Transactions on Modelling and Simulation vol 5, © 1993 WIT Press, www.witpress.com, ISSN 1743-355X

A C-library for fringe pattern processing

M.L. Luchi, A. Poggialini, S. Rizzuti
Department of Mechanical Engineering, University of Calabria, 87030 Arcavacata di Rende (CS), Italy

ABSTRACT

The paper describes a software library developed in C language for the automated analysis of the fringe patterns which are obtained by the interferometric techniques most frequently employed in experimental mechanics. Since the images acquired are generally affected by high-frequency noise (particularly significant in the case of speckle interferometry), it has been necessary to develop special procedures for determining peak location using the minimum amount of local information.

INTRODUCTION

Experimental mechanics relies on several non-contact optical methods capable of giving full-field information. Information is usually encoded in the form of fringe patterns which represent the contour loci of the quantities detected. Although sometimes the desired data can be directly extracted, more often further processing is needed (e.g. when the quantity which is the object of the investigation is not directly represented by the fringe patterns or whenever a pointwise quantitative analysis is required). A more and more advanced treatment of the data is furthermore demanded by the increasing interaction between theoretical modeling and experimental analysis required by the new hybrid analytical/experimental approaches.

Current availability of advanced instrumentation and computers enables the analyst to record and process a larger amount of information than ever before; digital image processing techniques have hence recently been introduced in experimental mechanics to automate the data reduction process [1]. However, although specific algorithms can be successfully developed for particular classes of problems [2], a general and

automated fringe analysis system, capable of interpreting the experimental data without any user intervention, is still far from being available [3,4,5].

In the present paper a small library of user-callable functions, specifically developed for fringe pattern analysis, is described. The library has been structured in such a way that it can be used to develop either simple application programs with a high degree of man/machine interaction or more complex and automated procedures addressed to specific problems. Procedures are provided for retrieving the experimental data as accurately as possible also when a high noise level is present.

The library consists of several functions, transparent to the user, which operate on a data base, hidden to the user, containing information on the screen coordinates and gray level of the pixels to be processed. The same functions return information, more directly utilizable by the user, in terms of real world coordinates.

The library, devised to be machine independent, has been developed under Unix System V in C language on a general purpose image processing computer system.

LIBRARY STRUCTURE

Figure 1 shows schematically a general overview of the interconnections between the user application program and the fringe processing routines which constitute the library (`fplib.a`). The routines operate on a structured data set of external variables, which remain hidden to the user, and provide information on the quantities of interest to the user on an output file. The output data, which may be further processed if necessary, are given in terms of real world coordinates whereas external variables are defined in screen coordinates to improve efficiency.

Since the library must have access to the video main frame memory to retrieve information on the gray level of the image stored and to the overlay frame memory to plot the results of image processing, it cannot obviously be completely machine independent. However, in order to assure the maximum portability, the device dependent routines have been maintained as much as possible separate from the "fplib". In fact, it has only been necessary to include in the library the call to two device dependent functions, the first of which plots a dot on the screen while the second retrieves the gray level of a picture element of the image. Other device dependent routines are supposed to be accessible by the user program for graphic input (via trackball or similar devices) and graphic output (color plotting of line segments, selective erasing of overlay planes, etc.).

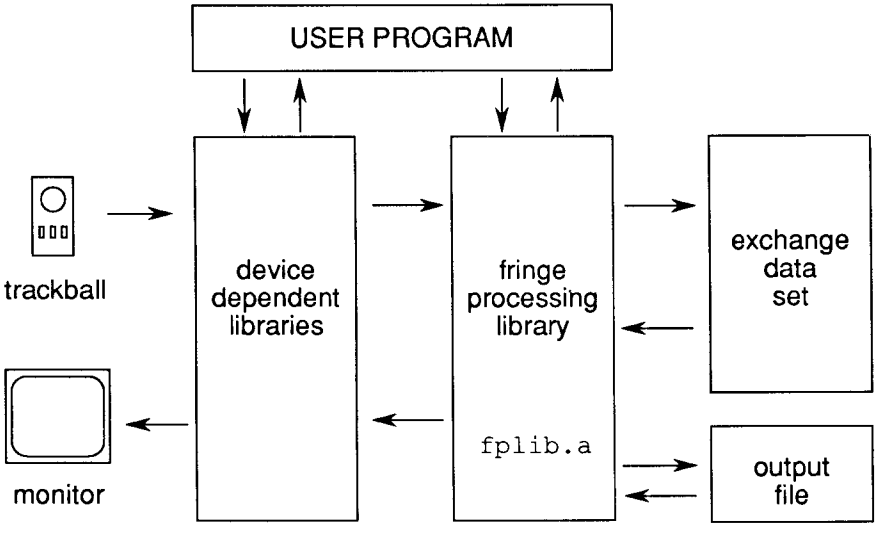

Figure 1. Overview of the interconnections between the user application program and the fringe processing routines.

The software was developed on a general purpose digital image processing computer. An input monochrome video signal (european standard: 625 lines, 50 fields/sec) is digitized in real time according to the CCIR digital coding recommendation by a 8-bit A/D converter operating at 13.5 MHz. The resolution of the digitized image (aspect ratio 4:3) is 702 x 576 pixels (256 gray levels); the pixel aspect ratio differs consequently from unit of about 0.7%. The image is stored in the main frame memory and is displayed on an analogic RGB color monitor by means of three D/A converters. Three 14-bit output look-up tables are available for each r-g-b 8-bit D/A converter. The data stored in the 6 most significant bits of the overlay frame memory (8-bit planes) may be displayed together with the acquired image through the output look-up tables. By individually addressing the bit planes of the overlay frame memory, 6 different sets of graphic information can be overlayed in different colors and selectively erased when no longer necessary.

A basic set of user-callable routines is provided for handling the overlay planes and plotting line segments. Two additional routines are included in the device dependent library which return the pixel coordinates of the end points of a segment or of the vertices of a polygonal line selected by the trackball.

```
sgt_ball(&i1,&j1,&i2,&j2);
sgs_ball(&n,i,j);
```

and a third routine which enables the user to interupt a loop cycle

```
b_ball();.
```

Even if these routines, being machine dependent, are not included in the "fplib", they are reported here since they will be encountered in the test examples described later.

The functions available to the user when the library is linked to an application program can be subdivided into three classes: point-level operating functions; peak-level operating functions; scaling and i/o functions. A fourth class consists of functions which are employed for internal data processing and are not directly accessible to the user.

Storage allocation for the variables used by the functions to exchange data must be provided in the user program by a file inclusion statement

```
#include "myfp.h"
```

Point-level operating functions

All the functions of this class operate on a structured sequence of points. Points are defined in the coordinate system of the screen wherein coordinates are assumed to coincide with the row and column indices of the matrix containing the picture elements (see fig.2). The location of a point must not necessarily coincide with that of a picture element; points may hence belong to any geometric entity and their position will be generally identified by real coordinates. When necessary, the nearest pixel will be used for the representation on the overlay planes.

The term "point" will denote henceforth a more complex structure which will include the components of the unit vector tangent to the line to which the point belongs.

Sequences of points are generated (i.e. coordinates and tangent vector components are stored in the respective arrays in the exchange data set) by functions such as those listed below

`mkpnt(i0,j0,di,dj);` generates a single point at i0, j0 with direction di, dj;

`mksgt(i1,j1,i2,j2);` generates a sequence of points along a segment from i1, j1 to i2, j2;

`mkpts(n,i,j);` generates a sequence of n single points at i[k], j[k] {k=1,n};

`mksgs(n,i,j);` generates a sequence of points along a polygonal line connecting n key-points;

`mkspl(n,i,j);` generates a sequence of points along a cubic spline through n key-points;

Input to these function can be quite simply supplied interactively by the trackball routines.

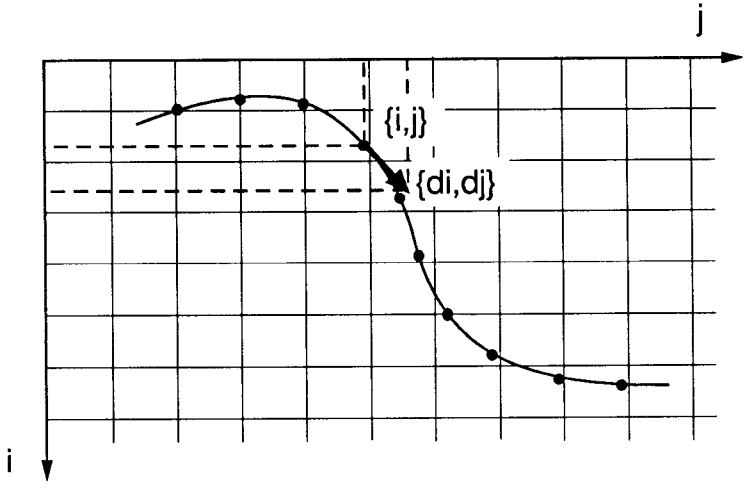

Figure 2. Structured set of points in the screen coordinates system.

Point spacing is made equal to pixel spacing so that each point is mapped, by rounding its coordinates, into a different pixel with the same correspondance as that given by the algorithms generally used in raster graphics representation. For example point coordinates defined by the `mksgt` function, when rounded to the closest integer numbers, define the same pixel locations obtained by the integer arithmetic Bresenham algorithm.

The defined sequence of points can be visualized by the function

`base();` displays the pixels corresponding to the points of the defined sequence;

Once a structured set of points has been defined by the procedures previously described, it is quite straightforward to plot the gray level of a digitized image along any straight or curve segment chosen by the user. On this purpose, two functions can be used to retrieve the gray levels to be plotted:

`rdpoints();` reads the gray levels at the points of the defined sequence (by bilinear interpolation);

`rdpixels();` reads the gray levels at the pixels corresponding to the points of the defined sequence;

the first one defines the gray level at a point by a bilinear interpolation of the gray levels of the four picture elements surrounding the point; the second takes the level to be equal to that of the closest pixel.

The gray levels thus defined can be eventually plotted in the form of a graph or a histogram, using a scale defined by the user, by the two functions:

`graph(scale);` plots a graph of the gray levels along the defined sequence of points;

`histo(scale);` plots a histogram of the gray levels of the defined sequence of points;

Any reference value of the gray level (e.g. the maximum:255) can be displayed by the user using the function

`value(value);` plots a constant value along the defined sequence of points;

Whatever the `mk-`function employed to generate the sequence of points, having previously defined the tangent vector components provides the plotting function with the data necessary to perform its task in an identical and simple way.

The point-level functions described in the present paragraph enable simple utility programs to be built-up for acquiring and processing the light intensity of a digitized image. These same functions can be also used to define manually fringe center lines and to supply the input data to the functions for automatic peak-searching which will be described in the following paragraph.

Peak-level operating functions

Once a structured sequence of points has been defined with the aid of the point-level operating functions, the user may access the higher level peak optimization and searching functions. The term "peak" is used here and in the following to indicate the location of a local minimum (or maximum) of brightness. A peak will belong to a bright or dark fringe according to whether it corresponds to a maximum or a minimum; a fringe will be hence treated as a sequence of peaks, where a peak, analogously to a "point", is defined not only by its screen coordinates but also by the components of the unit vector tangent to the fringe center line. Peak definition includes an additional information, that is distance between fringes measured orthogonally to the fringe center line. An estimate of fringe spacing will in fact be useful to the peak optimization functions in searching the peak location.

All the information specified above is stored in the exchange data set in array form. Five different arrays contain the sequence of the peak coordinates, the correspondent components of the unit vector defining the tangent direction and fringe spacing; each row of the arrays refers to a same fringe (or portion of fringe). Fringe values, defined in units chosen by the user or simply coincident with fringe orders, are stored in a separate array whose elements are associated with the rows of the peak arrays.

Functions pertaining to this section of the library can be used to develop different procedures with different level of automation and/or accuracy.

For example, the cumbersome and inaccurate procedure consisting in tracing manually the fringe center lines can be made less time consuming and much more precise. A structured point sequence can in fact be generated by linear (`mksgs` function) or cubic (`mkspl`) interpolation through a few key-points located, via the trackball, on a fringe center line visually estimated by the user; the user is also required to assign interactively the fringe value. All the information concerned to the points is hence transferred to the proper row of the peak arrays by the function

`addpeaks(fringe);` converts a sequence of points into a sequence of peaks which is appended to the fringe of index <fringe>;

The number of peaks thus defined will coincide with the number of points in the sequence. Recalling the criteria followed in generating peak sequences, point density will result the same as pixel density, that is the density which enables all the information actually available to be exploited in the subsequent peak optimization process. The first manual approximation will in fact be automatically refined using the peak optimization routines described later.

A procedure with a much higher degree of automation has been developed which involves "cutting" a set of fringes with a line (or more than one, if necessary) approximately orthogonal to the fringes. The line is defined manually by the user, using once again the trackball, who also identifies approximately the points where the line intersects the fringe center lines and inputs for each of them the corresponding fringe order. A point sequence is thus generated (`mkpts`) which includes the fringe centers only. Although, at the present stage of development, this first step of the procedure must be accomplished manually, it is susceptible of being completely automated where the location of the fringe centers along the "cutting" line is concerned, and partially automated in the assignement of fringe values; in this latter respect the user intervention may in fact be limited to defining the first fringe value and its increment between two consecutive fringes. The operator is free to choose whether to work on dark or bright fringes, or on both.

Point information is transferred by the function

`addfringes();` converts each point of the sequence into a peak pertaining to a different fringe;

to the first column (starting from the first free row) of the peak arrays; note that the unit tangent vector must be rotated counterclockwise of 90° to be approximately aligned with the tangent to the fringe center line

while fringe spacing can be automatically calculated from the distance between the intersection points previously identified.

Starting from each intersection point, each fringe can be now automatically traced. After the location of the peak has been more accurately determined by the peak optimization functions, a second peak is searched advancing along the tangent direction of an increment defined by the user. An appropriate function

`gopeak(fringe,increment);` adds a peak to the fringe of index <fringe> moving along the unit tangent vector of an increment of length <increment>;

is provided, whereby a new element is added to the row of the peak arrays which corresponds to the fringe being processed. The same function performs the additional task of checking whether the screen or object boundary (or the starting point in the case of closed loop fringes) has been reached. Once the new peak location has been optimized, the unit tangent vector is updated on the basis of the previous peak coordinates to be kept as much as possible aligned with the actual tangent to the fringe center line. The function

`udpeak(fringe);` updates the unit tangent vector of the last peak of the fringe of index <fringe>;

is used on this purpose.

All the procedures presently developed for locating automatically the peak on the top of the ridge (or on the bottom of the valley) of light intensity, involve searching the peak on a point sequence distributed along a segment normal to the tangent to the fringe center line, whose length is related to fringe spacing. The simplest approach consists in searching along the point sequence the minimum (or maximum) of the gray levels acquired. This task is performed by

`ominpeak(fringe,peak);` relocates the peak of index <peak> pertaining to the fringe of index <fringe> on the point with the lowest gray level detected along the normal to the fringe center line;

or by the analogous `omaxpeak`.

A more sophisticated approach, which enables a sub-pixel accuracy to be achieved, can be optionally adopted. The procedure makes use of the function

`o2peak(fringe,peak);` optimizes peak location by 2-nd order polynomial fitting;

and identifies the peak as the minimum (or maximum) of the second order polynomial which best fits the gray level values on a segment equal to the half of fringe spacing. It must be pointed out that this second approach must necessarily substitutes the first when noise is present.

The procedure has proved effective for fringes which are approximately equispaced; when, on the contrary, fringe spacing varies sensibly running along a fringe, a different procedure should be employed which enables fringe spacing to be continously updated. The function

`o4peak(fringe,peak);` optimizes peak location by 4-th order polynomial fitting;

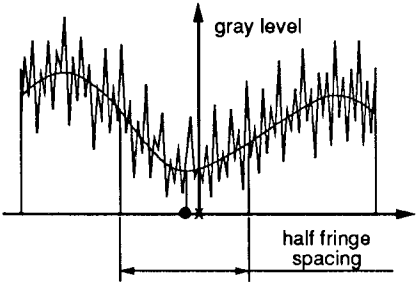

Figure 3. 4-th order polynomial fitting of the gray levels.

has been developed whereby a 4-th order polynomial is adopted to fit the gray level values on a point sequence of length equal to fringe spacing (see fig.3); fringe spacing can thus be estimated as the double of the distance between the points where the second derivative of the best fitting polynomial is zero.

Other functions pertaining to this section of the library are utility functions necessary for developing application programs

`nfringes();` returns the total number of fringes;
`lastfringe();` returns the index of the last fringe;
`npeaks(fringe);` returns the total number of peaks defined on the fringe of index <fringe>;
`lastpeak(fringe);` returns the index of the last peak defined on the fringe of index <fringe>;

or graphic functions used to visualize peak locations, such as

`dotpeaks(fringe);` displays a dot at each of the pixels corresponding to the peaks of the fringe of index <fringe>;

or the analogous `dmdpeaks` which replaces the dot with a (x).

Scaling and I/O functions

Whenever an accurate quantitative analysis is to be carried out, peak locations (together with the associated fringe values) must be returned to the user in terms of real world coordinates. On this purpose scaling and I/O functions are provided which transform the screen coordinates stored in the peak arrays into real world coordinates and output them on an

ASCII format file opened by the user program. Reconstruction of the peak arrays from a peak file previously saved is also possible.

Scaling functions are mainly intended for internal use with the exception of those functions which are accessed by the user to define scaling parameters. On this respect note that when different fringe patterns are processed which are observed on the same object from the same point of view by the same imaging system, the scaling parameters can be loaded from a previously saved file.

Even in the simplest case where the object surface is plane, an a-priori evaluation of the trasformation matrix from screen to world coordinates (and of its inverse) can prove a difficult task. In fact, not only the magnification ratio is affected by several different factors such as the focal length of the imaging lens, the distance of the object and the sensing area, but it also varies with direction when the pixel aspect ratio differs from unit. Furthermore, the most convenient orientation of the object coordinate system could not necessarily coincide with that of the screen coordinate system. A further and greater difficulty derives from the perspective distortion which is introduced by oblique viewing. It must eventually be pointed out that the image may also be distorted by the imaging lens and/or the sensing device; should this latter type of distortion be significant, it would prove necessary to carry out a pointwise evaluation of the trasformation matrix.

When the coordinate trasformation is carried out on a plane surface sufficiently small with respect to the its distance from the point of view to involve only a shear trasformation, all the probems mentioned above can be overcome by evaluating a-posteriori the trasformation matrix by means of functions accessible by the user. The trasformation matrix is then stored, together with its inverse, in the exchange data set.

The user is required to identify the location of the origin of the coordinate system attached to the object on its image on the screen and to input a reference length on each of the two axes (which, when imaged, do not necessarily form an angle of 90°) and the resulting projected length on the screen axes. The reference length can be input in units chosen by the user whereas the projected length must be given in screen coordinate units. The task can be quite easily accomplished by imaging a test target (or the object itself, provided that appropriate reference points have been marked on it) and by subsequently re-tracing the reference segments with the trackball cursor.

`origin(i0,j0);` locates the origin of the world coordinate system;

`xscale(u,di,dj);` defines the first column of the trasformation matrix;

`yscale(u,di,dj);` defines the second column of the trasformation matrix;

`scale();` evaluates the inverse trasformation matrix;

TEST EXAMPLES

A few examples are reported where a fringe pattern, obtained by Digital Speckle Pattern Interferometry, represents the out-of-plane displacements detected on an inflected membrane with a sensitivity of about 4 fringes/μm. When this particular technique is employed, correlation fringes are obtained by subtracting the gray levels of two speckled images produced by a speckle interferometer. A high noise level is always present; poor results are hence generally obtained if the usual image processing techniques or automatic fringe analysis algorithms developed for high quality interferograms are used.

A central region (512x512) of the fringe pattern has been subjected to a histogram equalization while a smaller internal zone has been processed with a non-linear filter developed on-purpose (7x7 window size) and, subsequently, with a Sobel filter (5x5 window size).

Figure 4 shows two different plots of the gray level obtained by the program lines listed below.

```
int i1,j1,i2,j2;
float scale=1.5;
.........
sgt_ball(&i1,&j1,&i2,&j2);
mksgt(i1,j1,i2,j2);
rdpixels();
base();
histo(scale);
value(255/scale);
.........


int n,i[8],j[8];
float scale=2.0;
.........
sgs_ball(&n,i,j);
mkspl(n,i,j);
rdpoints();
base();
graph(scale);
.........
```

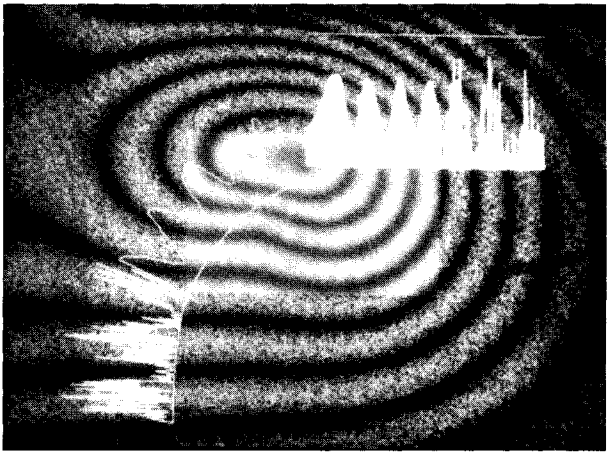

Figure 4.

An example of manual tracing of a fringe center line followed by peak optimization is illustrated in figg. 5, 6 and 7. The corresponding program lines are:

```
int n,i[8],j[8],fringe=0,peak;
.........
sgs_ball(&n,i,j);        (see fig.5)
.........
mkspl(n,i,j);
base();                  (see fig.6)
.........
addpeaks(fringe);
for(peak=0;peak<npeaks(fringe);peak++)
    {o2peak(fringe,peak);}
dotpeaks(fringe);        (see fig.7)
.........
```

The final examples reported in fig.8 and fig.9 are relative to the automatic peak location obtained by:

```
int n,i[8],j[8],fringe;
float increment=1.0;
.........
sgs_ball(&n,i,j);
mkpts(n,i,j);
addfringes();
for(fringe=0;fringe<nfringes();fringe++)
```

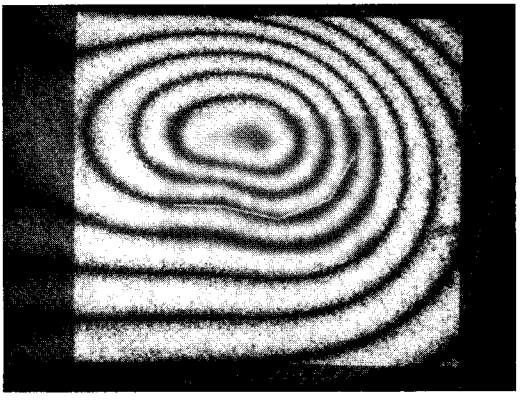

Figure 5.

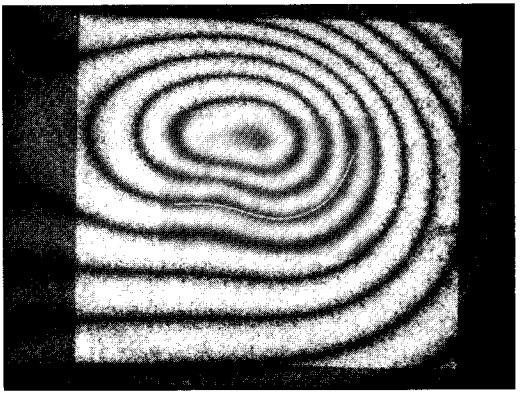

Figure 6.

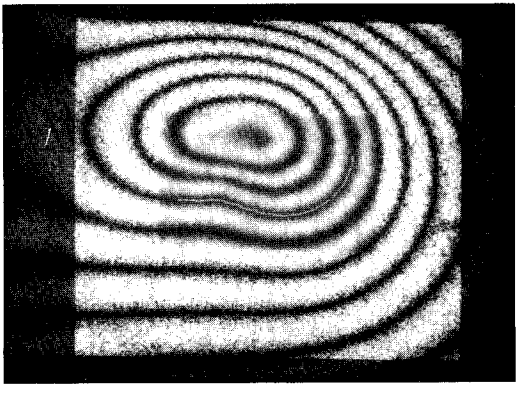

Figure 7.

```
    {
    gopeak(fringe,increment);
    o4peak(fringe,lastpeak(fringe));
    udpeak(fringe);
    if(b_ball()) break;
    }
for(fringe=0;fringe<nfringes();fringe++)
    dotpeaks(fringe);      (see fig.8)
.........
```

An analogous procedure was employed for locating the fringe peaks shown in fig.9 which have been evidenced by the `dmdpeaks` function. The increment between peaks was in this case of 8 raster units.

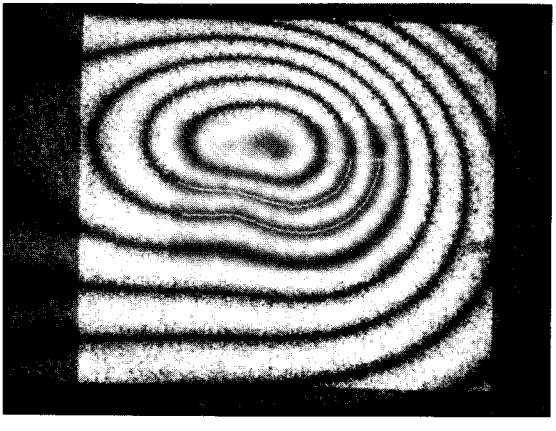

Figure 8.

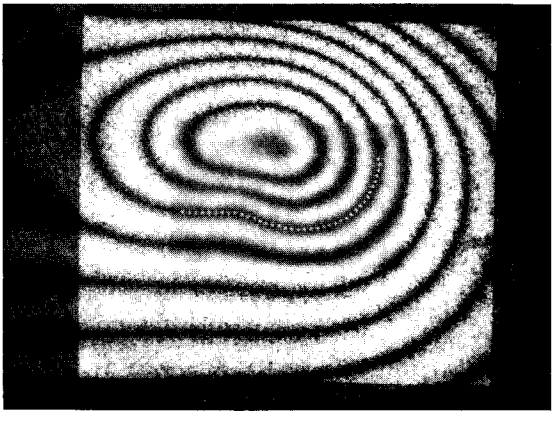

Figure 9.

CONCLUSIONS

In the paper the structure of a C-library for fringe analysis is outlined and the routines directly accessible by the user briefly described.

The library is obviously susceptible of several improvements. More sophisticated peak optimization functions should be developed, using, for example, a local approximation by a second-order surface. Further improvements are necessary to enable more robust procedures to be built-up and memory allocation optimized. Higher level functions might anyhow be implemented even using only the functions currently available.

User friendly problem-oriented programs can be easily developed; the level of automation can be particularly high when the programs are aimed to analyze fringe patterns presenting similar features, typically encountered within the same class of problems.

REFERENCES

1. Reid, G.T. 'Automatic Fringe Pattern Analysis: a Review' *Optics and Lasers in Engineering*, Vol.7, pp.37-68, 1986

2. Huntley, J.M. 'Speckle Photography Fringe Analysis: Assessment of Current Algorithms' *Applied Optics*, Vol.28, pp.4316-4322, 1989

3. Chen, T.Y. and Taylor C.E. 'Computerized Fringe Analysis in Photomechanics' *Experimental Mechanics*, Vol.29, pp.323-329, 1989

4. Yatagai, T.'Automated Fringe Analysis Techniques in Japan *Optics and Lasers in Engineering*, Vol.15, pp.79-91, 1991

5. Robinson, D.W. 'Automatic Fringe Analysis with a Computer Image-Processing System' *Applied Optics*, Vol.22, pp.2169-2176, 1983