# APPLICATION OF MOIRE TECHNIQUE AND QUALITY CONTROL OF A TRANSPARENT OBJECT

By

CARL B. LOPEZ

Bachelor of Science
Oklahoma Baptist University
Shawnee, Oklahoma
1990

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1993

OKLAHOMA STATE UNIVERSITY

APPLICATION OF MOIRE TECHNIQUE AND

QUALITY CONTROL OF A

TRANSPARENT OBJECT

Thesis Approved:

[illegible]

Thesis Advisor

[illegible]

[illegible]

[illegible]

Dean of the Graduate College

## ACKNOWLEDGMENTS

I would like to thank Dr. Jersy Krasinski for allowing me to be a part of his research group and his help preparing this paper. Dr. Gary Pearson I thank for his help getting the programs started and debugged. I also thank Dr. Teague for help with the computer routines, use of the frame grabber and for serving on my commitee. I would also like to thank Dr. Jim Baker for serving on my committee. I would also like to thank Scott Holmstrom for all his help setting the format of the final draft.

I would also like to thank Denise, my wife, for her patience and support during this last semester of late hours. My parents continued devotion and constant encouragement during the good times and the trying times was more benificial than they will ever know, thanks for always being there.

I would like to thank God for all the many ways he has blessed me including a supportive family and so many wonderful friends. With out him this paper could have never been written.

TABLE OF CONTENTS

## LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

Measuring has always been an important part of technology. By definition, measuring is comparing some test object to a known reference. Of course there are many things to measure and almost as many ways to measure them. Often the key is to find the best way to obtain the measurement needed. One of the most common and versatile ways to measure something optically is using interference. "Optical interference may be termed an interaction of two or more light waves yielding a resultant irradiance that deviates from the sum of the component irradiances'[1]. The results of this interaction can be used for many applications. One can determine the thickness of an object, its refractive index, surface defects, and even defects within the material of an object. Another not quite so common way to find the same types of measurements as mentioned above is the moire method.

For the interferometeric and moire techniques, a fringe pattern is created. Unlike the interferometric methods, moire methods do not directly use the wave properties of light. Moire uses the light for illumination only, and actually measures ray deflection. To a large extent this method can be explained with optical geometry.

For our research the moire technique was used. Before applying a specific technique of interpreting a fringe pattern, some back ground information on interferometers will be given to show how interference techniques and moire deflectometry are related. Some advantages of using Moire techniques to measure strongly distorted objects like the jet canopies will be given.

For the testing of jet canopies, moire deflectometry is an excellent tool for finding a local ray deflection. This is done by creating a fringe pattern, processing the information that corresponds to the ray deflection of a phase object in this case a jet canopy, then calculating the slope of each of those fringes and finally plotting the results. In the second chapter examples of an interferometer will be discussed and compared with the moire deflectometer. Conclusions will be drawn as to why the moire deflectometer is the better choice for this application. The theory of the moire method will be shown in chapter three. Computer routines that extract the interferometer from the fringe pattern will be discussed in chapter four and listings of these routines will be shown in appendix A. Chapter five will contain the application of the theory and computer routines to produce the ray deflection plots for the canopy. Finally in chapter six conclusions and recommendations will be stated.

Testing of the canopies could be done by methods other than those mentioned, but the moire deflectometry method was selected for several reasons. Except for the major scratches caused from handling the canopy one can not tell if there is a change in the focal length within the material by merely looking at the canopies. By using the moire techniques, we will show that the fringes reveal even slight changes in the focal local length of the material making up the canopy.

One goal of this research was to build a moire deflectometry system with the capabilities to produce and record the fringe pattern of the canopy. The other goal was to develop the software that will compose a map of the ray deflection created by the defects of the canopy.

It will be shown that by using a moire deflectometer to test the canopy one can benefit by using the two main advantages of this method; adjustable resolution and capability to test large objects. By being able to adjust the sensitivity of the deflectometer, testing can be done on phase objects; such as the canopies, that have slightly distorted, strongly distorted surfaces or inhomogenous defects. The variations in the defects create the need for this adjustable precision, but since the high sensitivity limit is comparable to the interferometer methods measurements can still be quite precise. This precision can be on the order of a fraction of the wave length. These factors show that indeed the use of the moire method will yield accurate results with more freedoms than that of the interferometer methods.

# CHAPTER II

# CREATING FRINGE PATTERNS

## Interference Techniques

One of the most well known interferometers is the Michelson interferometer. It is not only well know but also one of the most versatile interferometers used for measurements. This measuring device uses an interference pattern that is created by a difference in the optical length[2]. It can be used for all of the examples mentioned in the introduction and is a very precise way to obtain various measurements.

In figure one an example of the Michelson interferometer is shown. The light wave enters the interferometer and is split by a beam splitter that divides the wave into two parts. These two parts then proceed along two different paths and come back together at the detector. The detector is set at a point where the interference pattern can be examined. One of the mirrors is mobile along it's axis to adjust for the pattern. One of the mirrors can be replaced by a surface to create an interference pattern of that surface. The reflection of that surface will change the path of the light wave if there is any defect in it. A phase object to be tested may also be placed in one of the arms of the interferometer to create the interference pattern.

When placing an object into the arm of the interferometer one can see any deformation contained by the phase object. This change might be caused for one of several different reasons. One might see a change in the fringe pattern if there was a change in the index of refraction within the material of the phase object. The same change of the fringe pattern could be created by using a phase object

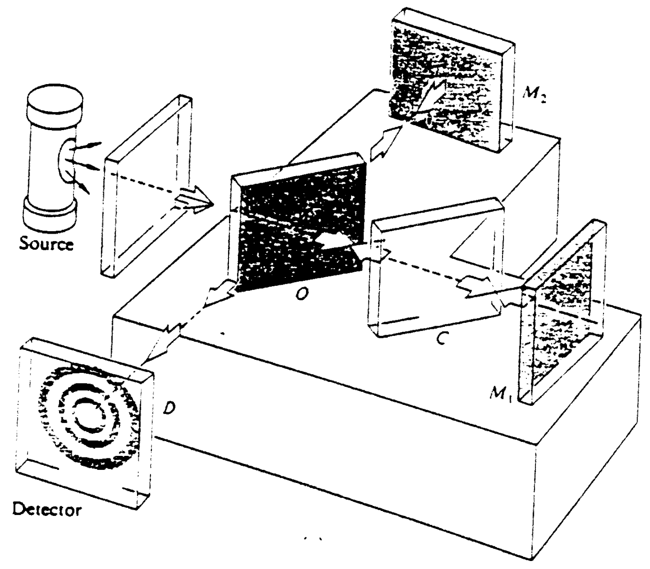

Figure 1. Diagram of the Michelson interferometer

[1]

that has a different thickness at one point than another. The change could also be caused by some defects on the surface of the object.

The Michelson interferometer has capabilities of high accuracy in measuring lengths. When moving one of the mirrors just $\lambda/2$ closer or farther away from the original position a single fringe will move position of the previous adjacent fringe. Therefore one can count the number of fringes N to determine the distance traveled by the mirror

$$\Delta d = N * \frac{\lambda}{2}. \tag{1}$$

With technology of today this can be done with electronics, but for high precision the alignment is crucial.

An extension of the Michelson interferometer for larger transparent phase objects would be the Mach-Zehnder interferometer. It uses two beam splitters and two mirrors as shown in figure 2.

In this set up the phase object is placed into on arm of the interferometer. Again, as in the Michelson interferometer, the separated parts of the wave come back together and the interference pattern at the detector is caused by the slight path difference created by the phase object itself. Even though larger objects can be tested it is extremely difficult to align the system.

## Moire Techniques

Unlike the interference techniques shadow moire is done by creating a shadow of the Ronchi rulings onto a surface[4]. A Ronchi ruling is a simple grid of parallel lines that has a certain pitch or spacing between the lines. This set up is shown in figure 3 and has the capabilities of being used on large scale phase objects like the Mach-Zehnder interferometer[5]. In figure 3 an example of the shadow moire method is shown. The Ronchi rulings are projected onto a young child's face.

This method has low spatial resolution but can be used to create a surface map of the contour elevations. The resolution is adjustable but only by replacing

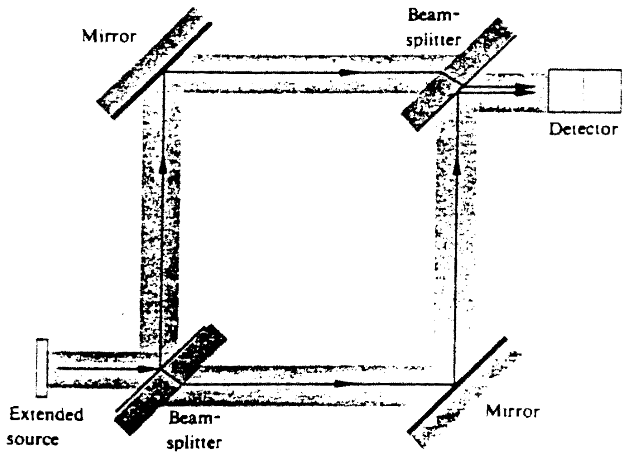

Figure 2. Diagram of the Mach-Zehnder interferometer

[1]

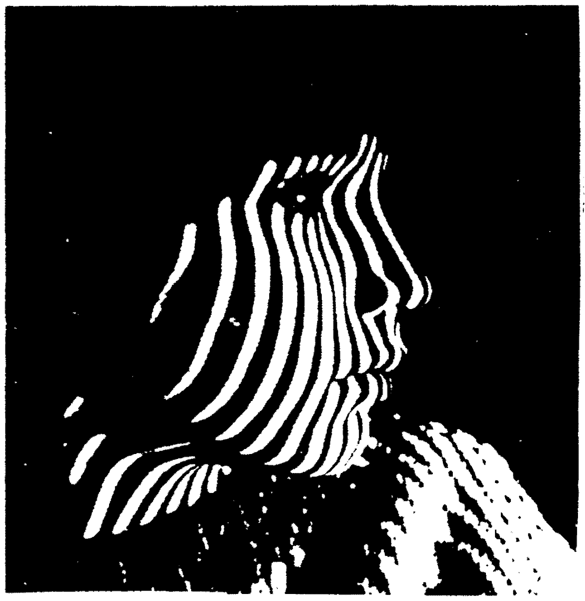

Figure 3. A contour map of a human head

[5]

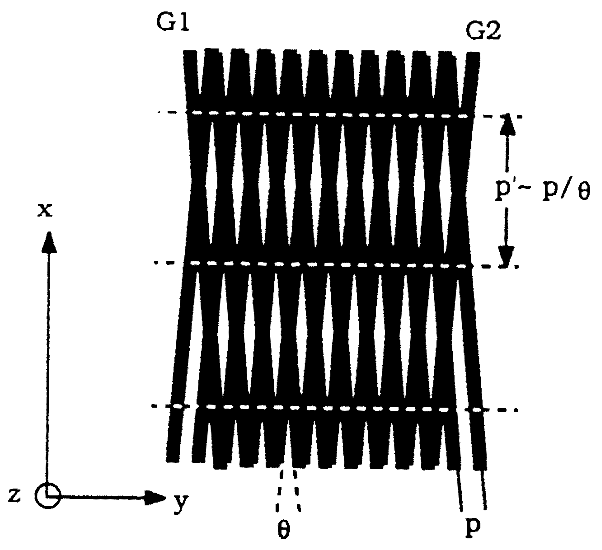

Figure 4. Moire pattern of two Ronchi rulings superimposed at a small angle $\theta$ [1]

the grating with one of a different pitch. One could also project this fringe pattern on larger surfaces.

The last two examples to be shown are similar to the shadow moire method and use two Ronchi rulings. Moire deflectometer and the Talbot interferometer measure slope derivatives instead of lines of equal height as shadow moire[3]. The difference between these to methods is where the phase object is placed within the deflectometer. These methods produce fringe patterns by the super positioning of two grid patterns with similar pitches. Since a shadow of the first grating is cast onto the second the sensitivity is adjustable by changing the distance between gratings. The Ronchi rulings (fig. 4) are represented by G1 and G2 and the pitch of each is p. The angle between the two is represented by $\theta$. The pitch is adjustable by varying $\theta$ and the resultant pitch is $p'=p/\theta$.

The coherent light is projected through these grid patterns and the shadow and projected light can be observed. Depending on several factors (e.g., the pitch of the gratings, the distance between gratings, and the reflection off or transmission through the incident phase object) results can be determined about the sample. One of the main benefits of using this technique is the fact that it has adjustable resolution and it can be applied to large surfaces[4].

Moire deflectometry is used to produce a fringe pattern of a phase object that is actually a map of ray deflection[4]. Unlike the interferometric techniques that measure a difference in the path traveled by the wave parts. Even though both techniques can produce a fringe pattern caused by the same defect there is a difference in how the fringe pattern is created.

In figure 5 the moire deflectometer and the Talbot interferometer are shown. Component number one is a collimated light beam. For our set up, a HeNe laser is used as the source and it is projected through a pin hole to get the resulting collimated beam. The fringe pattern is created by the use of Rhonchi rulings (G1,G2) which are two sets of gratings that consist of parallel lines of a certain pitch[6]. The pitch is the spacing of the lines and can be selected depending on the resolution you desire. The gratings are either parallel or one is rotated with respect to the other by some angle. For this project, rulings with the pitch of 4 lines per millimeter are used and the angle is varied. The last component is the screen on which the fringe pattern will be projected. It is partially transparent so that a camera can be used to make an image of the pattern[7]. Once these components are combined a phase object can be placed in the beam's path before the gratings.

For a defect on the canopy's surface or within the material that makes up the canopy the fringe pattern will be changed from straight lines to distorted lines. Since the beam is collimated the change in the fringe pattern is due to a focal length change in the material. The sensitivity can be adjusted by changing the angle between the gratings[2]. One problem with interferometric methods is due to their high sensitivity, which means that there is need for high mechanical stability and low noise of the system. With moire techniques the results can simply be calculated using geometrical optics by means of ray tracing. This is possible since defraction effects can be minimized by placing the second grating at a distance which is an integer multiple of the Fourier plane[4]. Sensitivity of the deflectometer is not dependent on wavelength of the light being used, but is adjustable by varying the grating pitch or spacing. Thus at any sensitivity of the system there is less need to control vibrations of the system.

There are many uses of the moire deflectometry method[4]. It has been used to image air flow from the after burner of a jet fighter in an aircraft hanger. Using moire for strain analysis, one will observe a difference in the fringe patterns that are produced with and without stress being applied on the phase object. This method can make sensitive measurements without having to have a special environment, and unlike the interference techniques the resolution is adjustable. These properties of moire make it more desirable to use when testing larger phase objects such as the jet canopies.

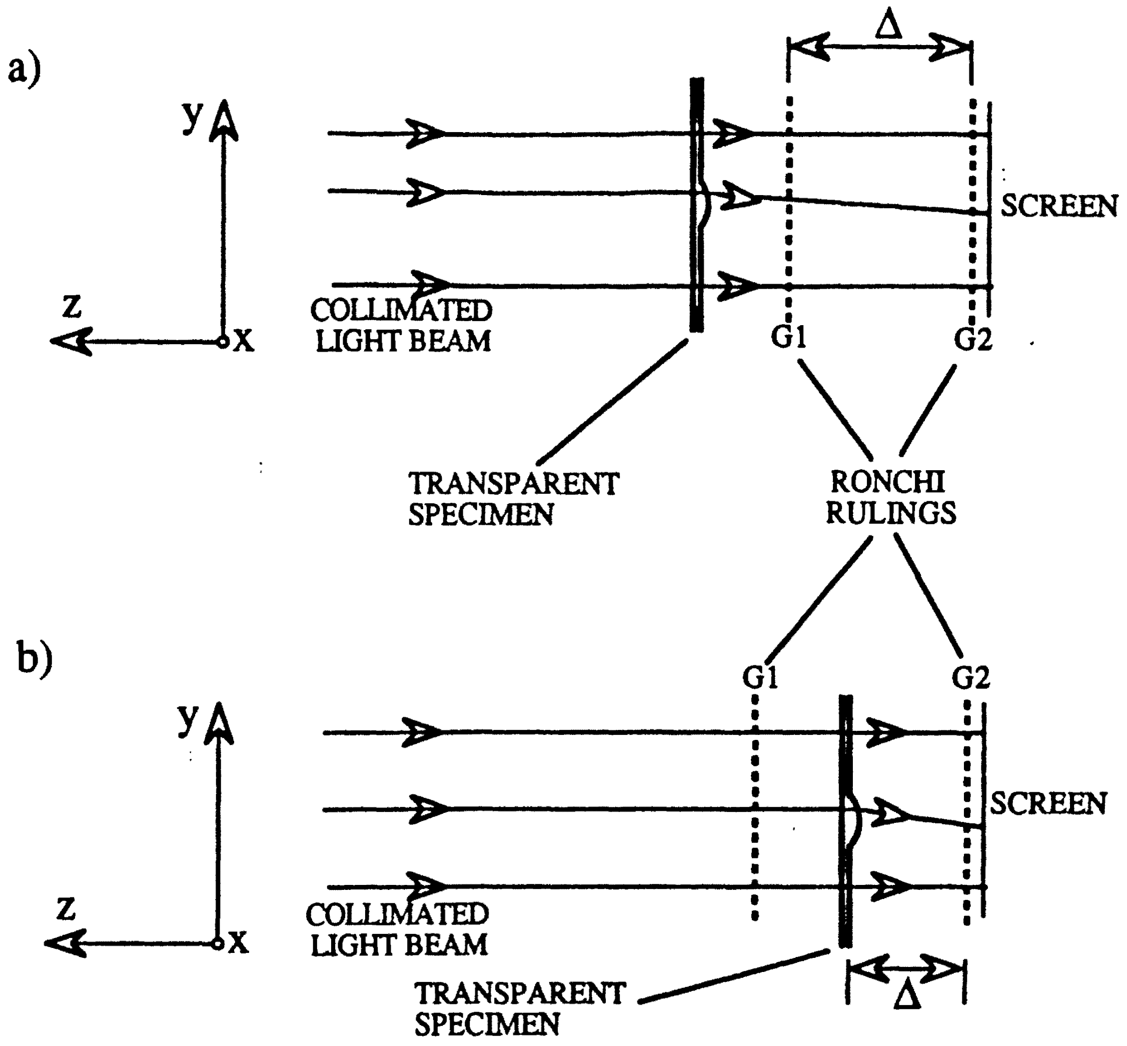

Figure 5. a)Diagram of moire deflectometer, b)Talbot interferometer

# CHAPTER III

# A SIMPLE EXAMPLE USING MOIRE DEFLECTOMETRY

For a phase object that is transparent but curved one can use moire techniques to see the aberrations in the material that makes up the object. A change in the focal length of a surface would be one type of aberration that can be checked. The phase object can also be inspected for dents or inhomogenous material deformations that cause changes in ray deflections that can be seen in the fringe pattern.

Moire fringes are produced when the collimated light beam projects the shadow of the first grating upon the second grating. Any defect in the phase object causes a deflection of the light beam. This deflection will result in a fringe shift. The result of one fringe shift at the screen corresponds to a deflection angle of $\phi$=p/$\Delta$ (where p is equal to the pitch and $\Delta$ is the distance between gratings). The slopes of the fringes are also related to the deflection angle by $\phi$=2*$\beta$ (when $\beta$ =dh/dx the slope of the distorted fringe)[4]. The moire deflectometer's accuracy is bounded by the diffraction limit [8][9] (i.e., dxd$\phi > \lambda/2\pi$). Reading this equation dx is the spatial resolution, d$\phi$ is the angular resolution of the instrument and $\lambda$ is the wavelength of the light. This equation is also the classical interferometer limit.

If there is a change in the focal length the moire pattern will be effected by a rotation of the fringes at the optical center by an angle $\alpha$. This angle can be calculated using the following equation

$$\alpha = \tan^{-1}(\frac{\Delta}{\theta * f}) \qquad (2)$$

By rearranging the previous equation the local focal length can be determined:

$$f = \frac{\Delta}{\theta * \tan\alpha}. \tag{3}$$

The accuracy of the local focal length of measurement df is given by a formula [4]

$$df > q\frac{p}{d}\frac{f^2}{a} \tag{4}$$

where a is effective aperture, and q is resolution of measurement (minimum resolvable fraction of a moire fringe).

By merely looking at a canopy, it is difficult to tell if there is a change of direction of optical rays propagating throughout the material. It will be shown, by using the moire techniques, that the fringe image reveals even slight changes of the ray's direction in the material of the canopy. The moire deflectometer produces a fringe pattern for the canopy and software has been created to compose a map of the local optical power of the canopy from the fringe pattern.

In our experiment a television camera recorded the image that was transferred into a computer using a frame grabber. By digitizing the fringe pattern computer routines can be used to manipulate the image and calculate the focal length of the phase object. These computer routines will be discussed in the next chapter for now the proof that the focal length can be calculated will be shown.

For an example of this process a lens of known focal length (1000 mm) was implemented into a piece of plastic as shown in the figure 6. This was done by drilling a circule into a piece of plexi glass and placing the lens into it. This creates a phase object of known focal length that can be placed into the deflectometer and a picture of the fringe pattern can be taken. Then that digitized image can be processed by means of several computer programs to get information about the optical parameters of the lens.

Measuring the fringe rotation angle $\alpha$=.6981 rad, distance between gratings $\Delta$=167 mm and angle between gratings $\Theta$=.0645 rad a focal length f=989.22 mm is calculated. This represents and error of ~1%.

By using the deflectometer, data can be obtained and used to map the focal information from a given object. Using a frame grabber to digitize the fringe pattern, an image file can be made and the process of constructing the needed images can begin. The results can then be used to create a map of the local focal length or a contour map of the surface. In figure 7 a diagram is presented to give an overall picture of the process.

Using the process described in the figure 7 a map of the ray deflection can be generated using the theory of moire. In the following chapter the computer routines will be discussed and the method of finding the slope of the fringes will be shown.

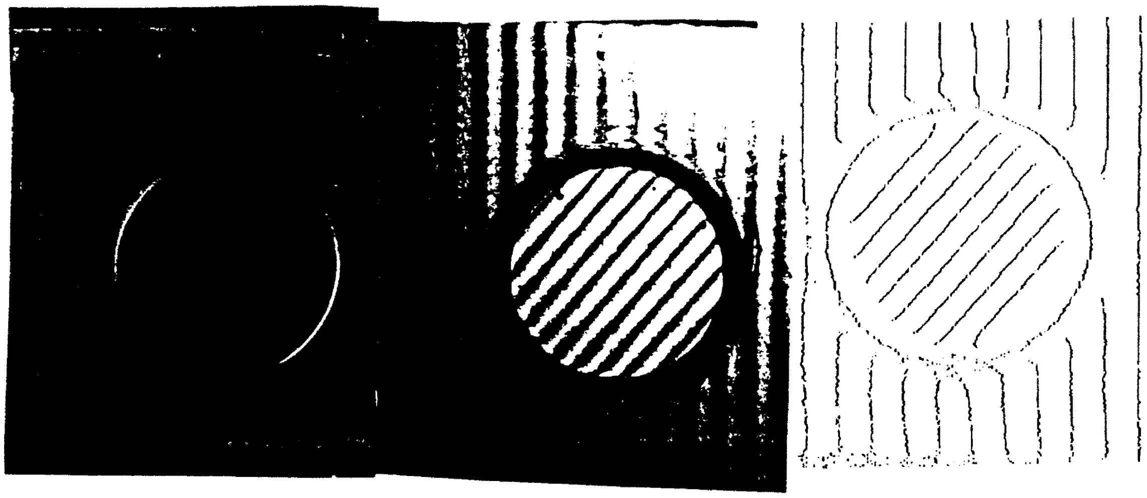

Figure 6. Lens, moire fringe pattern, and computer matched lines from the fringe pattern

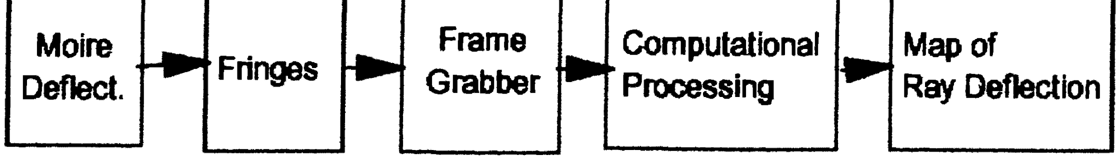

Figure 7. Block diagram of process from moire fringe pattern to the map of ray deflection.

# CHAPTER IV

# COMPUTER ROUTINES FOR AUTOMATED ANALYSIS OF MOIRE PATTERNS

## Explanation of Routines

Several computer programs have been developed to manipulate the fringe patterns of a phase object that have been obtained using a moire deflectometer. The goal was to take the fringes and obtain the change in slope information and use it to determine the change in focal length. In figure 8 a flow chart of how these routines will be used together after obtaining the digitized image.

These C programs are for filtering and extracting information from the fringe patterns. The data will have obtained noise from the various steps of the process it takes to get the information converted from the screen to the map of ray deflection. We desire to filter out as much noise as possible and limiting loss of usable data.

One way the image will pick up noise is through the process of transferring the actual data on the screen to the initial digital image. The screen might have some slight movement when the image is taken or there could be some imperfections on the screen itself. Noise has also entered when taking the photograph of the screen then using that photograph to make the digitized image of the fringe pattern. By

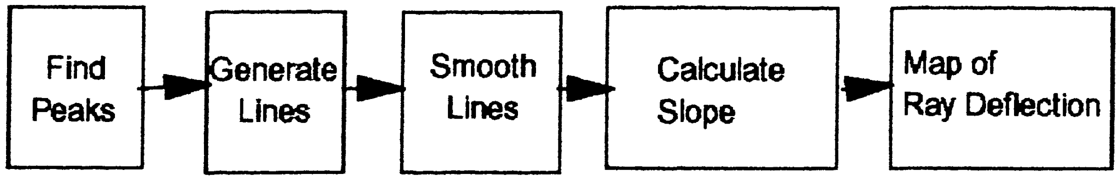

Figure 8. Block diagram of steps in computer processing

3 4 6 7 8 5 4 3 5 6 7 8 9 7 6 4 2 1 0 2 3 4 5 6 7 5 4 2 1

Figure 9. Sample data before the first image processing routine.

capturing the image directly from the screen this last type of noise can hopefully be reduced. Most of the problems that occur in attempting to make the map of ray deflection can be found in the scratches of the phase object that show up as noise. This noise is the most difficult to filter out.

The first routine takes the digitized image that has been created by the frame grabber and finds the peaks and valleys of the fringe pattern. The fringes are not constant intensities but vary from dark to light. The digital image that has been captured has a 256 gray scale. The computer can find the locations of the peaks and valleys by a simple analysis of data.

The routine will scan the image file for a single line 512 pixels wide. This line of data might consist of 512 different intensity values, or one value for each pixel. The routine scans through the data line the values are compared and positions of each intensity are saved. This is done by starting at one end a comparing each pixel with its neighbor.

For example if one had the data given in figure 9 where 0 represents white and 9 represents black and the value's in-between represents an increase shade of gray. The routine would start at 3 and check to see if the neighboring pixel was higher or lower since it is higher it will change the value of the pixel from 3 to 0 and move to the next pixel ect... When the routine gets to the 8 its neighbor is 7, so it saves the position of pixel 8, changes it's value to 9 and proceeds. Figure 10 shows the results of this example line of data after it passes through the first image processing routine. This process returns the peaks for the black fringes. A similar routine is used for the white fringes except the check would be for the white pixels.

```
0 0 0 0 9 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 0 0 9 0 0 0 0
```

Figure 10. Sample data after the first image processing routine.

Once the file of the fringe pattern has been converted to and image pattern of lines another routine is used to cut out the area of the file to be used. This routine is necessary since the frame grabber is not adjustable for the exact frame size of the picture needed. It simply prompts the user for the dimensions to which the image should be reduced and returns a new image file of the curtailed image. It can be seen in figure 11 that the usable data is limited to a certain area so it must be removed before further processing.

The next routine will create lines from the processed image file. The routine will take each line in the image file and make individual line files. These line files are made by starting in one corner of the image file and proceeding across until the beginning of an individual line is found. Then it proceeds up the line recording the location of the data for that specific line for an individual line file. At the end of the line the routine picks up just to the right of where the last line started and proceeds to find another line.

For example in figure 12 the routine is moving up the first full line of data. The solid line represents the data recorded in the first line file. The pixels that have been used are turned the same color as the background so that they will not be used again. At point a the routine is searching for the next black pixel. It searches at a search radius of R. This R is set by the user and can be changed depending of the quality of the image file produced thus far. At the top of the first column of data the first line file will be closed. The position moves to the pixel that is now the most left black pixel on the first row of data, and the second line file will be opened. This process is continued until all of the black pixels on the first line of the image file are white.

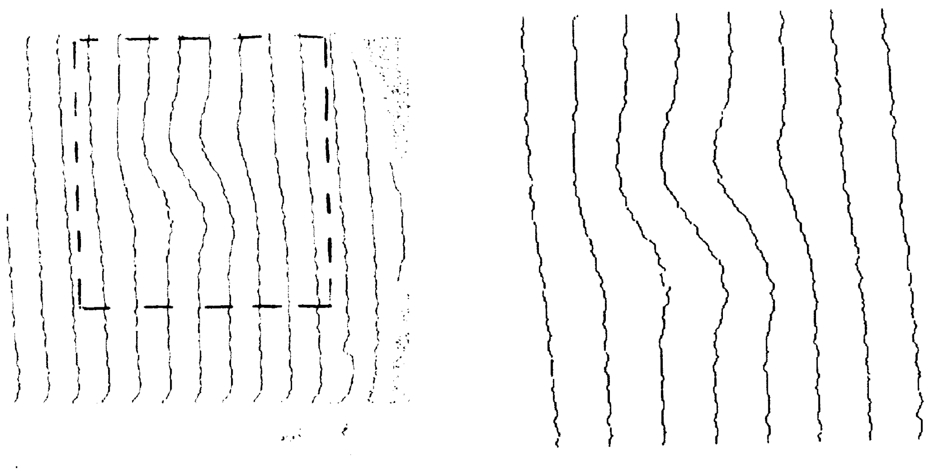

Figure 11. a) Original image file b) Cut image file

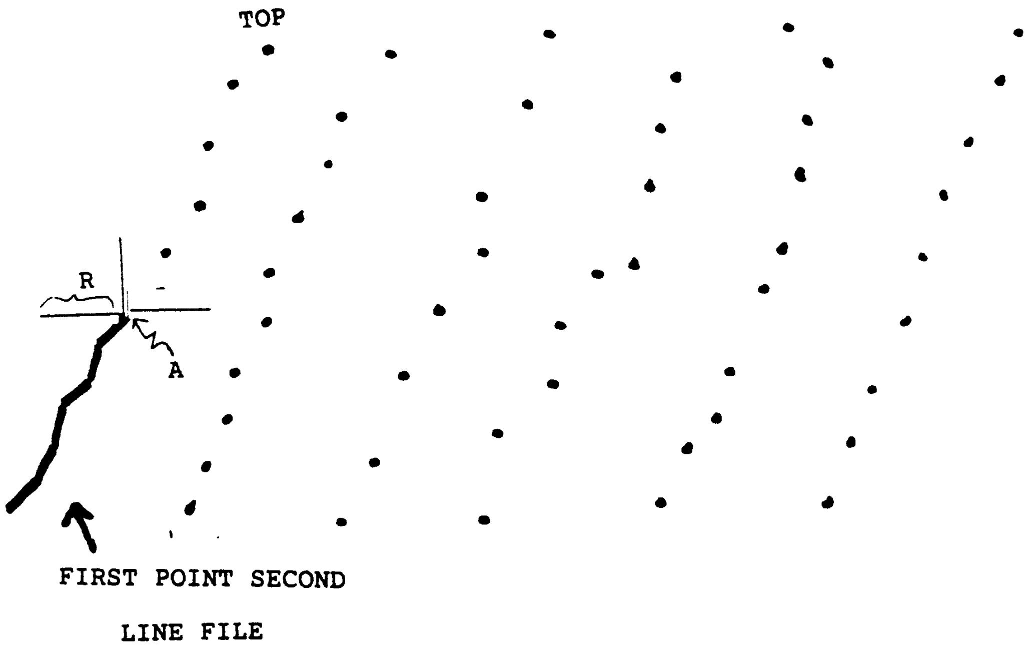

Figure 12. Diagram of constructing the line files.

One problem with this routine used to build the line files is the fact that the data in the image may have data that drifts off the actual path. This may be caused by a scratch in the phase object or sharp defect within the material. Adjusting the resolution can sometimes fix this problem but not always. Figure 13 a) shows broken areas in data of the dark fringes that are going to cause a problem in building the line file. In figure 13 b)–d) show where the some of the line files ended prematurely because there were no pixels within the search radius to continue the line(s). We had to use a search radius of 20 to get complete line file data from the image file shown in figure 13 e), but by extending the search radius to 23 the lines cross over producing undesirable data. One can see that the search radius must be select carefully depending on the quality of the image file and the deformations of the fringes.

The first three routines are standard for our purposes. They do not require any special set up but the smoothing routine may or may not be used depending on the resolution and the quality of the image that has been manipulated thus far. The image file can be smoothed again by a smoothing filter routine if necessary. The smoothing is done by a simple averaging of the data file. One problem is that the more an image is smoothed the more data precision is lost, but the lines in the file must be continuous for the next routine to work properly. Obviously there is a trade off here to get usable data, but there is sometimes a need for reducing the precision of the image so that the lines will remain continuous.

The smoothed individual files can now be manipulated to find the slope of each individual fringe at known sections of the fringe. Line regression is the process used to obtain this slope. The routine can be modified to calculate the slopes over a longer or shorter section of the line.

Once the individual lines are constructed and the localized slopes are found. The map of the ray deflection can be plotted. Since the line files have lost some of their accuracy as they are manipulated the desire is to filter and smooth only as much as necessary to produce the most accurate information.

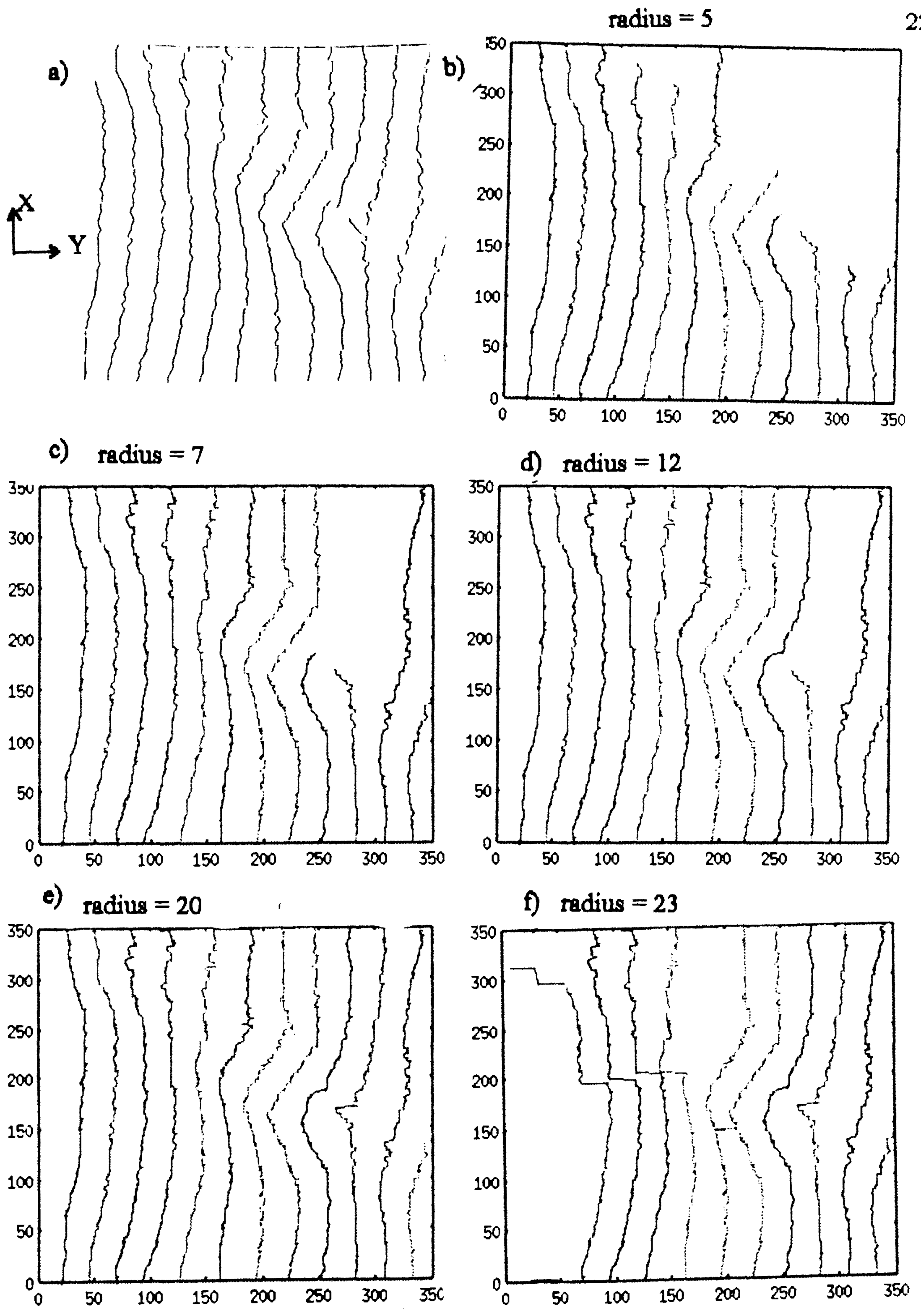

Figure 13. a) Diagram of data that is noncontinuous b)– f) Examples of broken line files when the search radius is 5,7,12,20,23

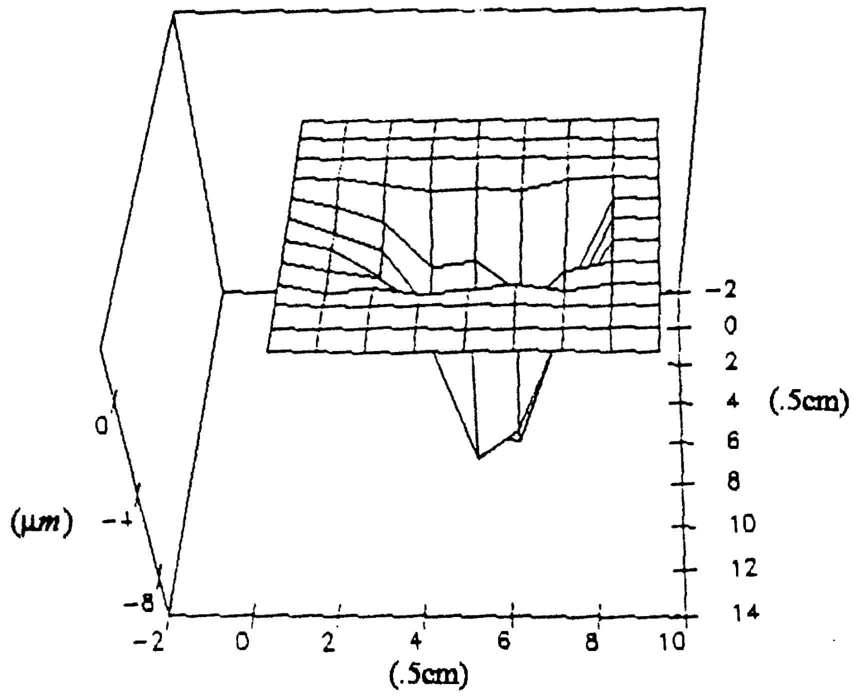

Figure 14. Surface map of the distorted plexi glass plate

By using other more complex algorithms some more accuracy might be saved in the process of finding the focal map. Another way to get more accuracy is to use the intensity across the entire fringe. The intensity of the fringe is a sine wave and greater precision might be achieved by using this information.

## Testing of the Routines

To test the routines previously described a phase object of known dimension was needed. This element was obtained by taking a two inch square piece of plexi glass that was a quarter of an inch thick and slightly polishing out an indention. After polishing, measurements of the surface were taken using a dial indicator. Setting the plastic on three steel ball bearings to simulate a flat surface the thickness of the plastic was recorded. Figure 14 is a plot of the surface of the phase object.

Placing the phase object into the deflectometry set up the fringe pattern was photographed. The pattern has become distorted from the original plexi glass without polishing as seen in the figure 15.

Processing the fringe patterns through the computer routines the slopes of the original fringes and the distorted fringes can be compared. The following figure shows the lines of maximum and minimum intensities extracted from the image. It can be seen in figure 16 that there is twice as much information when dark and bright extremes are used compared to using only the bright fringe pattern as shown in the previous figure. In principle, the number of lines and the resolution of the grid of lines can be increased by following the constant intensity along the fringes.

Estimating the relative slope, m, is done by using the slope routine. The slopes are calculated using linear regression. This slope can be used to calculate the ray deflection. In the figure 17 the slope is presented over the segment lengths 10, 20 and 25 elements.

It can be seen that the longer the segment the smoother the slope curve. There is some loss in the amplitude for the 25 length segment compared to the 10 length segment, but for the combined map of the slopes this smoothness is needed. Of course more smoothing could be done but with a loss of more of the amplitude and resolution. This information is used to plot the ray deflection map.

By plotting the localized slopes from each line together the map of the ray deflection was created. It can be seen in figure 18 that the slope is changing in the area of the created distortion. This change is directly related to a change in the local focal length.

Using the technique that has been explained above, one can also see the changes of the ray deflections in a canopy. Using the canopy as the phase object it can be seen that the fringe pattern is unchanged unless the focal length in the material changes. By inspection of the fringe pattern one can immediately see where the focal length is changing, but to determine the change the series of routines must be applied.

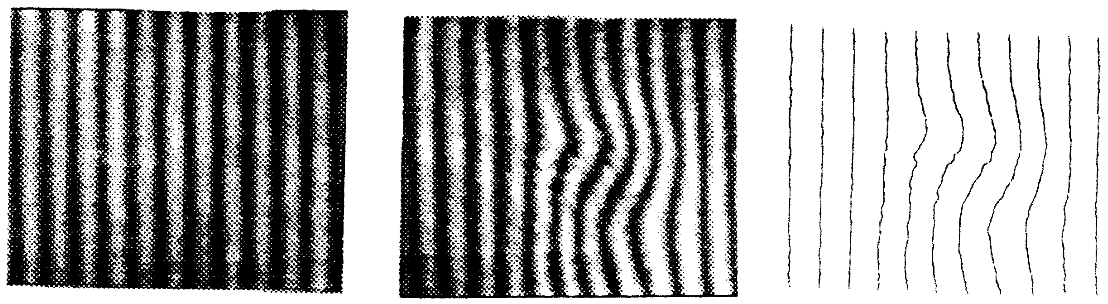

Figure 15. a) Moire fringes of original plastic piece, b) moire fringes of the same plastic piece after small deformation has been polished, c) computer image of fringes.

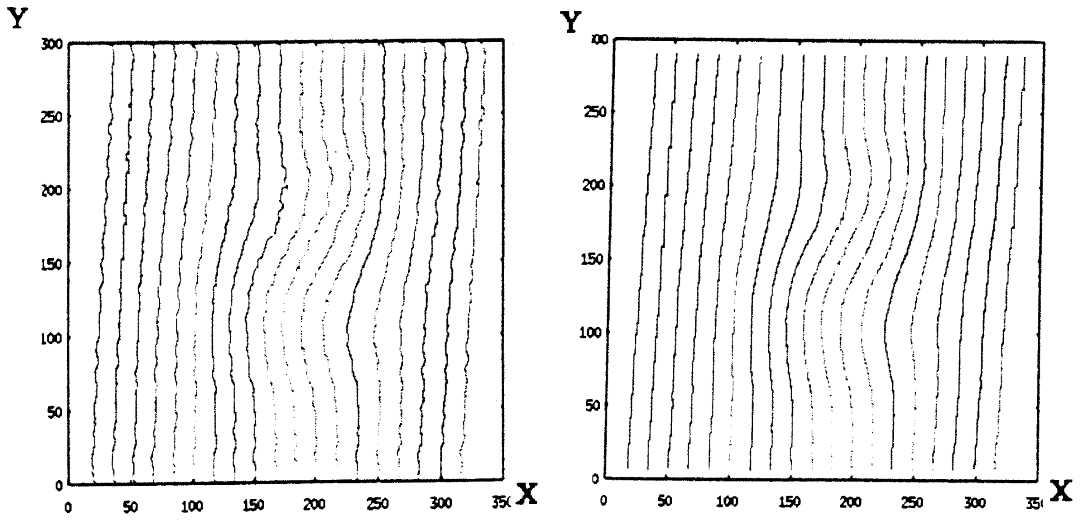

Figure 16. Computer matched lines of bright and dark fringes. a) Unsmoothed lines, b) Smoothed lines.

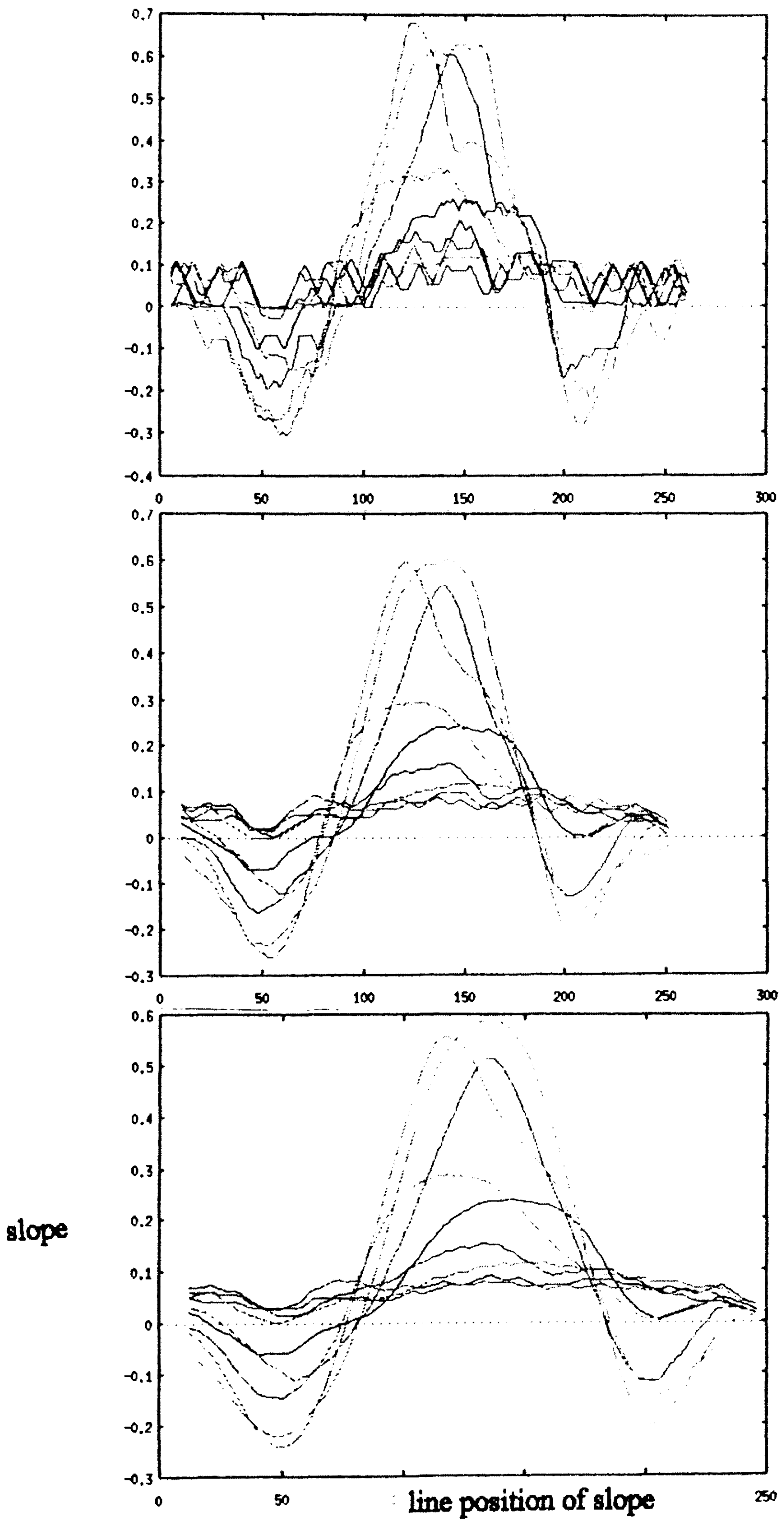

Figure 17. Slope files of 10, 20 and 25 length segments

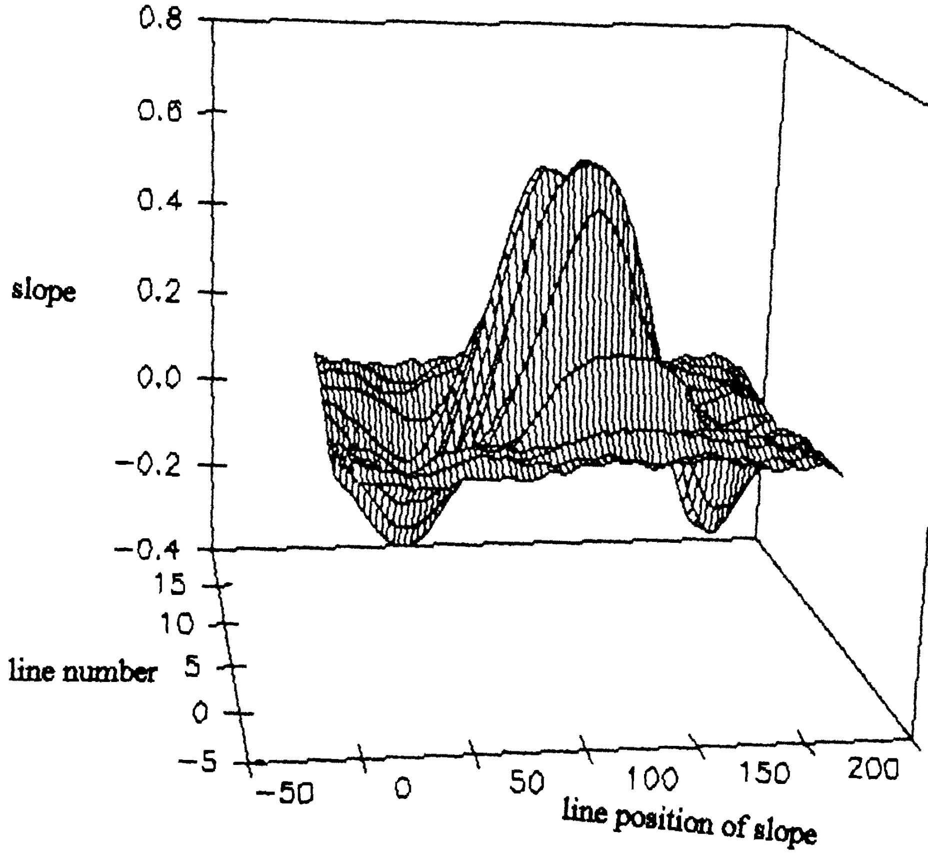

Figure 18. Fringe slope map of the plastic phase object

# CHAPTER V

## APPLICATION TO CANOPY

The following figures are to show how the moire procedure can be used to create a localized slope map of the canopy. In figure 19, an element of the canopy and the fringes from a section of the canopy are shown.

For this set of fringes the distance between gratings is $\Delta$ =50 mm and the angle between gratings is $\alpha$ =.0322 rad.

Large defects seen in figure 19 a) are vivid with the eye. For smaller defects the computer processing can be used to obtain their position. The computer routines will return the localized slope change of all defects.

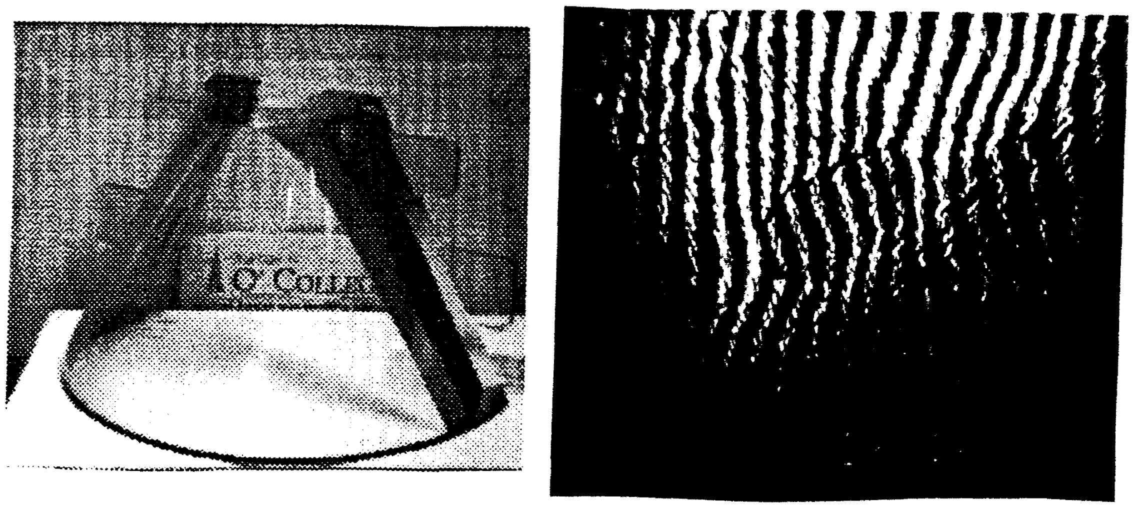

Figure 19. a) element of the canopy b) moire fringes from small area of same canopy

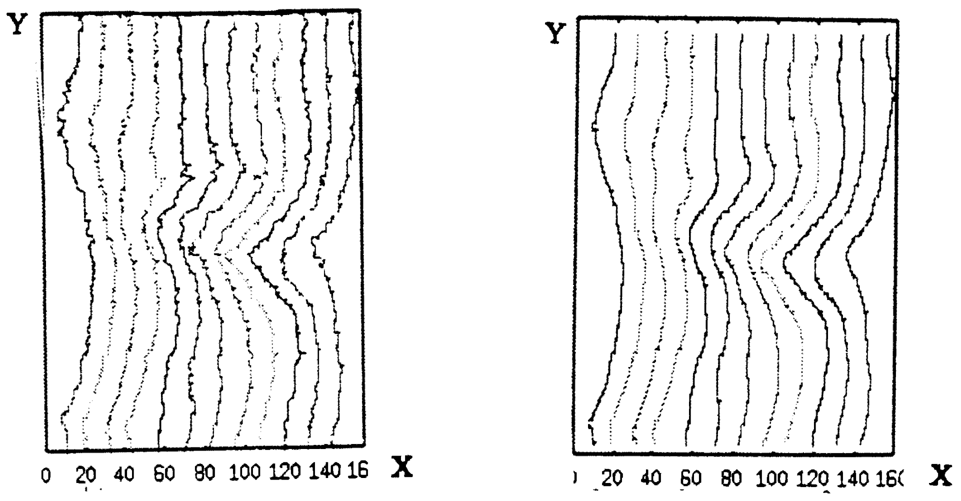

Figure 20. a) computer fit fringes, b) smoothed version of fit fringe lines

The figure 20 shows that the computer generated fringe pattern reveals changing slopes of the fringes. The slope change over this section of the canopy is plotted in the figure 21.

One can see that there is a definite change in the slope and that corresponds to a change in the local focal length in the material that makes up the canopy or a change in the surface of the canopy itself.

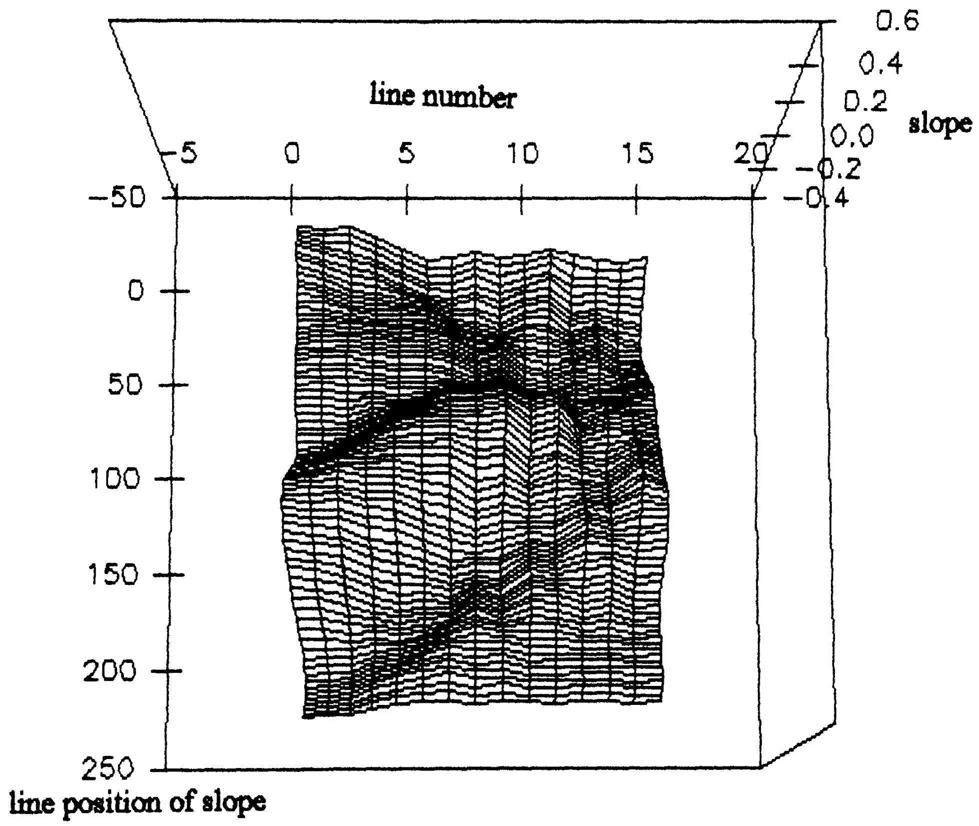

Figure 21. Fringe slope map of a section of the canopy presented in figure 20

# CHAPTER VI

# CONCLUSIONS

The moire deflectometry technique can be used as a utility to map the ray deflection a curved surface. Processing the moire fringe patterns with the computer routines and using the simple equations based on geometric optics the ray deflections of a phase object can be plotted. It has been shown that large defects in the canopy that can be detected by the eye and smaller defects can both be detected by the computer.

By varying different filters and smoothing routines the fringes can be processed for various phase objects. For the canopy one section was shown where the sensitivity and the resolution were set. For other areas of the canopy the parameters may need to vary to get a different resolution for the fringe pattern. For different resolutions more or less smoothing may need to be done. Making routines adjustable was the desire for this project. Therefore, it is hard to give specific settings for a general phase object. By setting the variables for the canopy one may find difficulties in getting precise results for the entire canopy.

The goals now are to improve the precision of the process and to establish a computer routine that might fill in the gaps where the digitized image information is not complete or gets filtered out. This will take programming of a smart routine that will not only scan the next pixel but also read pixel values ahead and move in the weighted direction. Scaling of the ray deflection is needed to produce the focal map of the phase object. This ratio not only varies depending on the sensitivity and distance between gratings, but also depends on pixel width and smoothing.

When using moire techniques, the capabilities for adjustments are at least simpler, since there is less concern with the alignment of the system. The method also has the capabilities of adjustable sensitivity and resolution. These are definite advantages when measuring the defects in a curved phase object.

## BIBLIOGRAPHY

1. E. Hecht, and J. Feinberg, **Optics**, Addison-Wesley, Massachusetts,1987).

2. Z. Karny and O. Kafri, Appl. Opt. **21**, 3326 (1982).

3. O. Kafri and A. Livnat, Appl. Opt. **20**, 3098 (1981).

4. O. Kafri and I. Glatt, **The Physics of Moire Metrology** (Wiley, New York, 1990).

5. O. Kafri and A. Livnat, Opt. Eng. **24**, 150 (1985).

6. J. Krasinski, D.F. Heller, and O. Kafri, Appl, Opt. **24**, 3032 (1985).

7. O. Kafri, Opt. Lett. **5**, 555 (1980).

8. M. Born and E. Wolf, **Principles of Optics**, (Pergamon Press, 1970).

9. E. Keren and O. Kafri, J. Opt. Soc. Am. **2**, 111 (1985).

10. K. Gasvik, Appl. Opt. **22**, 3543 (1983).

APPENDIX

# COMPUTER ROUTINES

```
/*
        moresmob.c reads in a raw file and then extracts black lines by using a peak
                    filtering routine with a thirteen point smoothing routine
*/
#include <stdio.h>
#include <math.h>
 FILE *in_file, *out_file;
 int i,j,k,              /* counters             */
   N = 512,               /* number of rows       */
  m,                     /* number of lines to skip  */
  number,
  header_size = 512,        /* size of header       */
  row_len = 512;           /* number of samples in row */
 unsigned char
    in_buffer[512],
   out_buffer[512];
 char
   infil[81],             /* array to read in binary file */
  outfil[81];
 float x,y,z,
    smooth[512],
    sample[512],
    derivative[512]
; main()
 {
 banner:
      printf("**************************************************\n");
      printf("*  interferometric fringe processing progam          *\n");
      printf("*  last update Aug. 23, 1993                         *\n");
      printf("**************************************************\n");
      printf(" \n");
querry:                  /* prompt user for the file name to be used and the file name
                            to be created.                                    */
       printf("name of input file ?");
      scanf("%s", infil);
      printf("\n");
      printf("name of output file ?");
     scanf("%s", outfil);
      printf("\n");
 if ( (in_file = fopen(infil,"r")) == NULL){
  printf("YOU IDIOT!!! There's no such file!\n");
   return 1;
   }
 out_file = fopen(outfil,"w+");                 /*read in digitized image*/
 /* strip off first 512 bytes (header) */
fread((char *) in_buffer,sizeof(unsigned char), row_len, in_file);
```

```
  /* use 5 point smoothing filter, result is float */
  for(i=0; i <= N-1; i++){
    number = fread((char *) in_buffer,sizeof(unsigned char),row_len, in_file);
     for(j=6; j<= row_len-7; j++)
        smooth[j] = (in_buffer[j-6] + in_buffer[j-5] + in_buffer[j-1]
                 + in_buffer[j-3] + in_buffer[j-2] + in_buffer[j-1]
                  + in_buffer[j] + in_buffer[j+1] + in_buffer[j+2]
                  + in_buffer[j+3] + in_buffer[j-4] + in_buffer[j+5]
                  + in_buffer[j+6])/13.0;
        smooth[0] = smooth[1] = smooth[2] = smooth[3] = smooth[4] =smooth[5]
                  = smooth[6];
        smooth[row_len-1] = smooth[row_len-2] = smooth[row_len-3]
                    = smooth[row_len-4] = smooth[row_len-5]
                    = smooth[row_len-6] = smooth[row_len-7];
     for(j=0; j<= row_len-2; j++)
        derivative[j] = (smooth[j+1]-smooth[j]);
     derivative[row_len-1] = derivative[row_len-2] ;
      for(j=2; j<= row_len-3; j++)
      sample[j] = derivative[j-2] + derivative[j-1]+ derivative[j] +
          derivative[j+1] + derivative[j+2];
     sample[0] = sample[1] = sample[2];
     sample[row_len-1] = sample[row_len-2] = sample[row_len-3];
        for(j=1; j<= row_len-1; j++){
       out_buffer[j] = 255;
      /*if ( sample[j] > 0.0 && sample[j-1] < 0.0 )
         out_buffer[j] = 0; */
       if ( sample[j] < 0.0 && sample[j-1] > 0.0 )   /*check for valleys*/
         out_buffer[j] = 0;
        }
     out_buffer[0] = 255;
    fwrite((char *) out_buffer,sizeof(unsigned char), row_len, out_file);/*write info to
file*/
    }
  fclose(in_file);
   fclose(out_file);
} /* end of main*/


******************************************************************************
******************************************************************************


/*        moresmob.c reads in a raw file and then extracts black lines
                    using a thirteen point smoothing filter
*/
#include <stdio.h>
#include <math.h>
```

```
FILE *in_file, *out_file;
int i,j,k,               /* counters              */
  N = 512,                /* number of rows         */
  m,                     /* number of lines to skip  */
  number,
  header_size = 512,        /* size of header         */
  row_len = 512;           /* number of samples in row */
unsigned char
    in_buffer[512],
   out_buffer[512];
char
   infil[81],             /* array to read in binary file */
   outfil[81];
float x,y,z,
    smooth[512],
    sample[512],
    derivative[512]
; main()
 {
banner:
      printf("**************************************************\n");
      printf("*   interferometric fringe processing progam             *\n");
      printf("*   last update Sept.6, 1993                             *\n");
      printf("**************************************************\n");
      printf(" \n");
querry:
       printf("name of input file ?");
      scanf("%s", infil);
      printf("\n");
      printf("name of output file ?");
     scanf("%s", outfil);
      printf("\n");
 if ( (in_file = fopen(infil,"r")) == NULL){
  printf("YOU IDIOT!!! There's no such file!\n");
   return 1;
   }
 out_file = fopen(outfil,"w+");
 /* strip off first 512 bytes (header) */
fread((char *) in_buffer,sizeof(unsigned char), row_len, in_file);
/* use 5 point smoothing filter, result is float */
for(i=0; i <= N-1; i++){
  number = fread((char *) in_buffer,sizeof(unsigned char),row_len, in_file);
   for(j=6; j<= row_len-7; j++)
      smooth[j] = (in_buffer[j-6] + in_buffer[j-5] + in_buffer[j-1]
             + in_buffer[j-3] + in_buffer[j-2] + in_buffer[j-1]
              + in_buffer[j] + in_buffer[j+1] + in_buffer[j+2]
              + in_buffer[j+3] + in_buffer[j-4] + in_buffer[j+5]
              + in_buffer[j+6])/13.0;
     smooth[0] = smooth[1] = smooth[2] = smooth[3] = smooth[4] =smooth[5]
               = smooth[6];
     smooth[row_len-1] = smooth[row_len-2] = smooth[row_len-3]
                 = smooth[row_len-4] = smooth[row_len-5]
                 = smooth[row_len-6] = smooth[row_len-7];
    for(j=0; j<= row_len-2; j++)
      derivative[j] = (smooth[j+1]-smooth[j]);
    derivative[row_len-1] = derivative[row_len-2] ;
     for(j=2; j<= row_len-3; j++)
     sample[j] = derivative[j-2] + derivative[j-1]+ derivative[j] +
         derivative[j+1] + derivative[j+2];
    sample[0] = sample[1] = sample[2];
    sample[row_len-1] = sample[row_len-2] = sample[row_len-3];
       for(j=1; j<= row_len-1; j++){
      out_buffer[j] = 255;
     /*if ( sample[j] > 0.0 && sample[j-1] < 0.0 )
       out_buffer[j] = 0; */
      if ( sample[j] < 0.0 && sample[j-1] > 0.0 )  /* check for peaks in bright fringes*/
       out_buffer[j] = 0;
      }
    out_buffer[0] = 255;
   fwrite((char *) out_buffer,sizeof(unsigned char), row_len, out_file);
   }
 fclose(in_file);
  fclose(out_file);
} /* end of main*/


*****************************************************************************
*****************************************************************************


/*
grow2b.c -- grow the dots into a series of lines and store the lines This program takes the
choped image and generates a line file for each  individual line by connecting the data
points from the chopped image for an  output file of the following type 'line00' and will
increment by factor of 2.
 */

#include <stdio.h>
#include <math.h>
FILE *in_file, *out_file, *line_file;
int i,j,k,                  /* counters             */
   kk, jj,   radius,        /* max distance for search  */
   number,
    sample,
    line_count,
    row_size,
   column_size,
    x_pointer,
   past_x_pointer,
    past_y_pointer,
    y_pointer;
unsigned char
     frame[512][512],          /* work area for picture      */
    in_buffer[512],
    out_buffer[512];

 char
     infil[81],            /* array to read in binary file */
     linefil[81],           /* name of line output file     */
    outfil[81];            /* name of output file minus line */

main()
 {
 banner:
       printf(" \n");
       printf("**************************************************\n");
       printf("*   grow2b.c -- grow a series of line files                   *\n");
       printf("*   last update July 23, 1993                                 *\n");
       printf("**************************************************\n");
      printf(" \n");
 querry:
       printf("name of input file ?");
       scanf("%s", infil);
       printf("\n");
       printf("name of line file ?");
       scanf("%s", linefil);
       printf("\n");
       printf("size of search radius ?");
       scanf("%d", &radius);
       printf("\n");
       printf("\nlength of rows? ");
       scanf("%d", &row_size);
       printf("\nlength of columns? ");
       scanf("%d", &column_size);
       printf("row_size = %d\n",row_size);
       printf("colunm_size = %d\n",column_size);

 if ( (in_file = fopen(infil,"r")) == NULL){
   printf("YOU IDIOT!!! There's no such file!\n");
```

```
    return 1;
    }
  line_file = fopen(linefil,"w+");

  /* read into array frame */
  for(i=0; i<= column_size-1; i++){
    number = fread((char *) in_buffer,sizeof(unsigned char),row_size,          in_file);
    for(j=0; j<= row_size-1; j++)      frame[j][i] = in_buffer[j];
    }
  fclose(in_file);
  for(jj = 0; jj<=2; jj++){
   for(kk = 1;kk<=5;kk++){
    line_file = fopen(linefil,"w+");
    y_pointer = jj;
    x_pointer = radius;
    while(frame[x_pointer][y_pointer] == 255 && x_pointer < row_size-1){
      x_pointer++;
      }
    frame[x_pointer][y_pointer] = 255;
    fprintf(line_file,"%d   %d\n",x_pointer, y_pointer);    /* go to next scan line */
   past_y_pointer = y_pointer;
   do{      y_pointer++;
      past_x_pointer = x_pointer;
      x_pointer = x_pointer - radius;
      while(frame[x_pointer][y_pointer] == 255 && x_pointer < row_size-1){
        x_pointer++;
        }
      if ( x_pointer < past_x_pointer + radius && y_pointer < past_y_pointer + 5){
        frame[x_pointer][y_pointer] = 255;
        fprintf(line_file,"%d   %d\n",x_pointer, y_pointer);
        past_y_pointer = y_pointer;
        }
      else{
        x_pointer = past_x_pointer;
        }
      }
 while (x_pointer > radius && x_pointer < row_size -1 &&
            y_pointer < column_size);
   fclose(line_file);
    linefil[5]+=2;
   } /* end of kk loop */
  linefil[5] = linefil[5] - 10;  linefil[4]++;
  } /* end of jj loop */
  /*fclose(out_file); */
} /* end of main */
```

```
**************************************************************************
**************************************************************************

/*
grow2w.c -- grow the dots into a series of lines and store the lines This program takes the
choped image and generates a line file for each  individual line by connecting the data
points from the chopped image for an  output file of the following type 'line01' and will
increment by factor of two.
*/
#include <stdio.h>
 #include <math.h>
FILE *in_file, *out_file, *line_file;
int i,j,k,                    /* counters              */
   kk, jj,
   radius,                    /* max distance for search  */
   number,
   sample,
   line_count,
   row_size,
   column_size
   x_pointer,
   past_x_pointer,
   past_y_pointer,
   y_pointer;
 unsigned char
    frame[512][512],          /* work area for picture      */
    in_buffer[512],
   out_buffer[512];
 char
    infil[81],              /* array to read in binary file */
    linefil[81],            /* name of line output file     */
    outfil[81];             /* name of output file minus line */
main()
 {
 banner:
       printf(" \n");
       printf("****************************************************\n");
       printf("*   grow2w.c -- grow a series of line files             *\n");
       printf("*   last update Sept. 25, 1993                          *\n");
       printf("****************************************************\n");
       printf(" \n");
 querry:
         printf("name of input file ?");
       scanf("%s", infil);
       printf("\n");
       printf("name of line file ?");
```

```
        scanf("%s", linefil);
        printf("\n");
        printf("size of search radius ?");
        scanf("%d", &radius);
        printf("\n");
        printf("\nlength of rows? ");
        scanf("%d", &row_size);
        printf("\nnumber of columns? ");
        scanf("%d", &column_size);
        printf("row_size = %d\n",row_size);
        printf("colunm_size = %d\n",column_size);

  if ( (in_file = fopen(infil,"r")) == NULL){
    printf("YOU IDIOT!!! There's no such file!\n");
    return 1;
   }
  line_file = fopen(linefil,"w+");

  /* read into array frame */
  for(i=0; i<= column_size-1; i++){
   number = fread((char *) in_buffer,sizeof(unsigned char),row_size,
          in_file);
   for(j=0; j<= row_size-1; j++)
    frame[j][i] = in_buffer[j];
   }
 fclose(in_file);
  for(jj = 0; jj<=1; jj++){
  for(kk = 1;kk<=5;kk++){
   line_file = fopen(linefil,"w+");
   y_pointer = jj;
   x_pointer = radius;
   while(frame[x_pointer][y_pointer] == 255 && x_pointer < row_size-1){
     x_pointer++;
     }
   frame[x_pointer][y_pointer] = 255;
   fprintf(line_file,"%d   %d\n",x_pointer, y_pointer);
   /* go to next scan line */
   past_y_pointer = y_pointer;
   do{
     y_pointer++;
     past_x_pointer = x_pointer;
     x_pointer = x_pointer - radius;
     while(frame[x_pointer][y_pointer] == 255 && x_pointer < row_size-1){
       x_pointer++;
       }
     if ( x_pointer < past_x_pointer + radius && y_pointer < past_y_pointer+ 5){
        frame[x_pointer][y_pointer] = 255;
        fprintf(line_file,"%d   %d\n",x_pointer, y_pointer);
        past_y_pointer = y_pointer;
        }
      else{
       x_pointer = past_x_pointer;
        }
     } while (x_pointer > radius && x_pointer < row_size -1 &&
           y_pointer < column_size);
   fclose(line_file);
    linefil[5]+=2;
   } /* end of kk loop */
  linefil[5] = linefil[5] - 10;  linefil[4]++;
  } /* end of jj loop */
  /*fclose(out_file); */
 } /* end of main */

********************************************************************************
*
********************************************************************************


/*
chop.c -- clip out relavant portion of a binary image user is promted to for the file to be
          cut and at the diminsions desired.
*/
#include <stdio.h>
 #include <math.h>
FILE *in_file, *out_file;
int i,j,k,                  /* counters              */
   N = 512,                  /* number of rows          */
   row_info_size,
   column_info_start,
   column_info_size,
   row_info_start,
   number,
   header_size = 512,          /* size of header          */
   row_len = 512;            /* number of samples in row */
 unsigned char
    in_buffer[512],
   out_buffer[512];
 char
    infil[81],             /* array to read in binary file */
    outfil[81];

main()
  {
```

```
banner:
     printf(" \n");
     printf("***************************************************\n");
     printf("*  chop -- clips out portion of binary file         *\n");
     printf("*  last update June 29, 1993                        *\n");
     printf("***************************************************\n");
     printf(" \n");
querry:
      printf("name of input file ?");
     scanf("%s", infil);
     printf("\n");
     printf("name of output file ?");
     scanf("%s", outfil);
     printf("\n");
     printf("what is first scanline with information? ");
     scanf("%d", &row_info_start);
     printf("\nhow many scanlines contain information? ");
     scanf("%d", &row_info_size);
     printf("\nwhat is first column with information? ");
     scanf("%d", &column_info_start);
     printf("\nhow many scanlines contain information? ");
     scanf("%d", &column_info_size);
     printf("row_info_start = %d\n", row_info_start);
     printf("row_info_size = %d\n",row_info_size);
     printf("column_info_start = %d\n",column_info_start);
     printf("colunm_info_size = %d\n",column_info_size);

 if ( (in_file = fopen(infil,"r")) == NULL){
   printf("YOU IDIOT!!! There's no such file!\n");
   return 1;
   }
 out_file = fopen(outfil,"w+");
 /* strip off first 512 bytes (header) */
 for(i=0; i<=row_info_start-1; i++)
   fread((char *) in_buffer,sizeof(unsigned char), row_len, in_file);
 /* use 5 point smoothing filter, result is float */
 for(i=0; i <= row_info_size-1; i++){
   number = fread((char *) in_buffer,sizeof(unsigned char)row_len, in_file);
    k = 0;
    for(j=column_info_start-1; j<= column_info_size+column_info_start-1; j++){
     out_buffer[k] = in_buffer[j];
      k++;
      }
    fwrite((char *) out_buffer,sizeof(unsigned char), column_info_size;out_file);
    }
 fclose(in_file);
```

```
  fclose(out_file);
 } /* end of main */
******************************************************************************
******************************************************************************


/*  smoothline.c - takes lines from line00 -- line19 files and smooths them out
                    to line40 -- line 59 files using a twenty on point filter
*/
 #include<stdio.h>
#include<math.h>
FILE *in_file, *out_file;
 int ii,i,j,k,
   counter = 0,
   x,y,
   ch = 0;
 unsigned char
   in_buffer[512],
   out_buffer[512];
 char
  bogus[81],
   infil[81] = "line00",
   outfil[81] = "line40";
 int
  sample,
   line[512][2];
 main()
 {
  for(j=0;j<=1;j++){  for(i=0; i<=9; i++){
   counter = 0;
   printf("file%s\n",infil);
   printf("number of lines read = %d\n",counter);
   in_file = fopen(infil,"r");
  k = 0;
   do{
     fscanf(in_file, "%d    %d",&y,&x);
      line[k][0] = x;
      line[k][1] = y;
      k++;
      counter++;
     } while ((ch = getc(in_file)) != EOF);
   printf("end of file reached\n");
   printf("number of lines read = %d\n", counter);
   if (counter > 20) {
   out_file = fopen(outfil,"w+");
     for(ii=11;ii<=counter-19;ii++){
            sample = line[ii-1][1]+line[ii+1][1]+line[ii][1];
          sample = sample+line[ii-2][1]+line[ii+2][1];
          sample = sample+line[ii-3][1]+line[ii+3][1];
          sample = sample+line[ii-4][1]+line[ii+4][1];
          sample = sample+line[ii-5][1]+line[ii+5][1];
          sample = sample+line[ii-6][1]+line[ii+6][1];
          sample = sample+line[ii-7][1]+line[ii+7][1];
          sample = sample+line[ii-8][1]+line[ii+8][1];
          sample = sample+line[ii-9][1]+line[ii+9][1];
          sample = sample+line[ii-10][1]+line[ii+10][1];
          sample = sample/21.0;
          fprintf(out_file,"%d     %d\n",sample,line[ii][0]);
          }
      fclose(out_file);
      outfil[5] =outfil[5] +1;
      }
      /*printf("press 'j'");
      scanf("%s",bogus);
  */   fclose(in_file);
      infil[5] = infil[5] + 1;
        } /*end of reading files loop */
      infil[5] = infil[5] -10;
      infil[4] = infil[4] +1;
      outfil[5] = outfil[5] - 10;
      outfil[4]=outfil[4] + 1;
      } /* end of j loop */
  } /* end of main */

*****************************************************************************
*****************************************************************************
**

/*
  rtemp.c-- set the bent line in an array then calculates the slope by
            line regression.  Calculates the local slope and places them into a file
            so the user can plot the localized slope map of all files.
 */

#include<stdio.h>
#include<math.h>
FILE *in1_file,
      *out_file;
int i,j,jj,k,kk,s,t,size=5,segm=25,counter=0;
float value[350][2],sslop=30,slo[350],a1[350],tana,d=70.0,theta=.0565,focal,
                sumx,sumy,sumxx,sumxy,x,y,ch=0.0;
unsigned char
```

```
  in1_buffer[390],
  out_buffer[390];
char
  infil1[261] = "line40",
  outfil[261] = "reg340";
main()
{
 for(s=0;s<=1;s++){
 for(t=0; t<=9; t++){
  counter = 0;
 if((in1_file = fopen(infil1,"r"))==NULL){
   printf("no such displaced file\n");
    return 1;
   }
  k = 0;
  do{
    fscanf(in1_file,"%f  %f",&x,&y);
    value[k][1]=x;
    value[k][2]=y;
    k++;
    counter++;
  } while ((ch = getc(in1_file)) != EOF);
    printf("end of file %d reached\n",counter);
    printf ("%d\n",counter);
    if((out_file = fopen(outfil,"w+"))==NULL){
   printf("no such output file\n");
   return 1;
  }
   for(j=0;j<=counter-segm-1;j++){
     slo[j+segm/2] = (value[j+segm-1][1]-value[j][1])/
           (value[j+segm-1][2]-value[j][2]);
      }
  for(j=segm/2;j<=counter-segm-1;j++){
     for (i=1; i<=segm;i++){
         a1[j] = a1[j] + slo[j+i];
      }
     a1[j] = a1[j]/segm;
      tana=(sslop-1/a1[j])/(1+sslop*a1[j]);
      focal=d/(theta*tana);
        fprintf(out_file,"%f\n",a1[j]);
   }
  fclose(out_file);
 outfil[5] =outfil[5] +1;
  close(in1_file);
 infil1[5] = infil1[5] + 1;
 }
```

```
  infil1[5] = infil1[5] -10;
  infil1[4] = infil1[4] +1;
  outfil[5] = outfil[5] - 10;
  outfil[4]=outfil[4] + 1;
  } /* end of j loop */
} /* end of main */
```

VITA

CARL B. LOPEZ

Candidate for the Degree of

Master of Science

Thesis: APPLICATION OF MOIRE TECHNIQUE AND QUALITY CONTROL OF A TRANSPARENT OBJECT

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Edmond, Oklahoma, November 11, 1967, the son of William and Glenda Lopez, and Married to Denise Layne Stenner August 7, 1993.

Education: Received Bachelor of Science Degree in Physics from Oklahoma Baptist University, Shawnee, Oklahoma, May, 1990; completed the requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, December, 1993.