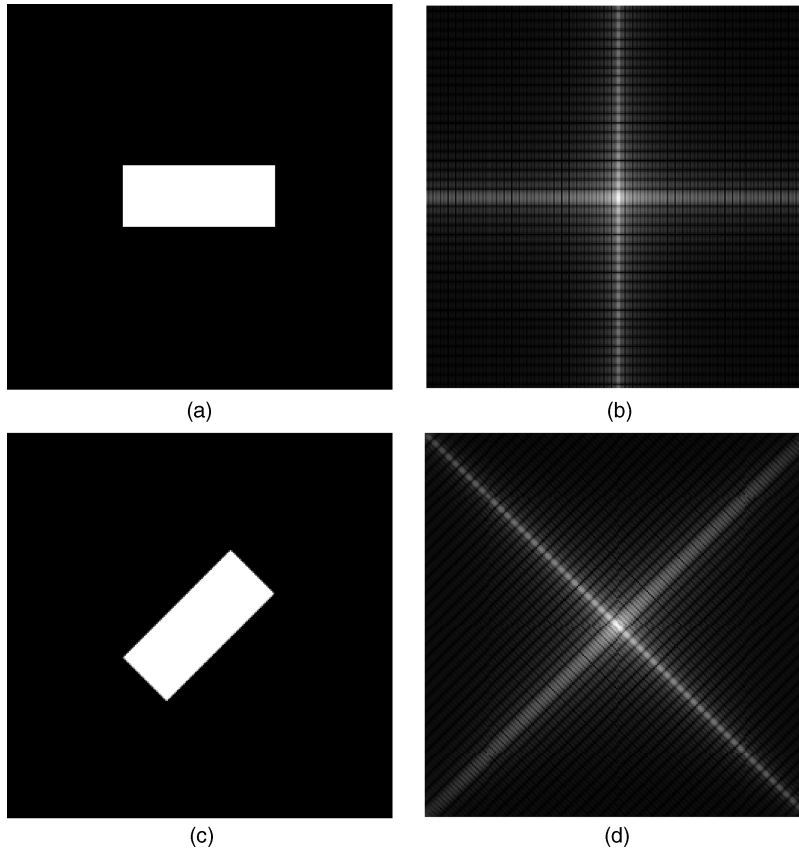
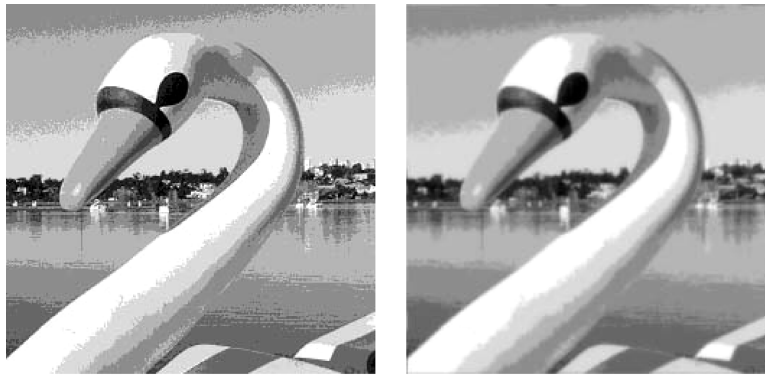


FIGURE 11.5 Original image (a) and its 2D FT spectrum (b); rotated image (c) and its 2D FT spectrum (d).

Rotation If an image is rotated by a certain angle θ, its 2D FT will be rotated by the same angle (Figure 11.5).

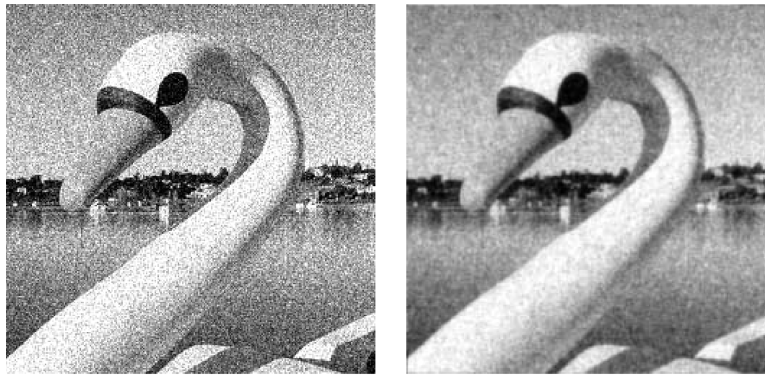
11.2.4 Other Mathematical Transforms

In addition to the Fourier transform, there are many other mathematical transforms used in image processing and analysis. Some of those transforms will be described later in the book, whenever needed (e.g., the discrete cosine transform (DCT) in Chapter 17), while many others (e.g., sine, Hartley, Walsh, Hadamard, wavelet, and slant, to mention but a few) will not be discussed in this text. Refer to the Section "Learn More About It" at the end of the chapter for useful pointers and references.



(a) (b)

FIGURE 11.6 Example of using LPF to smooth false contours: (a) original image; (b) result of applying a LPF.



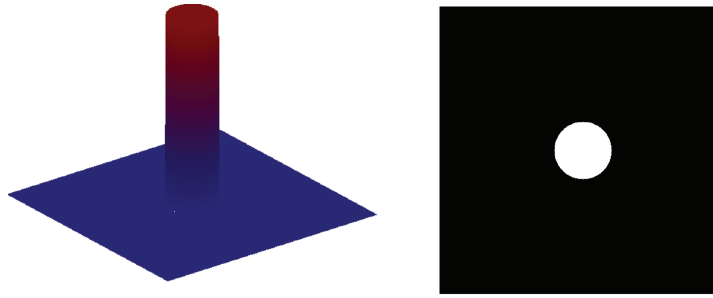
(a) (b)

FIGURE 11.7 Example of using LPF for noise reduction: (a) original image; (b) result of applying a LPF.

11.3 LOW-PASS FILTERING

Low-pass filters attenuate the high-frequency components of the Fourier transform of an image, while leaving the low-frequency components unchanged. The typical overall effect of applying a low-pass filter (LPF) to an image is a controlled degree of blurring. Figures 11.6 and 11.7 show examples of applications of LPFs for smoothing of false contours (Section 5.4.3) and noise reduction,[4] respectively.

[4]We shall discuss noise reduction in more detail in Chapter 12.



(a) (b)

FIGURE 11.8 Frequency response plot for an ideal LPF: (a) 3D view; (b) 2D view from the top.

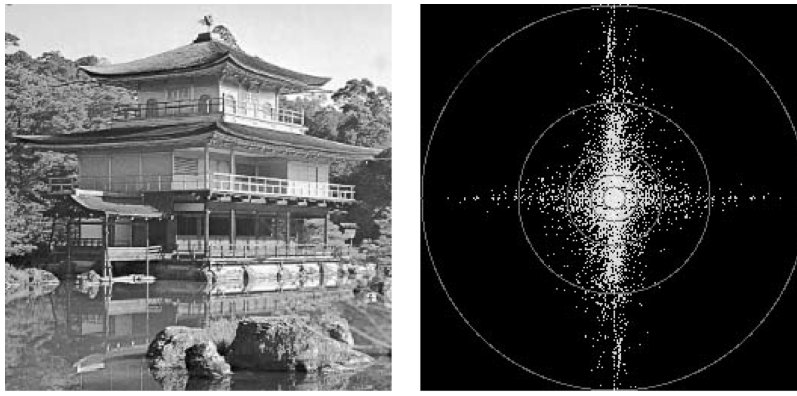
11.3.1 Ideal LPF

An ideal low-pass filter enhances all frequency components within a specified radius (from the center of the FT), while attenuating all others. Its mathematical formulation is given as follows:

$$H_I(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_0 \\ 0 & \text{if } D(u, v) > D_0 \end{cases} \tag{11.20}$$

where $D(u, v) = \sqrt{(u^2 + v^2)}$ represents the distance between a point of coordinates (u, v) and the origin of the 2D frequency plan, and D_0 is a nonnegative value, referred to as the *cutoff frequency* (or *cutoff radius*).

Figure 11.8 shows the frequency response plot for an ideal LPF. Figure 11.9 shows an example image and its Fourier spectrum. The rings in Figure 11.9b represent different values for cutoff frequencies (D_0): 8, 16, 32, 64, and 128.



(a) (b)

FIGURE 11.9 (a) Original image (256 × 256 pixels); (b) Fourier spectrum of the image in (a). The rings represent cutoff frequencies for the low-pass filter examples described later.

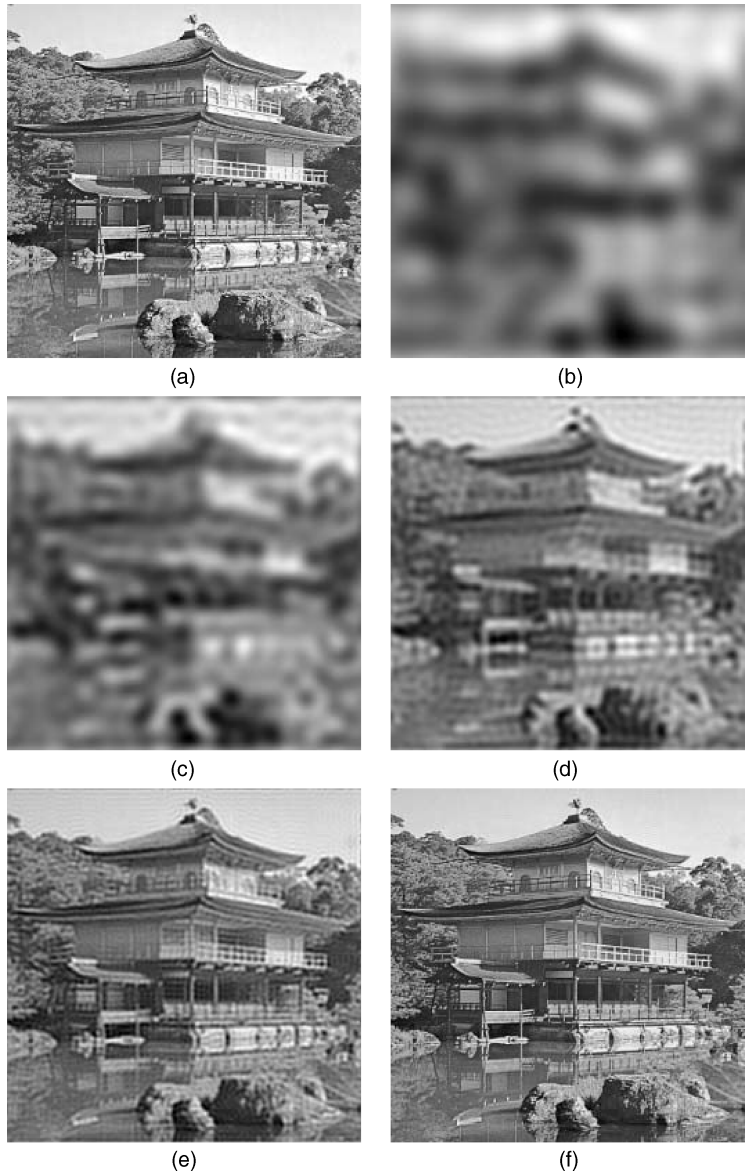
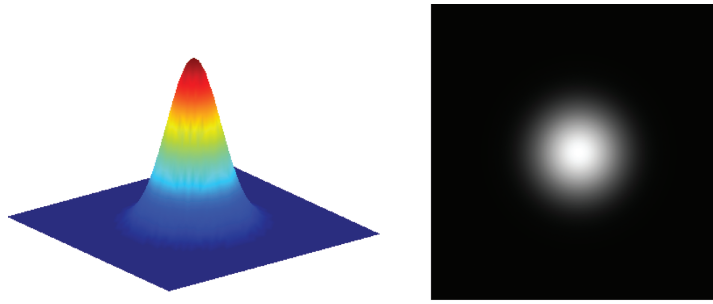


FIGURE 11.10 (a) Original image (256 × 256 pixels); (b–f) ideal LPF results for filters with cutoff frequency corresponding to the radii in Figure 11.9b, namely, 8, 16, 32, 64, and 128 pixels.



(a) (b)

FIGURE 11.11 Frequency response plot for a Gaussian LPF: (a) 3D view; (b) 2D view from the top.

Figure 11.10 shows the results of applying ideal low-pass filters with different cutoff frequencies to the original image: lower values of D_0 correspond to blurrier results. A close inspection of Figure 11.10 shows that the filtered images not only are blurry versions of the input image—an expected outcome, common to all low-pass filters—but also exhibits noticeable *ringing* artifacts that appear because of the sharp transition between passband and stopband in ideal (low-pass) filters.[5]

11.3.2 Gaussian LPF

A Gaussian low-pass filter attenuates high frequencies using a transfer function whose shape is based on a Gaussian curve. The width of the bell-shaped curve can be controlled by specifying the parameter σ, which is functionally equivalent to the cutoff frequency D_0 defined previously: lower values of σ correspond to more strict filtering, resulting in blurrier results. The smooth transition between passband and stopband of the Gaussian LPF guarantees that there will be no noticeable ringing artifacts in the output image. The Gaussian LPF can be mathematically described as

$$H_G(u, v) = e^{-(D(u,v)^2)/2\sigma^2} \tag{11.21}$$

Figure 11.11 shows the frequency response plot for a Gaussian LPF.

Figure 11.12 shows the results of applying Gaussian low-pass filters with different values of σ.

11.3.3 Butterworth LPF

The Butterworth family of filters—widely used in 1D analog signal processing—provides an alternative filtering strategy whose behavior is a function of the cutoff

[5]The fact that the "ideal" LPF produces less than perfect results might seem ironic to the reader; in the context of this discussion, "ideal" refers to abrupt transition between passband and stopband.

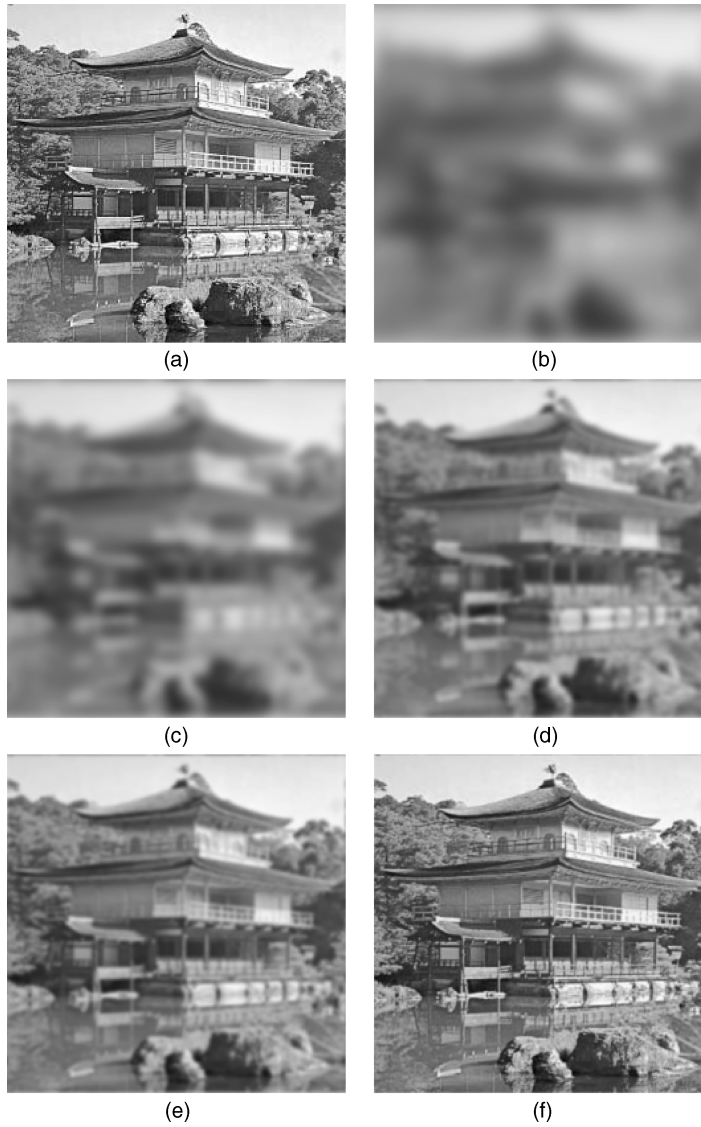


FIGURE 11.12 (a) Original image (256 × 256 pixels); (b–f) Gaussian LPF results for filters with different values for σ: 5, 10, 20, 30, and 75.

frequency, D_0, and the order of the filter, n. The shape of the filter's frequency response, particularly the steepness of the transition between passband and stopband, is also controlled by the value of n: higher values of n correspond to steeper transitions, approaching the ideal filter behavior.

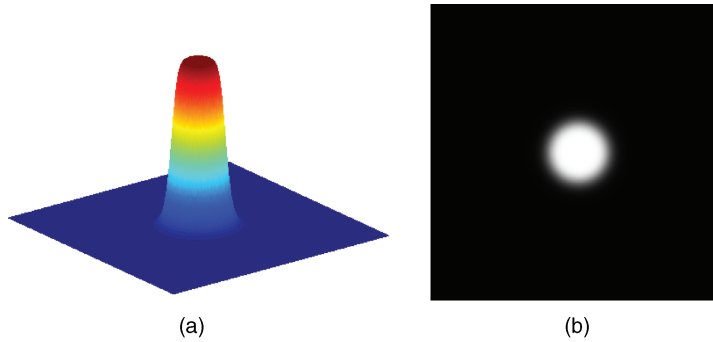


FIGURE 11.13 Frequency response plot for a Butterworth LPF of order $n = 4$: (a) 3D view; (b) 2D view from the top.

The Butterworth filter can be mathematically described as

$$H_{\mathrm{B}}(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}} \tag{11.22}$$

where D_0 is the *cutoff frequency* and n is the *order* of the filter. Figure 11.13 shows the frequency response plot for a Butterworth LPF of order $n = 4$.

Figure 11.14 shows the results of applying Butterworth low-pass filters with $n = 4$ and different cutoff frequencies to the original image. Comparing these results with the ones observed for the ideal LPF (Figure 11.10) for the same cutoff frequency, we can see that they show a comparable amount of blurring, but no ringing effect.

11.4 HIGH-PASS FILTERING

High-pass filters attenuate the low-frequency components of the Fourier transform of an image, while enhancing the high-frequency components (or leaving them unchanged). The typical overall effect of applying a high-pass filter to an image is a controlled degree of sharpening.

11.4.1 Ideal HPF

An ideal high-pass filter attenuates all frequency components within a specified radius (from the center of the FT), while enhancing all others. Its mathematical formulation is given as follows:

$$H_I(u, v) = \begin{cases} 0 & \text{if } D(u, v) \leq D_0 \\ 1 & \text{if } D(u, v) > D_0 \end{cases} \tag{11.23}$$

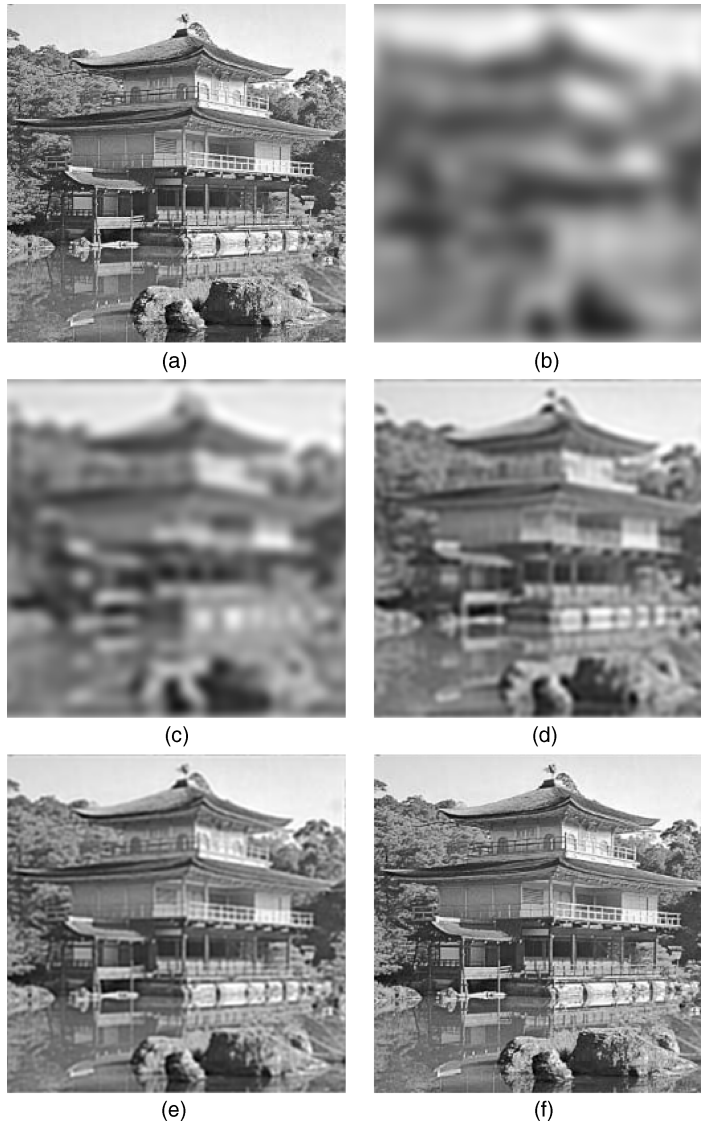


FIGURE 11.14 (a) Original image (512 × 512 pixels); (b–f) fourth-order Butterworth LPF results for filters with cutoff frequency corresponding to the radii in Figure 11.9b, namely, 8, 16, 32, 64, and 128 pixels.

where $D(u, v) = \sqrt{(u^2 + v^2)}$ represents the distance between a point of coordinates (u, v) and the origin of the 2D frequency plan, and D_0 is a nonnegative value, referred to as the *cutoff frequency* (or *cutoff radius*).

Figure 11.15 shows the frequency response plot for an ideal HPF.

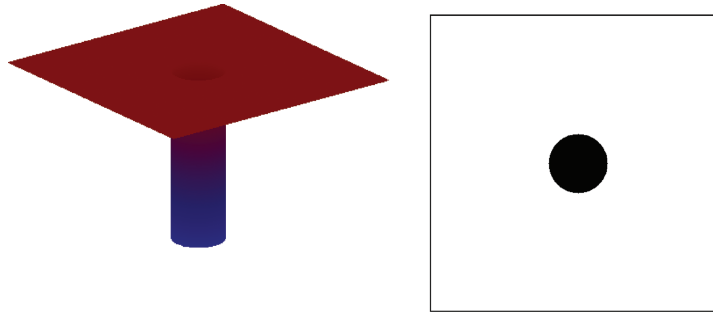


FIGURE 11.15 Frequency response plot for an ideal HPF: (a) 3D view; (b) 2D view from the top.

11.4.2 Gaussian HPF

A Gaussian high-pass filter attenuates low frequencies using a transfer function whose shape is based on a Gaussian curve. The behavior of the filter can be controlled by specifying the parameter σ, which is functionally equivalent to the cutoff frequency D_0. The Gaussian HPF can be mathematically described as

$$H_{\mathrm{G}}(u, v) = 1 - e^{-(D(u,v)^2)/2\sigma^2} \tag{11.24}$$

Figure 11.16 shows the frequency response plot for a Gaussian HPF.

11.4.3 Butterworth HPF

The Butterworth high-pass filter can be mathematically described as

$$H_{\mathrm{B}}(u, v) = \frac{1}{1 + [D_0/D(u, v)]^{2n}} \tag{11.25}$$

where D_0 is the *cutoff frequency* and n is the *order* of the filter.

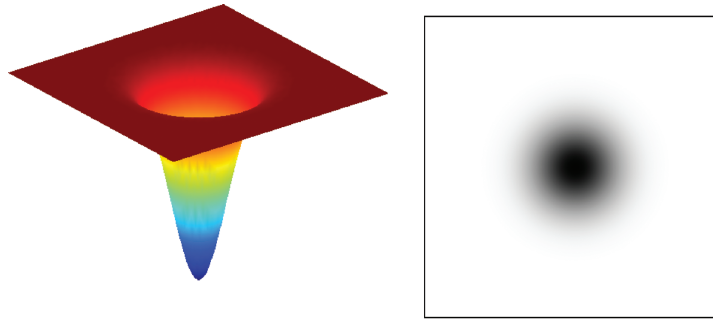


FIGURE 11.16 Frequency response plot for a Gaussian HPF: (a) 3D view; (b) 2D view from the top.

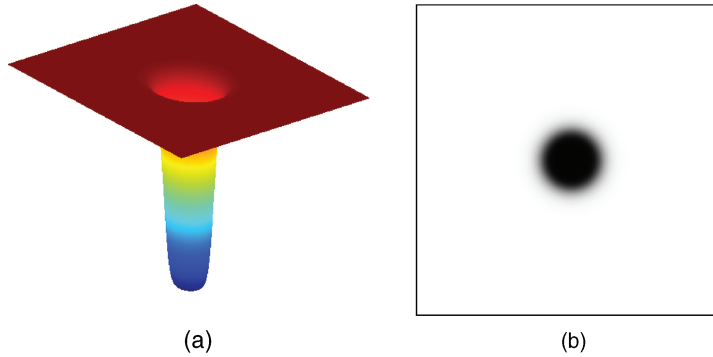


FIGURE 11.17 Frequency response plot for a Butterworth HPF of order $n = 4$: (a) 3D view; (b) 2D view from the top.

Figure 11.17 shows the frequency response plot for a Butterworth HPF of order $n = 4$.

11.4.4 High-Frequency Emphasis

The application of a HPF to an image usually causes a crispening of the high-frequency contents of the image at the expense of the low-frequency contents, which are severely attenuated, leading to a loss of information present in large patches of the original image (its low-frequency components). High-frequency emphasis is a technique that preserves the low-frequency contents of the input image (while enhancing its high-frequency components) by multiplying the high-pass filter function by a constant and adding an offset to the result, that is,

$$H_{\text{hfe}}(u, v) = a + bH(u, v) \tag{11.26}$$

where $H(u, v)$ is the HPF transfer function, $a \geq 0$ and $b > a$. Figure 11.18 shows an example of high-frequency emphasis.

FIGURE 11.18 High-frequency emphasis: (a) input image; (b) result of applying a second-order Butterworth HPF (with $D_0 = 30$) to the input image; (c) result of high-frequency emphasis with $a = 0.5$ and $b = 1$.

11.5 TUTORIAL 11.1: 2D FOURIER TRANSFORM

Goals

The goals of this tutorial are to learn how to compute and display the FT of an image and how to develop filters to be used in the frequency domain.

Objectives

- Learn how to use the `fft2` function to compute the FT of a monochrome image.
- Learn how to visualize the FT results.
- Learn how to generate filters to be used in the frequency domain.

What You Will Need

- `distmatrix.m`

Procedure

To generate the FT of an image (a 2D function), we use the IPT function `fft2`, which implements the FFT algorithm.

1. Load the cameraman image, convert it to `double` (one of the data classes accepted as an input to `fft2`), and generate its FT.

```
I = imread('cameraman.tif');
Id = im2double(I);
ft = fft2(Id);
```

Question 1 What are the minimum and maximum values of the resulting discrete Fourier transform coefficients for the `cameraman` image?

To view the spectrum of the image, that is, the amplitude component of the FT results, we must shift the zero-frequency (DC) component to the center of the image using the `fftshift` function.

2. Shift the FT array of results.

```
ft_shift = fftshift(ft);
```

From your answer to Question 1, you should by now know that the range of values in the FT array of results (`ft`) extends well beyond the typical values of a grayscale image ([0, 255]). Consequently, we will try to display the resulting spectrum as an image using the "scaling for display purposes" option of function `imshow`.

3. Display the FT results, remapped to a grayscale range.

```
figure, subplot(1,2,1), imshow(abs(ft_shift),[]), ...
    title('Direct remap');
```

Question 2 Why are we required to use the `abs` function when displaying the `ft_shift` image?

Question 3 How did we remap the image to a different (narrower) grayscale range?

As you may have noticed, directly remapping the image to the grayscale range does not give us any useful information (only a white pixel at the center of the image—the DC component of the FT results—with all other pixels displayed as black). This suggests that there might be a significant amount of detail in other frequencies that we just cannot see. Recall from Chapter 8 that images with similar attributes (very large dynamic range of gray values) can be brought up by remapping the image with a log transformation.

We can perform the log transformation within the `imshow` function call and then remap the adjusted values to the grayscale range by specifying '[]' in the second parameter.

4. Display the log of the shifted FT image.

```
subplot(1,2,2), imshow(log(1 + abs(ft_shift)), []), ...
    title('Log remap');
```

Question 4 How does the log remap compare to the direct remap?

Distance Matrices

In the second part of this tutorial, we will look at distance matrices. To specify and implement frequency-domain filters, we must first generate a matrix that represents the distance of each pixel from the center of the image. We can create such matrix using the `distmatrix` function. The function takes two parameters, `M` and `N`, and will return the distance matrix of size `M` × `N`.

5. Close any open figures.
6. Generate a distance matrix that is the same size as the image `I`.

```
[M, N] = size(I);
D = distmatrix(M, N);
```

We can visualize the distance matrix in a 3D mesh plot, but we must first shift the image similar to the way we shifted the frequency spectrum earlier.

7. Create a 3D mesh plot of the distance matrix.

```
D_shift = fftshift(D);
figure, mesh(D_shift)
```

Question 5 Explain the shape of the 3D plot.

After obtaining the distance matrix, we can generate the filter of our choice. From that point, filtering in the frequency domain is as simple as multiplying the filter by the FT image and converting back to the spatial domain. These processes will be examined in the remaining tutorials of this chapter.

11.6 TUTORIAL 11.2: LOW-PASS FILTERS IN THE FREQUENCY DOMAIN

Goal

The goal of this tutorial is to demonstrate how to implement low-pass filters in the frequency domain.

Objectives

- Learn how to generate and apply an ideal low-pass filter.
- Learn how to generate and apply a Gaussian low-pass filter.
- Learn how to generate and apply a Butterworth low-pass filter.

What You Will Need

- distmatrix.m
- fddemo.m

Procedure

As we have learned in Tutorial 11.1, the `distmatrix` function returns a two dimensional array, which should be of the same size as the image being processed. The values in the array represent the distance from each pixel to the center of the image. To begin, we will use this matrix to generate an ideal low-pass filter.

1. Load the `eight` image, generate a FT, and display it.

```
I = imread('eight.tif');
Id = im2double(I);
I_dft = fft2(Id);
figure, imshow(Id), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('FT of original image');
```

2. Generate a distance matrix with size equal to the input image.

```
[M, N] = size(I);
dist = distmatrix(M, N);
figure, mesh(fftshift(dist)), title('Distance Matrix');
```

Question 1 Verify that the size of the distance matrix is in fact equal to the size of the image.

Question 2 What happens if we display the distance matrix without shifting?

Ideal LPF

To create an ideal low-pass filter, we will start out with a matrix of all zeros and then set specific values to 1 that will represent all frequencies for which we will allow to pass through. Since we are defining an ideal filter, we can simply define the radius of the filter and then set any values within that radius to 1, while all others remain zero.

3. Create initial filter with all values of zero.

```
H = zeros(M, N);
```

4. Create the ideal filter.

```
radius = 35;
ind = dist <= radius;
H(ind) = 1;
Hd = double(H);
```

Question 3 Explain *how* the previous code sets all values within a given radius to 1.

We can visualize the filter's frequency response by displaying it as an image.

5. Display the filter's frequency response.

```
figure, imshow(fftshift(H)), title('Ideal low-pass filter');
```

To apply the filter to the image, we simply multiply each value of the filter by its corresponding frequency value in the FT image.

6. Apply the filter to the FT image.

```
DFT_filt = Hd .* I_dft;
I2 = real(ifft2(DFT_filt));
```

Question 4 Why do we take only the real values when converting the FT of the filtered image back to the spatial domain?

7. Display both the filtered FT image and the final filtered image.

```
figure, imshow(log(1 + abs(fftshift(DFT_filt))),[]), ...
    title('Filtered FT');
figure, imshow(I2), title('Filtered Image');
```

Question 5 How does the filtered image compare to the original image? Can you see any noticeable artifacts?

To see how the choice of radius affects the filtered image, we can use the *frequency-domain demo*, `fddemo` (developed by Jeremy Jacob and available at the book's companion web site).

8. Load the frequency-domain demo.

```
fddemo
```

The default filter is the ideal low pass. To modify the cutoff value, select the magenta circle within the filter profile and drag it to a desired radius. The value of the radius is displayed below the filter profile.

Question 6 Experiment with different values for the radius of the filter. How does your choice of radius affect the amount of ringing in the output image?

9. Close the demo.

Gaussian LPF

The Gaussian low-pass filter is usually specified by providing a value for the standard deviation σ. We can use the distance matrix previously generated to create a Gaussian filter.

10. Create a Gaussian low-pass filter with $\sigma = 30$.

```
sigma = 30;
H_gau = exp(-(dist .^ 2) / (2 * (sigma ^ 2)));
figure, imshow(Id), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('DFT of original image');
figure, mesh(fftshift(dist)), title('Distance Matrix');
figure, imshow(fftshift(H_gau)), title('Gaussian low-pass');
```

11. Filter the FT image with the Gaussian low-pass filter and display the filtered image.

```
DFT_filt_gau = H_gau .* I_dft;
I3 = real(ifft2(DFT_filt_gau));
figure, imshow(log(1 + abs(fftshift(DFT_filt_gau))),[]), ...
    title('Filtered FT');
figure, imshow(I3), title('Filtered Image');
```

Question 7 Compare the output images between the ideal filter and the Gaussian filter. What are their similarities? What are their differences?

Question 8 Start the frequency-domain demo (`fddemo`) once again, and this time select the Gaussian low-pass filter. Experiment with different values of sigma (standard deviation). How does this value affect the output image?

Butterworth LPF

The Butterworth low-pass filter is usually specified by providing two parameters: the order of the filter, n, and the cutoff value, D_0. In our implementation of the ideal low-pass filter, earlier in this tutorial, we set the cutoff value to 35. For comparison purposes, we will use the same value for our Butterworth filter.

12. Generate a third-order Butterworth filter, where the cutoff value is 35.

```
D0 = 35; n = 3;
H_but = 1 ./ (1 + (dist ./ D0) .^ (2 * n));
figure, imshow(Id), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('FT of original image');
figure, mesh(fftshift(dist)), title('Distance Matrix');
figure, imshow(fftshift(H_but)), title('Butterworth low-pass');
```

13. Filter the image with the Butterworth low-pass filter and display the resulting image.

```
DFT_filt_but = H_but .* I_dft;
I4 = real(ifft2(DFT_filt_but));
figure, imshow(log(1 + abs(fftshift(DFT_filt_but))),[]), ...
    title('Filtered FT');
figure, imshow(I4), title('Filtered Image');
```

Question 9 Compare the ideal-filtered FT image and the Butterworth-filtered FT image.

14. Display all three filters as meshes in 3D and use the *Rotate 3D* option of function `imshow` to explore them in detail.

```
figure, mesh(fftshift(Hd)), title('Ideal low-pass filter');
figure, mesh(fftshift(H_gau)), title('Gaussian low-pass filter');
figure, mesh(fftshift(H_but)), title('Butterworth low-pass filter');
```

Question 10 Implement the Butterworth filter again, but this time using a much higher order, such as 20. How does this output compare to the ideal filter?

Question 11 Experiment with the Butterworth filter by using the frequency-domain demo (`fddemo`). What is the advantage of using this filter over the previous two?

11.7 TUTORIAL 11.3: HIGH-PASS FILTERS IN THE FREQUENCY DOMAIN

Goal

The goal of this tutorial is how to implement high-pass filters in the frequency domain.

Objectives

- Learn how to generate and apply an ideal high-pass filter.
- Learn how to generate and apply a Gaussian high-pass filter.
- Learn how to generate and apply a Butterworth high-pass filter.

What You Will Need

- distmatrix.m
- fddemo.m

Procedure

High-pass filters are conceptually the opposite of low-pass filters and can be implemented in MATLAB using techniques that are very similar to the ones described in Tutorial 11.2. There is, however, a problem when implementing high-pass filters: because a high-pass filter attenuates low frequencies, this means that the zero-frequency term (also known as the DC term) will be set to zero, in turn setting the average value of the image to zero. To compensate for this, we use a technique known as *high-frequency emphasis* filtering, which can be implemented by applying the high-pass filter as we normally would, then multiplying the result (still in the frequency domain) by a constant b, and finally adding an offset constant a.

To begin, let us implement an ideal high-pass filter.

1. Load the `eight` image, generate, and display its FT.

```
I = im2double(imread('eight.tif'));
I_dft = fft2(I);
figure, imshow(I), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('FT of original image');
```

Just as in low-pass filtering, we must generate a distance matrix and use the `fftshift` function when displaying it.

2. Generate a distance matrix based on the size of the input image.

```
[M, N] = size(I);
dist = distmatrix(M, N);
```

Ideal HPF

3. Create the ideal high-pass filter.

```
H = ones(M, N);
radius = 30;
ind = dist <= radius;
H(ind) = 0;
```

Question 1 Explain how the previous code generates an ideal high-pass filter.

We will now apply the high-frequency emphasis filtering technique with `a` = `b` = 1.

4. Apply high-frequency emphasis filtering to the high-pass filter.

```
a = 1;   b = 1;
Hd = double(a + (b .* H));
```

5. Apply the filter to the FT image and display the results.

```
DFT_filt = Hd .* I_dft;
I2 = real(ifft2(DFT_filt));
figure, imshow(log(1 + abs(fftshift(DFT_filt))),[]), ...
    title('Filtered FT');
figure, imshow(I2), title('Filtered Image');
```

6. Display the filter as an image and as a 3D mesh in separate figures.

```
figure, imshow(fftshift(Hd),[]), title('Filter as an image');
figure, mesh(fftshift(Hd)), zlim([0 2]), title('Filter as a mesh');
```

Question 2 When displaying the filter as an image, why must we scale the output for display purposes?

Question 3 How does the filtered image compare to the original image?

We can use the frequency-domain demo (`fddemo`) to experiment with different values for the radius of the filter. Note that to use `fddemo`, the M-file `distmatrix.m` must be in the same directory.

7. Start the frequency-domain demo.

```
fddemo
```

8. From the filter pull-down menu, select *Ideal High Pass*.

Notice that the filter profile shows a magenta circle, indicating the current cutoff value for the filter. The cutoff value is displayed below the profile figure. To change the cutoff value, drag the magenta circle in or out. You will notice that as you drag the circle, the displayed cutoff value changes.

9. Drag the cutoff circle so that the cutoff value is 30 to create the same filtered image as we did above.
10. Change the cutoff value to 10.

Question 4 How does the new image compare to the original image?

Question 5 How does the output compare between using a cutoff value of 30 and 10?

Question 6 What happens to the filtered image as we increase the cutoff value (beyond 30) with respect to the original image?

11. Close any open figures or demos.

Gaussian HPF

We can implement the Gaussian high-pass filter using the existing distance matrix (stored in variable `dist`).

12. Generate a Gaussian high-pass filter.

```
sigma = 30;
H_gau = 1 - exp(-(dist .^ 2) / (2 * (sigma ^ 2)));
```

13. Apply high-frequency emphasis filtering to the high-pass filter and display the filter.

```
H_gau_hfe = a + (b .* H_gau);
figure, mesh(fftshift(H_gau_hfe)), zlim([0 2]), ...
    title('Gaussian high-pass filter');
```

We can now apply the filter to the image.

14. Apply the filter and display the results.

```
DFT_filt_gau = H_gau_hfe .* I_dft;
I3 = real(ifft2(DFT_filt_gau));
figure, imshow(I), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('FT of original image');
figure, imshow(log(1 + abs(fftshift(DFT_filt_gau))),[]), ...
    title('Filtered FT');
figure, imshow(I3), title('Filtered Image');
```

Question 7 How does the Gaussian-filtered image compare to the ideal-filtered image?

We can see the effects of the value chosen for σ (`sigma`) by using the frequency-domain demo.

15. Start the frequency-domain demo.

```
fddemo
```

16. From the filter pull-down menu, select *Gaussian High Pass*.

Notice that the default value for the standard deviation (σ) is 30, so the filtered image should be equal to what we have implemented above. Let us now change this value to see its effect on both the filter and the resulting image.

17. Change the standard deviation to 10.

Question 8 What happened to the filter?

Question 9 How was the filtered image affected?

Question 10 In general, how does the filter change when the standard deviation of the filter is increased or decreased?

18. Close any open figures or demos.

Butterworth HPF

To implement the Butterworth high-pass filter, we can again use the existing distance matrix (stored in variable `dist`).

19. Generate a Butterworth high-pass filter.

```
cutoff = 30; order = 2;
H_but = 1 ./ (1 + (cutoff ./ dist) .^ (2 * order));
```

20. Apply high-frequency emphasis filtering to the high-pass filter and display the resulting filter.

```
H_but_hfe = a + (b .* H_but);
figure, mesh(fftshift(H_but_hfe)), zlim([0 2]), ...
    title('Butterworth high-pass filter');
```

21. Apply the filter to the input image and display the results.

```
DFT_filt_but = H_but_hfe .* I_dft;
I4 = real(ifft2(DFT_filt_but));
figure, imshow(I), title('Original Image');
figure, imshow(log(1 + abs(fftshift(I_dft))),[]), ...
    title('FT of original image');
figure, imshow(log(1 + abs(fftshift(DFT_filt_but))),[]), ...
    title('Filtered FT');
figure, imshow(I4), title('Filtered Image');
```

In the last portion of this tutorial, we will further explore Butterworth high-pass filters using the `fddemo`.

22. Load `fddemo`.

```
fddemo
```

23. From the filter pull-down menu, select *Butterworth High Pass*.

This filter has two parameters: *cutoff value* and *order*. The cutoff value can be adjusted simply by dragging the magenta circle. To change the order of the filter, type a new value and press *Update*.

Question 11 How does the order parameter change the shape of the filter?

Question 12 For large order values (such as 10), the Butterworth begins to take the shape of what other high-pass filters take?

WHAT HAVE WE LEARNED?

- The *Fourier transform* is the main mathematical tool used to obtain the 2D frequency contents of a digital image. The resulting image is said to be in *Fourier space* or *Fourier domain*. The 2D Fourier transform is implemented in MATLAB by function `fft2` and its results are displayed with function `fftshow`.
- The design of an image processing filter in the frequency domain involves the following steps: (1) determining the desired behavior (low pass, high pass, etc.), (2) choosing the type of filter (ideal, Gaussian, Butterworth, etc.), and (3) specifying the parameters associated with the chosen filter type (e.g, cutoff frequency for ideal filters, filter order for Butterworth filters, etc.).
- Low-pass filters attenuate the high-frequency components of the Fourier transform of an image, while leaving the low-frequency components unchanged. The typical overall effect of applying a low-pass filter to an image is a controlled degree of blurring.
- High-pass filters attenuate the low-frequency components of the Fourier transform of an image, while enhancing the high-frequency components (or leaving them unchanged). The typical overall effect of applying a high-pass filter to an image is a controlled degree of sharpening.
- Ideal (low-pass and high-pass) filters have a sharp transition from passband to stopband, which can lead to undesirable image artifacts, most noticeably ringing. Gaussian filters, on the other hand, show a smooth transition from passband to stopband. Butterworth filters allow the designer to choose their order—which impacts the shape of the transition between passband and stopband—and control how close to an ideal filter they should behave.

LEARN MORE ABOUT IT

- There are entire books devoted to the Fourier transform (e.g., [Pap62]) and its fast implementation (e.g., [Bri74]).
- Recommended references for one- and multidimensional signal analysis and Fourier transform are Chapters 13 and 14 of [BB08], Chapters 2–6 of [vdEV89], Chapters 4 and 5 of [OWY83], and Chapters 1 and 3 of [Lim90], among many others.
- Section 5.1 of [Umb05] provides useful examples of the relationships between mathematical transforms, transform coefficients, and basis images.
- Section 9.3 of [Pra07] discusses computational aspects of 2D FT calculations using the FFT implementation.
- Chapter 2.3 of [Bov00a] provides a friendly and informative explanation of the Fourier transform and image processing in the frequency domain.
- [Pra07] is a good source of additional information for mathematical transforms not covered in this book, for example, the cosine, sine, and Hartley transforms

(Section 8.3), the Hadamard, Haar, and Daubechies (a class of wavelets) transforms (Section 8.4), and the Kahunen–Loeve transform (KLT) (Section 8.5).

- Chapter 5 of [Umb05] also covers mathematical transforms not included in this chapter, such as cosine, Walsh–Hadamard, wavelet, Haar, and the principal components transform (PCT).
- A homomorphic filter is a special type of frequency-domain filter that can be particularly useful in situations where the input image shows significant variations in illumination. Homomorphic filtering is discussed in Section 4.9.6 of [GW08] and Section 7.9 of [McA04], among many other references.
- Other types of filters with image processing applications include bandpass, bandreject, and notch filters. These selective filters will be discussed in Section 12.4 in the context of noise reduction.
- The discussion on the computational cost of spatial-domain filtering versus frequency-domain filtering (and other related implementation implications) is beyond the scope of this book. The interested reader may consult Chapter 7 of [SOS00].

11.8 PROBLEMS

11.1 Write a MATLAB script to generate a 256×256 test image consisting of a white circle against a black background, use this image as an input to the `fft2` function, and display the resulting spectrum. You should be able to notice the presence of "ringing" artifacts in the resulting spectrum, owing to the sharp transitions between the circle and the background.

11.2 Implement the solution to the ringing artifact in Problem 11.1, while keeping the input image similar to the original (white circle against a black background) as much as possible.

11.3 In this problem,

(a) Design a spatial-domain averaging filter whose output is the average of the four neighbors of the center pixel in a 3×3 neighborhood.

(b) Use the MATLAB function `freqz2` to obtain its frequency-domain equivalent and plot the resulting filter function.

(c) Apply the two filters, one at a time, to an input image of your choice and observe the results. Are there any noticeable differences? Explain.