3.2 Interferogram Processing Using Frequency Techniques

3.2.1 Demodulating simulated fringes due to a tilt

This section simulates a typical interference pattern and deciphers it using the frequency technique. Consider the following test interferogram consisting of parallel interference fringes with an inclination angle of θ, whose intensity is given by

$$I(x, y) = a + b \sin[k_0(x \cos \theta + y \sin \theta)], \tag{3.1}$$

where a and b are constants. These fringes represent a carrier frequency or an impulse in the frequency domain. In an optical interferometer, the fringes represent a tilt between the reference and the object beam.

The MATLAB script "my_tilt.m," listed in Table 3.1, deciphers the fringes and produces a tilt. Figures 3.1(a) and 3.1(b) show the fringes and the deciphered fringe pattern. In this example, the information is the tilt, so the algorithm works by cropping only where the carrier is and performing an inverse Fourier transform. The general case in which the carrier must be eliminated is discussed in Section 3.2.2.

3.2.2 Demodulating fringes embedded with a carrier

The measured interferometric intensity distribution $I(x, y)$ can be written as

$$I(x, y) = a(x, y) + b(x, y) \cos[(k_{x_0}x + k_{y_0}y) + \phi(x, y)], \tag{3.2}$$

where k_{x_0, y_0} are the carrier spatial frequencies, $a(x, y)$ is the background variation, and $b(x, y)$ is related to the local contrast of the pattern. In other words, $a(x, y)$ carries the additive and $b(x, y)$ carries the multiplicative disturbances, respectively, and $\phi(x, y)$ is the interference phase to be computed from $I(x, y)$. Equation (3.2) can be written as [1,2]

$$\begin{aligned} I(x, y) = a(x, y) &+ b_c(x, y) \exp[j(k_{x_0}x + k_{y_0}y)] \\ &+ b_c^*(x, y) \exp[-j(k_{x_0}x + k_{y_0}y)], \end{aligned} \tag{3.3}$$

where $b_c(x, y) = (b(x, y)/2) \exp[j\phi(x, y)]$. The Fourier transform of Eq. (3.3) produces

$$\tilde{I}(k_x, k_y) = \tilde{a}(k_x, k_y) + \tilde{b}_c(k_x - k_{x_0}, k_y - k_{y_0}) + \tilde{b}_c^*(k_x + k_{x_0}, k_y + k_{y_0}), \tag{3.4}$$

where $\sim$ indicates the function in the Fourier domain. Assuming that the background intensity $a(x, y)$ is slowly varying compared to the fringe spacing, the amplitude spectrum $\tilde{I}(k_x, k_y)$ will be a trimodal function with $\tilde{a}(k_x, k_y)$ broadening the zero peak and $\tilde{b}_c$, $\tilde{b}_c^*$ placed symmetrically with respect to the

Table 3.1 MATLAB code "my_tilt.m" creates a fringe pattern [see Figs. 3.1(a) and (b)].

```
%%This section starts with a computer-generated fringe
%and
%tries to decipher the unwrapped phase
clc
close all
clear all
lambda=0.632;                          %In microns
%- - - - - - - - - - - - -Create Test Image- - - - - - - - - - - - - -
pts=2^8;
x=linspace(0,pts/8-1,pts);
y=x;
[X0,Y0]=meshgrid(x,y);
theta=45*pi/180;
f0=1;
I =128+127*. . .
sin(2*pi*(X0*f0*cos(theta)+Y0*f0*sin(theta)));
figure
imagesc(I)
colormap(gray(256))
colorbar
title('Simulated fringe pattern')
%Crop to make it square
[rows,cols]=size(I);
if cols>rows
rect_crop=[floor(cols/2)-floor(rows/2) 1 rows-1 rows];
elseif cols<rows
rect_crop=[1 floor(rows/2)-floor(cols/2) cols cols-1];
else
rect_crop=[1 1 cols rows];
end
I=imcrop(I,rect_crop);
im(I)
axis xy
max(I(:))
min(I(:))
%Go to Fourier domain to select the region of interest
I1F=(fft2(fftshift(I)));
I1F_s=fftshift(I1F);
abs_I=abs(I1F_s).^2;
I1F_c=zeros(pts,pts);
%%
mesh(abs_I)
for m=1:1:1
im(abs_I)
axis xy
[Ro,Co]=size(I1F);
line([ceil(Co/2)+1 ceil(Co/2)+1],[0 ceil(Ro)])
line([0 ceil(Co)],[ceil(Ro/2)+1 ceil(Ro/2)+1])
rect1=[151-5 151-5 10 10];
[I1,rect]=imcrop(abs_I,rect1);
I1F_c(round(rect(2)):round(rect(2)+rect(4)),. . .
round(rect(1)):round(rect(1)+rect(3)))=. . .
I1F_s(round(rect(2)):round(rect(2)+rect(4)),. . .
round(rect(1)):round(rect(1)+rect(3)));
end
```

(*continued*)

Table 3.1 (*Continued*)

```
abs_I_c=abs(I1F_c).^2;
im(abs_I_c)
[Ro,Co]=size(abs_I_c);
line([ceil(Co/2)+1 ceil(Co/2)+1],[0 ceil(Ro)])
line([0 ceil(Co)],[ceil(Ro/2)+1 ceil(Ro/2)+1])
axis xy
%%
%Calculate the wrapped phase
I2=fftshift(ifft2(fftshift(I1F_c)));
[ro,co]=size(I2);
I3=I2(:);
ind=find(real(I3)>0);
I3_phase=zeros(1,length(I3))';
I3_phase(ind)=atan(imag(I3(ind))./real(I3(ind)));
ind=find(real(I3)<=0);
I3_phase(ind)=atan(imag(I3(ind))./...
real(I3(ind)))+pi*sign(imag(I3(ind)));
I2_phase=reshape(I3_phase,ro,co);
%I2_phase=atan2(imag(I2),real(I2));
close all
figure
imshow(I2_phase)
max(I2_phase(:))
min(I2_phase(:))
%%%- - -Unwrapping using the 2D_SRNCP algorithm:
%http://www.ljmu.ac.uk/GERI/90225.htm
%Call the 2D phase unwrapper from C language.
%To compile the C code: in MATLAB command window, type
%Miguel_2D_unwrapper.cpp. The file has to be in the same
%directory
%as the script to work.
WrappedPhase=I2_phase;
mex Miguel_2D_unwrapper.cpp
%The wrapped phase should have the single (float in C)
%data type
WrappedPhase = single(WrappedPhase);
UnwrappedPhase = Miguel_2D_unwrapper(WrappedPhase);
UnwrappedPhase=double(UnwrappedPhase);
h = fspecial('average',15);
eta1 = imfilter(UnwrappedPhase,h);
eta1=eta1(10:end-10,10:end-10);
figure;
surfl(eta1/(4*pi)*lambda);shading interp;
colormap(gray);
 title('Unwrapped 3D Depth Map');
 zlabel('Depth in microns')
```

origin. The next step is to eliminate the zero-frequency term and one of the sidebands $\tilde{b}_c$, $\tilde{b}_c^*$. The new spectrum is no longer symmetric, and the space-domain function is no longer real but complex. Therefore, Eq. (3.4) becomes

$$\tilde{I}'(k_x, k_y) = \tilde{b}_c(k_x - k_{x_0}, k_y - k_{y_0}), \tag{3.5}$$

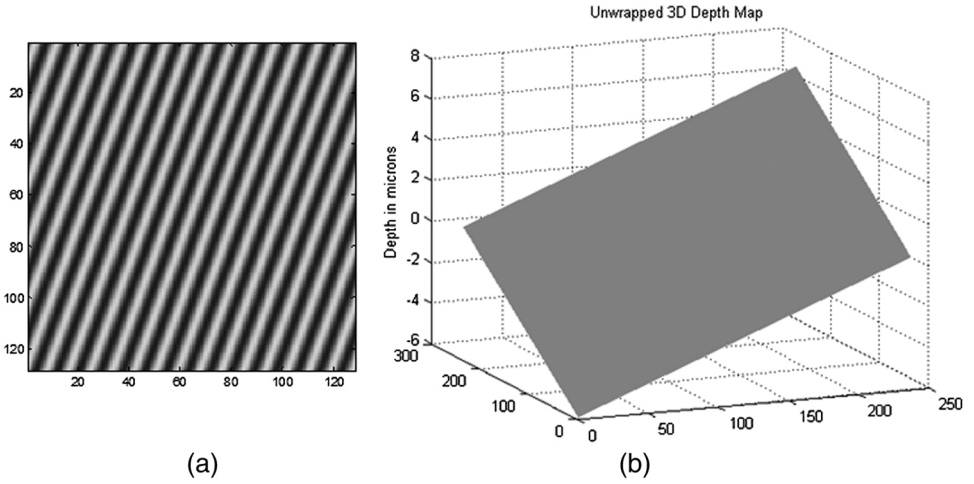

Figure 3.1 (a) Parallel interference fringes with an angle θ and (b) deciphered fringes showing relative tilt between two beams.

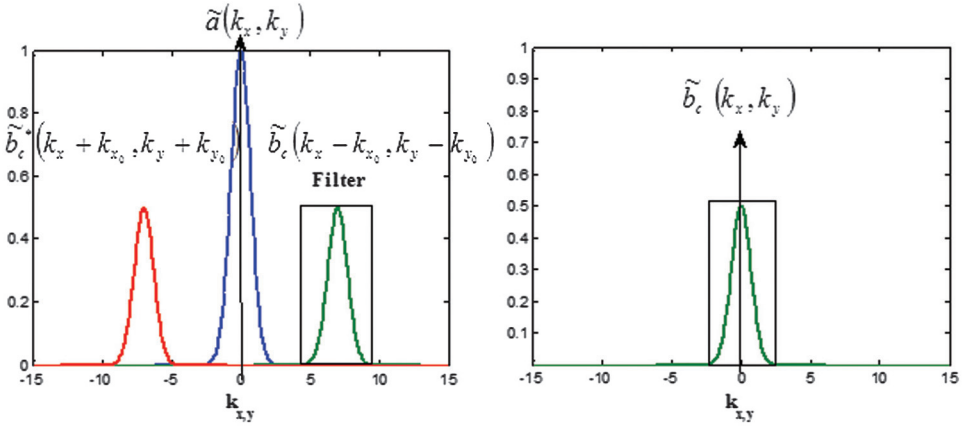

Figure 3.2 Schematic of the process.

which is the new filtered spectrum centered around the zero frequency. Then the inverse Fourier transform is performed on Eq. (3.5) to get

$$I'(x,y) = b_c(x,y) = \frac{1}{2} b(x,y) \exp[j\phi(x,y)]. \tag{3.6}$$

The following operation is performed to find the wrapped phase:

$$\phi(x,y) = \arctan\left\{\frac{\mathrm{Im}[b_c(x,y)]}{\mathrm{Re}[b_c(x,y)]}\right\}. \tag{3.7}$$

Figure 3.2 illustrates the process. A second example, the MATLAB script "freq_decipher.m," which uses the frequency deciphering technique, is shown in Table 3.2. Figure 3.3(a) shows the initial fringe pattern, Fig. 3.3(b) shows

Table 3.2 MATLAB code "freq_decipher.m" uses the frequency technique to decipher the fringe pattern (see Fig. 3.3).

```
%%This program takes an image that has fringes riding
%on a carrier and tries to get rid of the carrier and
%unwrap the phase
close all
clear all
clc
lambda=0.632;                          %In microns
I=double(imread('Campsp3.tif'));
I=I(:,:,1);
im(I)
axis xy
I = padarray(I,[50 50],0,'both');
im(I)
axis xy
[rows,cols]=size(I);
if cols>rows
rect_crop=[floor(cols/2)-floor(rows/2) 1 rows-1 rows];
elseif cols<rows
rect_crop=[1 floor(rows/2)-floor(cols/2) cols cols-1];
else
rect_crop=[1 1 cols rows];
end
I=imcrop(I,rect_crop);
im(I)
axis xy
max(I(:))
min(I(:))
%Go to Fourier domain to select the region of interest
I1F=(fft2(fftshift(I)));
I1F_s=fftshift(I1F);
abs_I=abs(I1F_s).^2;
[ro,co]=size(I);
im(abs_I)
%%
clc
disp('Drag the circle and center it around the carrier')
disp('which is a white circle centered at 250, 350')
disp('then double click the left button of the mouse')
[Ro,Co]=size(I1F);
line([round(Co/2) round(Co/2)],[0 round(Ro)])
line([0 round(Co)],[ round(Ro/2) round(Ro/2)])
h = imellipse(gca,[10,10,50,50]);
vertices = wait(h);
X=vertices(:,1);
Y=vertices(:,2);
I1F_s_selection=I1F_s(floor(min(Y)):...
floor(max(Y)),floor(min(X)):floor(max(X)));
im(abs(I1F_s_selection).^2)
%Use the Hamming window
if mod(size(I1F_s_selection,1),2)==0
I1F_s_selection=I1F_s_selection(1:end-1,1:end-1);
end
[A, XI, YI]=myhamming2D(size(I1F_s_selection,1))
I1F_c=A.*I1F_s_selection;
```

(*continued*)

Table 3.2 (*Continued*)

```
im(abs(I1F_c).^2)
% I1F_c=I1F_s_selection;
% calculate the wrapped phase
I2=fftshift(ifft2(fftshift(I1F_c)));
im(abs(I2))
[ro,co]=size(I2);
I3=I2(:);
ind=find(real(I3)>0);
I3_phase=zeros(1,length(I3))';
I3_phase(ind)=atan(imag(I3(ind))./real(I3(ind)));
ind=find(real(I3)<=0);
I3_phase(ind)=atan(imag(I3(ind))./real(I3(ind)))+ ...
pi*sign(imag(I3(ind)));
I2_phase=reshape(I3_phase,ro,co);
%Calculate the unwrpped phase on a certain line
figure
imshow(I2_phase)
max(I2_phase(:))
min(I2_phase(:))
%Uncomment if you like to select to unwrap along
%a certain line.
%h=improfile;
%figure
%subplot(2,1,1)
%plot(h)
%unwraph=unwrap(h);
%subplot(2,1,2)
%plot(unwraph)
%clear abs* h v* I I1* I2 I3* A C* R* X* Y* ans c* r*
%i* u*
%%%- - -Unwrapping using the 2D_SRNCP algorithm:
%http://www.ljmu.ac.uk/GERI/90225.htm
%Call the 2D phase unwrapper from C language.
%To compile the C code: in MATLAB command window, type
%Miguel_2D_unwrapper.cpp. The file has to be in the same
%directory as the script to work.
clc
disp('Choose a rectangular area')
disp('using the mouse to unwrap')
figure
imshow(I2_phase)
I2_phase=imcrop;
WrappedPhase=I2_phase;
mex Miguel_2D_unwrapper.cpp
%The wrapped phase should have the single (float in C)
%data type
WrappedPhase = single(WrappedPhase);
UnwrappedPhase = Miguel_2D_unwrapper(WrappedPhase);
UnwrappedPhase=double(UnwrappedPhase);
h = fspecial('average',15);
eta1 = imfilter(UnwrappedPhase,h);
figure;
surfl(eta1/(4*pi)*lambda);shading interp;
colormap(gray);
title('Unwrapped 3D Depth Map');
zlabel('Depth in microns')
```

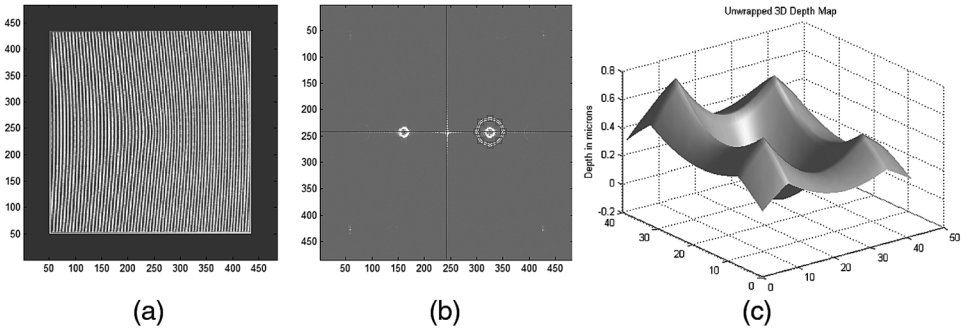

Figure 3.3 (a) The initial fringe pattern, (b) the frequency domain where the region of interest is inside the circle, and (c) the deciphered phase.

the frequency domain where the region of interest is inside a circle, and Fig. 3.3(c) shows the deciphered phase.

3.3 Interferogram Processing Using Fringe Orientation and Fringe Direction

Fringe orientation is often used for fringe processing.[3,4] Knowledge of the fringe orientation is useful in applications such as

(a) contoured window filtering,[5–7]
(b) filtering noise from electronic speckle pattern interferometry (ESPI) fringes,
(c) the contoured correlation method used to generate noise-free fringe patterns for ESPI,[8–10]
(d) corner detection and directional filtering,[11,12] and
(e) phase demodulation based on fringe orientation, where the phase information can be deciphered by computing the fringe orientation using the spiral-phase transform, as shown in Section 3.4.[13–15]

Orientation is related to the local features pertaining to spatial information of an interferogram. Structured and well-patterned features have a specific, well-defined orientation, while noisy regions, without any discernable structure, have no specific orientation.[16] Consequently, by performing the Fourier transform of a small region of a typical interferogram, the spectrum will be concentrated in a small spot oriented at the same angle as the local gradient of that specific region.[17] Thus, the fringe orientation is parallel to the interferogram gradient and is the local spatial-frequency vector orientation in the Fourier domain.[18,19] This section introduces some of the fringe-orientation computation techniques. A summary of the different methods is presented along with their corresponding MATLAB codes. At the end of the section, a comparison table outlines the advantages and