Data Processing in Nanoscale Profilometry

by

Cheng-Jung Chiu

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995
[June 1995]

Author .....
Department of Mechanical Engineering
May 12, 1995

Certified by .....
Dr. Kamal Youcef-Toumi
Associate Professor
Thesis Supervisor

Accepted by .....
Dr. Ain A. Sonin
Chairman, Departmental Committee on Graduate Students

Data Processing in Nanoscale Profilometry

by

Cheng-Jung Chiu

Submitted to the Department of Mechanical Engineering
on May 12, 1995, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

New developments on the nanoscale are taking place rapidly in many fields. Instrumentation used to measure and understand the geometry and property of the small scale structure is therefore essential. One of the most promising devices to head the measurement science into the nanoscale is the scanning probe microscope. A prototype of a nanoscale profilometer based on the scanning probe microscope has been built in the Laboratory for Manufacturing and Productivity at MIT. A sample is placed on a precision flip stage and different sides of the sample are scanned under the SPM to acquire its separate surface topography. To reconstruct the original three dimensional profile, many techniques like digital filtering, edge identification, and image matching are investigated and implemented in the computer programs to post process the data, and with greater emphasis placed on the nanoscale application. The important programming issues are addressed, too. Finally, this system's error sources are discussed and analyzed.

Thesis Supervisor: Dr. Kamal Youcef-Toumi
Title: Associate Professor

Acknowledgments

At the outset, my gratitute goes to my thesis advisor, Professor Kamal Youcef-Toumi, for his guidance, patience, and kindness. Thank him for giving me the chance to work on this project, help me learned so many things here. Also thank Dr. Eric Liu's guidance and advices in this project. The discussions with him provides me many, many ideas.

I would like to thank my collaborator Tarzen Kwok, for his tireless input and assistance. He not only supported and helped me in many parts of this project through the end of the thesis, his working altitude also lets me realize how to do research.

I also want to thank T.J. for his generous advices. He is always willing to listen to my problems and give me the appropriate suggestions, both in the research and in my personal life. I'm glad I can work with him in the same laboratory.

I also thank all the colleagues and friends I made here. Mingchih, Ilin, Jane, Mingyi, Cynthia, and many, many more. They mades my life at MIT has been more productive and enjoyable.

Finally, I have to thank my family and say sorry that I couldn't accompany them in the past two years. Thank my sister Ya-Ting and my brother Cheng-Wei for the love and cheer they shared with me. Most importantly, I would like to thank my parents, who give me their unconditional support, endless love and care. They made me know the joy of life. Their loving always encourages me while I'm down. If I could have any achievement, that is all attributed to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

New developments on nanoscale are taking place in many fields. Therefore, the instrumentation to understand and measure the geometry and properties of the small scale structure to nanometric resolutions are essential. One of the most promising devices to lead the measurement science into the nanoscale precision is the Scanning Probe Microscope. Scanning Probe Microscope (SPM) was first invented in the early 1980's. The extremely high resolution it provides (up to a few anstroms) made it be used widely in a variety of disciplines, including surface science, microstructure study, biomedical analysis, ... etc. Its high resolution ability along with scanning range up to 100 μm also makes it suitable as a "micro coordinate measuring machine".

An atomic-force-microscope-based nano-precision profilometer has been designed and implemented at the Laboratory for Manufacturing and Productivity at Massachusetts Institute of Technology [1]. The device is primarily used to measure the profile of the sample tip area. The sample is kinematically clamped to the reference surface of the sample jig (flip stage) under the AFM scanner. After one side measurement is finished, the sample is flipped over by rotating the high precision flip stage to measure the other side. Using the edge identification and image matching techniques.

[1] See the thesis by Tarzen Kwok[7]

the separate parts of the sample surface topography could be reconnected and the original three dimensional sample profile can be reconstructed.

All the styli-type profiliometers are suffered from the measurement distortion caused by the convolution between the tip shape and the measured surface, especially if the surface has steep slope or sharp edge. This situation occurs in the scanning of this sample's tip area. Based on David Keller's deconvolutoin method [5], the convolution problem can be reduced.

1.2 Thesis Contents

The thesis is organized as follows. Chapter 2 describes the basic configuration of the profilometer, it includes X-Y stages, flip stage and AFM machine. A brief operation procedure to measure a sample is given here, too.

The post-processing issues are addressed in Chapter 3. To recover the original sample profile from the noisy, distorted, and separated two side data, technique such as discrete filtering, deconvolution, edge identification, image matching and coordinate transformation are adopted.

Chapter 4 described some essential issues about software implementation. The data processing software are currently developed under MS-Windows environment to provide the graphical user interface. A brief operation procedure for this program is introduced here, too.

Chapter 5 is focused on the system's performance, or errors analysis. The system's error sources , including the AFM, positioning and flip stages, are reviewed. Some errors can be avoided or reduced by software or hardware methods, others are difficult to eliminate at least in this machine. They are all evaluated in this chapter.

Finally, the conclusion and suggestion are provided in the last chapter.

Chapter 2

System Configuration and Operation

2.1 introduction

The whole system used to measure the sample profile could be decomposed into three parts : the commercial AFM machine (from the Park Scientific Instruments, XL), the custom sample positioning system, this includes a sample flip stage and X-Y positioning stages, and post-processing software. This chapter will briefly describe the first two parts, or the hardware configuration. [1] Also, the measuring procedure will be covered here. Topics about the post processing issues (software part), like filtering, reconstruction, ...etc are addressed in the next chapter.

2.2 System Description

The outline of the customized AFM machine is shown in Figure 2-1 . It consists a commercial AFM machine, X-Y stages and a flip stage. The coordinate system is defined as follows : the X axis is parallel to the reference mounting surface (i.e. the granite table) and pointing into the sample, the Y axis is on the centerline of rotation

[1] A detailed design procedure and consideration can be found in [7]

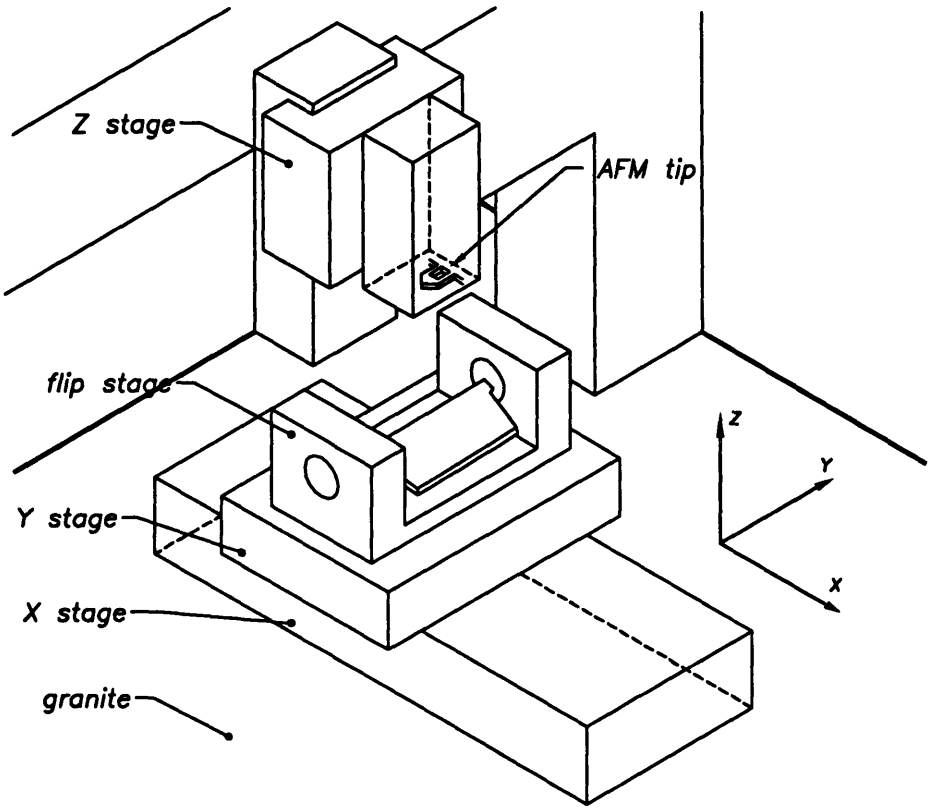


Figure 2-1: Overall System Configuration

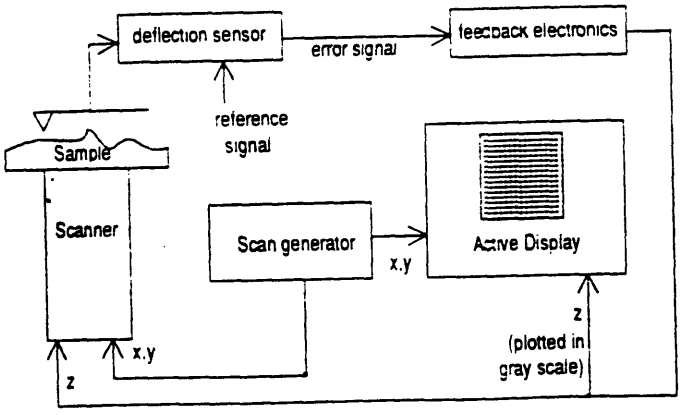


Figure 2-2: Schematic of a Generalized Atomic Force Microscope

(i.e. along the sample edge), and the Z axis is normal to the granite surface.

2.2.1 AFM machine

Basically, in the contact mode atomic force microscope, a cantilever beam mounted microstylus is moved relative to the sample surface using piezoactuators, (see Figure 2-2); and the deflection of the cantilever is taken to be a measurement of the surface topography. The AFM used throughout this thesis work is a commercial machine from the Park Scientific Instruments. It can provide a scanning range up to 100×100 μm in lateral and 8 μm in vertical (the Z direction), also it has a closed-loop/independent sensor monitoring of piezoelectric scanner motion to eliminate the errors from piezoelectric scanner's intrinsic nonlinear behavior. This makes it a suitable choice as a miniature CMM. Also, it comes up with a high magnitude CCD camera which makes the automated sample positioning possible.

2.2.2 Sample Positioning System

The sample positioning system is composed of X-Y stages and a flip stage.

The X-Y stages provides the required resolution to locate the sample at the desired measuring position in X and Y direction [2] while providing the required mechanical stability to get the nanometer level measurement. The stage also provides the long range movement of the sample for ease of loading.

A close-up look of the flip stage is shown in Figure 2-3. [7] The flip stage has two main functions : one is to hold the sample and provide a reference plane for the sample, the other is to provide the required high precision and high repeatable rotary movement for measuring the two sides of a sample. The sample is clamped against a reference surface plane by a clip and aligned to the axis of rotation (the line connecting the centers of two locator balls) by two precision stops. The axis of rotation is defined by the X-Y-Z and roll/yaw stops on the bases and two locator balls

[2] the coarse positioning in Z direction is accomplished by a stepping motor which moves the scanner

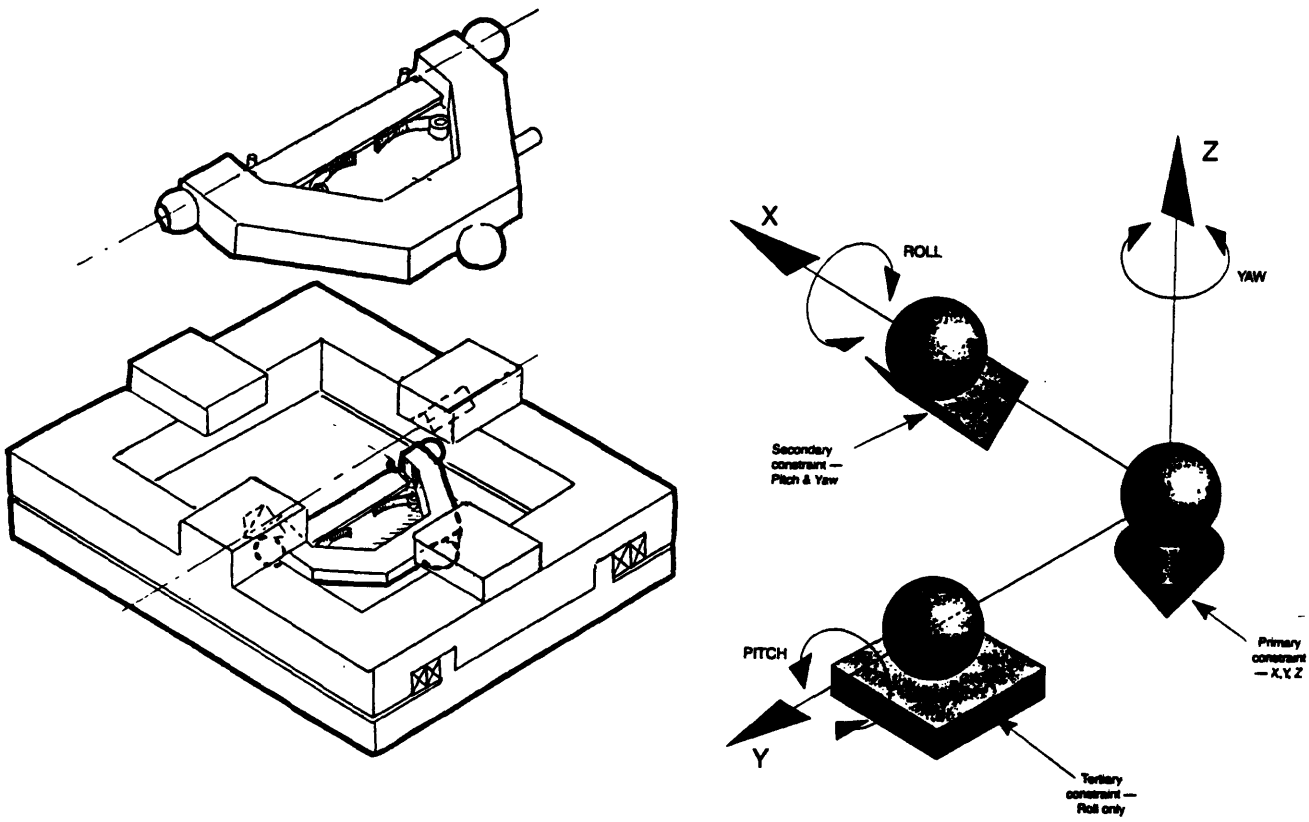


Figure 2-3: Flip Stage

on the sample holder. To position at the tilt angle, the sample holder is rotated to the near vicinity of the angle stop by a geared stepping motor; once there (i.e. position sensed by a limit switch) power to the motor is shut off and the holder is magnetically clamped against the angle stop. The X-Y-Z stop, constructed out of three balls, the yaw/roll angle stops, consisting of two rods that form a 'V' two point constraint, and the axis locator balls on the sample holder form a kinematic mechanism to provide the required high repeatability. To unclamp, the motor is reenergized and made to turn in the opposite direction.

2.3 Operation Procedure

The essential profiling procedure consists of the following operations :

1. Set up the AFM and load the sample;

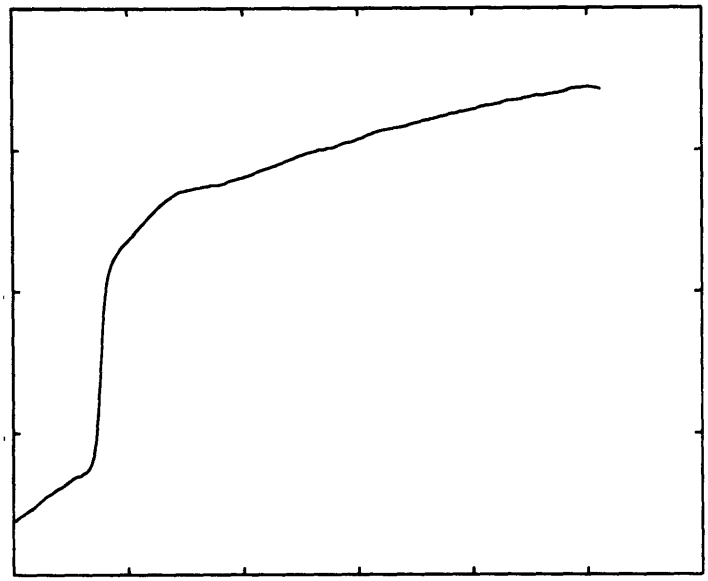


Figure 2-4: Typical Scanning Trace

2. Visually align the sample ultimate edge with the probe tip (in the X direction) and lower the AFM probe onto the sample surface, then use the AFM piezoscanner for lateral alignment with the ultimate tip; a single trace line should show a step-like feature, it means that the profile contains data both before and after the scanning tip falls off the sample edge. A typical sample profiling seen on the digital oscilloscope on the computer screen is shown in Figure 2-4.

3. Appropriately setting the AFM measurement parameters; two important things not usually seen in the surface topography measurement are pointed out. One is the scanning direction. Scanning direction should be from the sample body to the sample edge, this can reduce the piezoscanner's creeping error, also in this way, the transient error from the surface changes (like a step response) happened after the scanning tip has fallen off the sample's ultimate tip, and those data are not important for profile reconstruction later [3]. Another thing to notice is to turn off "autoslope". In the usual surface topography measurement, the data measured will be fitted a first order plane to remove the surface tilting component from the stages and the scanner (autoslope). But this is not the

[3]see chapter 3

case here. The data from two sides of the sample should be measured with respect to the same reference coordinate system, so the "autoslope" should be turned off.

4. Flip the sample to the other side and repeat steps 2 and 3

5. Post process the measurement data[4]

. One thing worthy to mention is, the scanning direction is set in the X direction while the sample's edge is parallel to the Y direction. These arrangements have the following benefits :

1. Because the cantilever is mounted parallel to the Y direction and the scanning tip is on the center line of the cantilever, it is easier to align the sample by looking at the center line of the cantilever if the sample edge is parallel to the Y direction instead of the X direction.

2. Because the scanning tip is tilted at a certain angle in the Y-Z plane, it will cause a different level of convolution problems due to the asymmetry of the scanning tip if the different scanning direction is taken [5]. (for example., if we scan a step sample from $-Y$ to $+Y$ and then from $+Y$ to $-Y$, because of the asymmetry of the tip's sidewall to the Z direction, the slope gradient at the step area will be different as shown in Figure 2-5). To reduce this problem, the scanning direction should be set in the X direction.

2.4 Summary

In this chapter, the hardware configuration of the profilometer was described. Some design features and consideration in this instrument were included. The operation procedure to measure a sample was also addressed. Again, the detailed design process for this instrument can be found in Kwok[7].

[4]See chapter 4

[5]see chapter 3 for the convolution problem between the stylus and the sample

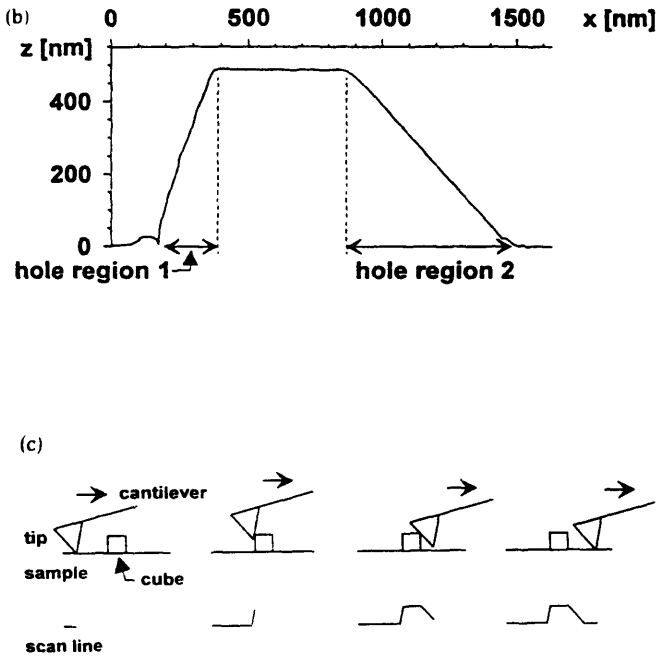


Figure 2-5: Different Profiling Results Due to the Tip's Orientation [16]

Chapter 3

Data Acquisition and Processing

3.1 Introduction

The data gathered from the profilometer represents only a sample's two separate surface topography. The critical issue is to transform these surface topography into the original sample profile. A typical profiling trace from the AFM is shown in Figure 2-5. This curve shows how a scanning tip moves along a sample surface. It contains information not only the sample profile but also the interaction between the scanning tip and the sample. It is also mixed with some irrelevant data. (i.e.: the data after the tip has fallen off the sample's ultimate edge.) Besides, the data is also contaminated by noise and disturbance. Unless the falling point can be identified first and the sample profile can be extracted from the noise-contaminated, extra-information-mixed data, the true three dimensional profile is hard to be established.

One thing to note is, the rotation of the flip stage will not only flip the sample over, it will also introduce some errors like the shifting along the sample edge. As shown in Figure 3-1, by comparing the edge features of the two images, we can see that the images are not matched very well, there is shifting along the edge direction and the left image is lower than the right image. This error needs to be effectively compensated by use of image matching technique before putting the two sides images together.

A feasible procedure to reconstruct the original sample profile has been proposed,

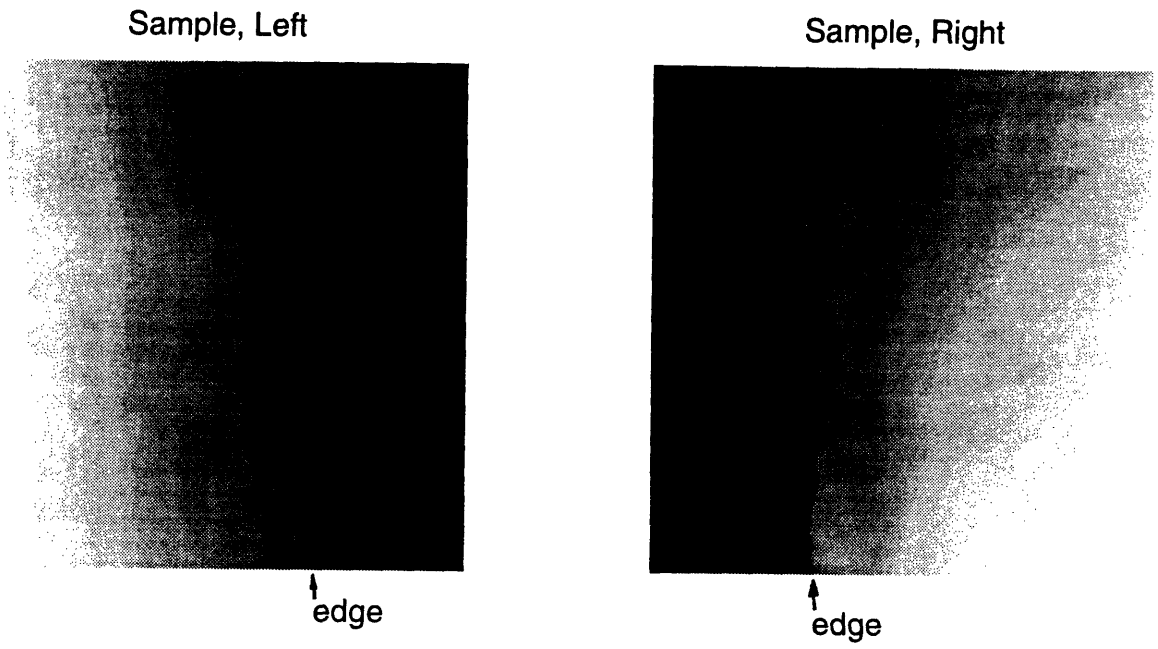


Figure 3-1: Image Shifting

implemented and tested here. First, the data is passed through a discrete filter to effectively reduce the noise. To eliminate the error from the finite size of the scanning tip (convolution problem between the tip and the sample edge), the "envelope image analysis" method developed by David Keller[5] is adopted to deconvolute the probe shape from the scanning profile. Then an edge identification technique is used to find the sample edge. At the assumption of the sample edges identified from the two sides images are the same, by use of coordinate transformation, the original three dimensional sample profile can be established.

Here is the list of the essential procedures :

1. discrete filtering of the data.

2. reconstructing the data by use of the "envelop image analysis" method.

3. identifying the sample edges of the two images.

4. image matching to correct the data shifting.

5. coordinate transformation to establish the whole profile.

The detailed procedures are described in the following sections.

3.2 Filtering (Noise Removal)

Basically, the scanning data from the AFM can be looked as a digital image. Noise can corrupt the data or image analysis in many ways, especially in edge detection of a image, a discrete approximation of first or second derivatives of the image data are commonly needed, and the noise can seriously distort the edge identified since differentiation will amplify those high frequency noise. Examples of additive noise degradation here could be electronic circuit noise, environment noise from acoustic or thermal effects, and even amplitude quantization noise. Basically, those noise can be taken as random and vuncorrelated.

An image degraded by additive random noise can be modeled by

$$g(n_1, n_2) = f(n_1, n_2) + v(n_1, n_2) \tag{3.1}$$

where $f(n_1, n_2)$ is the original, ideal image, $v(n_1, n_2)$ represents the signal independent, additive random noise and $g(n_1, n_2)$ represents the corrupted image. n_1 and n_2 are the indices of the pixel.

If every $v(n_1, n_2)$ is uncorrelated with zero-mean, then by scanning the sample's same area many times and averaging those images up, the noise could be reduced. Unfortunately, due to the time constraint and other problems like thermal drifting and the wearing of the scanning tip, this is infeasible.

To effectively reduce the noise, A few smoothing techniques commonly used in image processing have been tested and compared. [1]

[1] A good reference of noise removal in image analysis could be found in Lim[10].

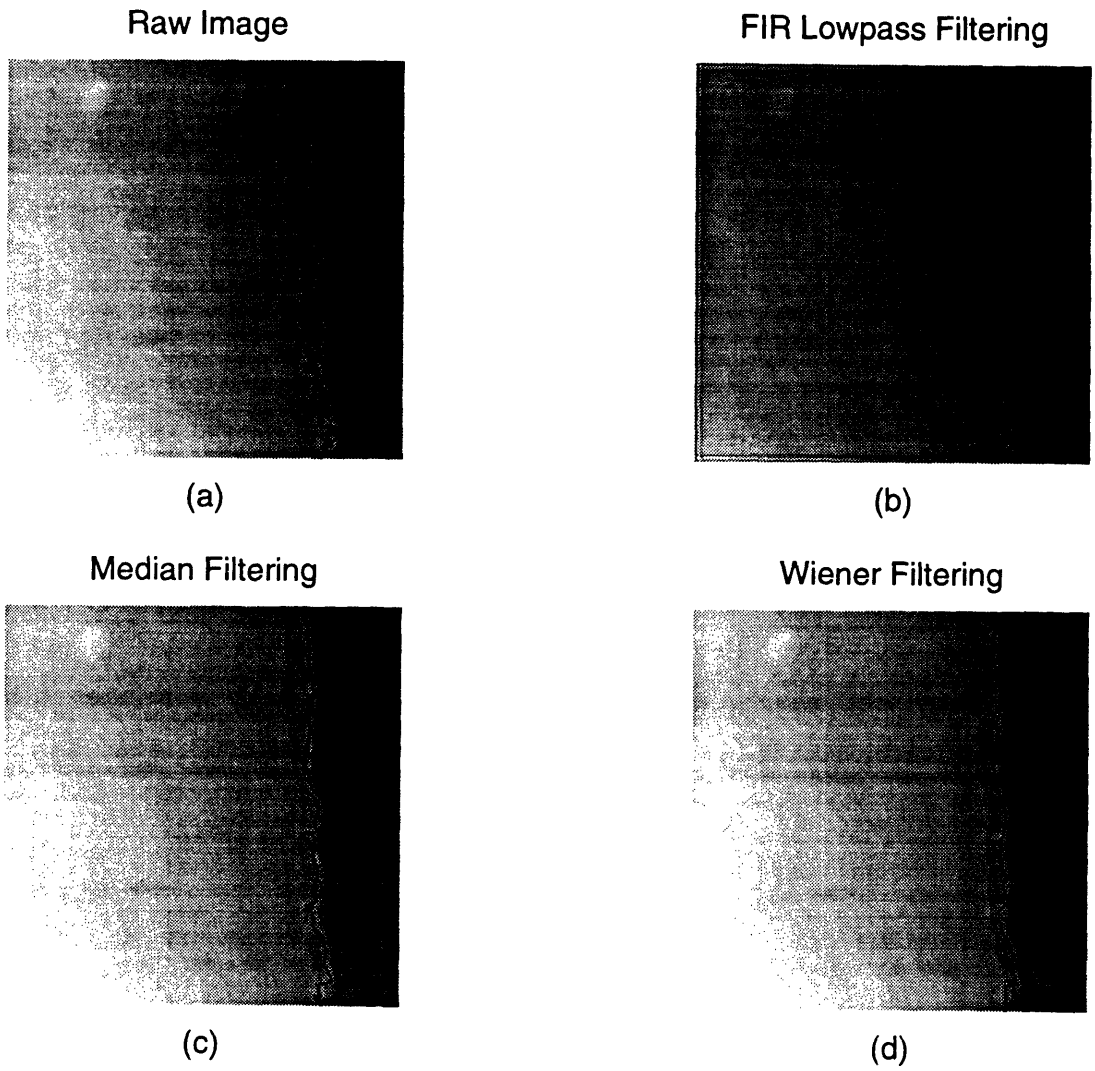


Figure 3-2: Filtered Image and the Identified Edge : (a) Raw Image (b) FIR Lowpass Filtering (c) Median Filtering (d) Wiener Filtering

3.2.1 Lowpass Filtering

Since the energy of a typical image noise contains higher frequency component compared to the true image, by reducing the high frequency components while preserving the low-frequency components, lowpass filtering reduces a large amount of noise at the expense of reducing a small amount of signal. As shown in Figure 3-2 (a) and (b), compared to the unfiltered image, lowpass filtering does reduce the additive noise, and smooth the "wiggle" of the identified sample edge, [2] but at the same time it blurs (distorts) the image. Blurring is a primary limitation of the lowpass filtering, especially at the border of the image or the area with significant changes. Experiments found that, compared to the other filtering methods (median and Adaptive Wiener filtering, see below), lowpass filtering causes more distortions at the identified edge points.

3.2.2 Median Filtering

Median filtering is a nonlinear process useful in reducing impulsive or salt-and-pepper noise. It is also useful in preserving edge in an image while reducing random noise.

In a median filter, a window slides along the image, and the median intensity value of the pixels within the window becomes the output intensity of the pixel being processed. For example, suppose the pixel values within a window are 5, 6, 55, 10, and 15, and the pixel being processed has a value 55. The output of the median filter at the current pixel location is 10, which is the median of the five values. Like lowpass filtering, median filtering smoothes the image, but it can preserve discontinuities in a step function and smooth a few pixel whose values differ significantly from their surroundings without affecting the other pixels.[10]

To better preserve 2D step discontinuities, the separable median filtering method are adopted, that is, the 2-D signal is filtered in one direction with 1-D median filter first, then filtered in the perpendicular direction. The result of separable median filtering depends on the order in which the 1-D horizontal and vertical median filters

[2] See Section 3.4 for the sample edge detection

are applied. In the sample scanning data, since the horizontal direction is taken as the raster scanning direction, the noise of the adjacent pixels in the vertical direction can be looked as uncorrelated, so it is preferable to filter in the vertical direction to remove the most significant noise first, then filter in the horizontal direction. Figure 3-2 (c) shows the filtered image and the edge detected.

3.2.3 Adaptive Wiener Filtering

In Equation (3.1), if signal $f(n_1, n_2)$ and the noise $v(n_1, n_2)$ are samples of zero-mean stationary random processes that are linearly independent of each other and their power spectra $P_f(w_1, w_2)$ and $P_v(w_1, w_2)$ are known, the minimum mean square errors estimate of $f(n_1, n_2)$ is obtained by filtering $g(n_1, n_2)$ with a Wiener filter whose frequency response $H(w_1, w_2)$ is given by

$$H(w_1, w_2) = \frac{P_f(w_1, w_2)}{P_f(w_1, w_2) + P_v(w_1, w_2)} \tag{3.2}$$

If $f(n_1, n_2)$ has a mean of m_f and $v(n_1, n_2)$ has a mean of m_v, then m_f and m_v are first subtracted from the degraded image $g(n_1, n_2)$. The resulting signal $g(n_1, n_2) - (m_f + m_v)$ is next filtered by the Wiener filter. The signal mean m_f is then added to the filtered signal.

Basically, the Wiener filter preserves the high SNR frequency components while attenuating the low SNR frequency components. Since the noise is generally wideband compared to the signal, the Wiener filter's lowpass characteristics can effectively reduce the noise. But, just like the usual lowpass filtering, Wiener filter blurs the image significantly. One reason is that a fixed filter is used throughout the entire image. In a typical image, image characteristics differ considerably from one region to another. Degradation may also vary from one region to another. It is reasonable, then, to adapt the processing to the changing characteristics of the image and degradation.

One adaptive Wiener filter developed by Lee [8] has the form below :

$$p(n_1, n_2) = m_f(n_1, n_2) + \frac{\sigma_f^2}{\sigma_f^2 + \sigma_v^2}(g(n_1, n_2) - m_f(n_1, n_2)) \tag{3.3}$$

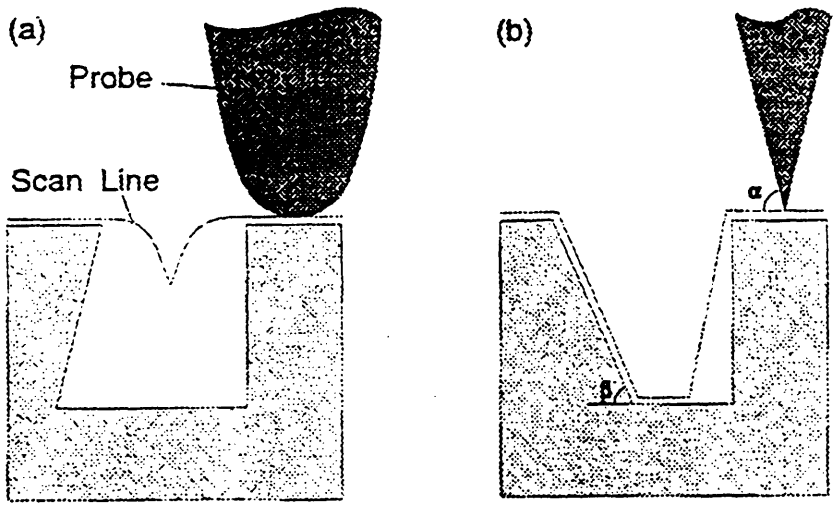


Figure 3-3: Examples of Influence of Tip Shape to Apparent Surface Profile [4]

where $p(n_1, n_2)$ is the processed image, m_f and σ_f are the local mean and standard deviation of $f(n_1, n_2)$ and are updated at each pixel, the additive noise $v(n_1, n_2)$ is assumed to be zero mean and white with variance of σ_v^2. Again, to further preserve the edge, a cascade of 1-D adaptive Wiener filtering can be applied. Figure 3-2(d) shows the filtered image and the edge detected. First, the 1-D filter is oriented in the same direction as the edge, the edge can be avoided and the image can be filtered along it. Then another 1-D filter that crosses the edge is applied.

Comparing the curves of the detected sample edges in Figure 3-2 (a)-(d), we found that filtering does effectively remove the noise and is helpful for further analysis. To this particular application, the adaptive Wiener filtering shows better performance over the other filtering methods tested.

3.3 Data Reconstruction (Deconvolution)

One difficulty in probe metrology arises when a surface has regions with steep slopes. How faithfully a probe microscope shows the surface topography depends strongly on the size and shape of the probe. Ideally, the scanning tip should have a impulse-like shape, or a tip which is much sharper and its sidewalls are much steeper than any feature in the sample, then the resulting image will closely follows the actual sample surface. But in practice, due to manufacturing issues, also taking into account the

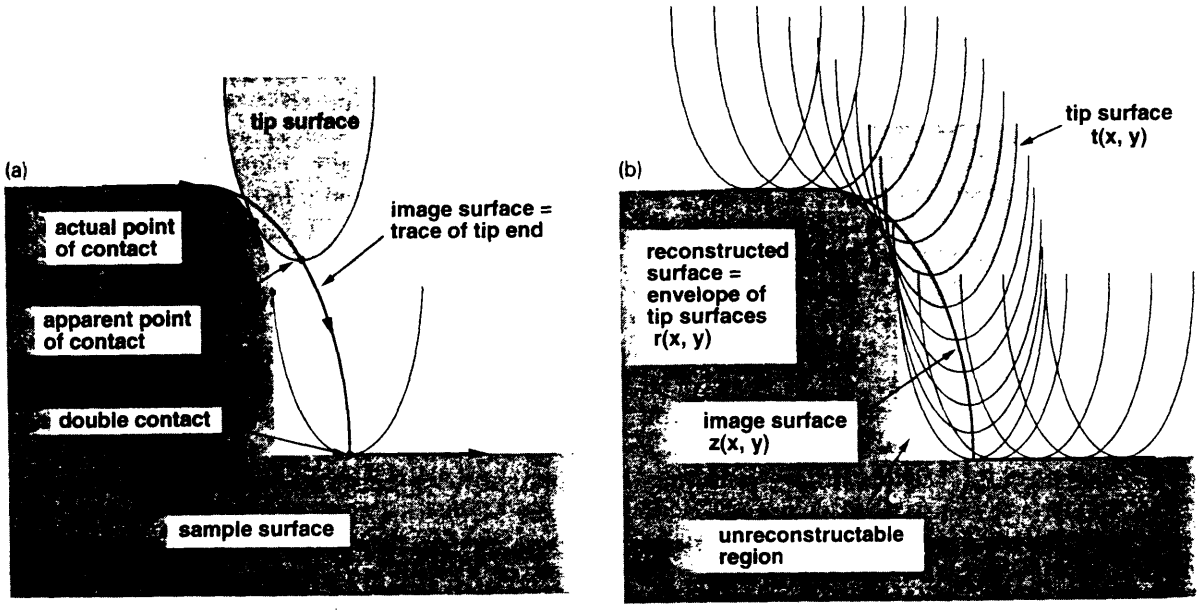


Figure 3-4: Envelop Image Analysis[5]

stiffness of the scanning tip, the tip is seldom ideal. Then some features may be completely inaccessible while others will be distorted. Figure 3-3 shows how a tip shape might effect the output profile. The distortion comes from the relative motion of the tip and the sample while scanning, which strongly depends on the shape of the tip and usually is a nonlinear process. [3] Unless the size and shape of the probe are accurately known, the distortion cannot be corrected. One way to acquire the actual tip shape is to scan the tip under another microscope, for example, SEM. Another feasible way is to scan a particular, well-defined sample with shape known beforehand [5]. Once the tip shape is acquired, there are a few methods available to reduce this problem. (e.g., Keller et al. [5][6], Lee et al.[9].) One effective method is the "envelope image analysis" method proposed by Keller[5] to reconstruct the distorted image. The envelope image analysis method computes the reconstructed sample surface as the envelop of a large set of probe tip surface functions. If the sample is considered to be beneath the tip, so the tip surface is assumed to be downward-pointing and to have a well defined minimum, which will be called the end point of the tip. By

[3] Although the process of extracting the true profile from the distorted data are usually referred to "deconvolution", actually, care must be taken. Since "convolution" is a linear process, but here, it is highly non-linear in the common case. Deconvolution techniques like Fourier Transformation cannot be used here.

placing the tip surface on each point of the image surface, as shown in Figure 3-4, the reconstructed sample surface is the envelop formed by all of such tip surface. The advantages of this method over the other methods is that no particular tip shape needed (e.g., Lee et al. [9]) and less noise-sensitive compared to some other methods. (e.g., Niedermann et al. [11], Reiss [15].)

As we can see from Figure 3-4, there are certain regions ("unreconstructable" regions) that can not be reconstructed no matter how accurate the data may be. The region between the contact points is never touched by the probe, so the image contains no informations about it. And in this method, they are filled with a segment of the tip surface. This segment needs to be identified so the sample profile can be extracted there (see section 3.4).

To implement the above method into a computer program, a mathematical description is needed. Consider an image surface $z(x, y)$, defined in a region R of the xy plane. Let the surface of the probe tip be given by a function $t(x, y)$, usually, it is convenient to define the minimum point of the tip shape as the origin. Let (x, y) and (x', y') be two points in the xy plane, and define a function $w(x, y; x', y') = z(x', y') + t(x - x', y - y')$. Here $w(x, y; x', y')$ is effectively a tip function with its axis located at (x', y') and with its end point raised to a height $z(x', y')$. Then the reconstructed surface, $r(x, y)$, is the minimum of all functions $w(x, y; x', y')$ over all (x', y') in R, so that

$$r(x, y) = \min z(x', y') + t(x - x', y - y'), all\ (x', y')\ in\ R \tag{3.4}$$

for each point (x, y) in R. For this expression to be defined, $w(x, y; x', y')$ must have a minimum, i.e., both $z(x, y)$ and $t(x, y)$ must have lower bounds.[5]

One common problem of all the deconvolution methods is, either the tip or the sample's true profile must be known beforehand. Here, the tip shape must be accurately measured or estimated if high accuracy is needed. But, even in the same manufacturing batch, the tips are different from each other, also it takes time to acquire the accurate tip profile. Other factors like the tip's tilt angle will also effect

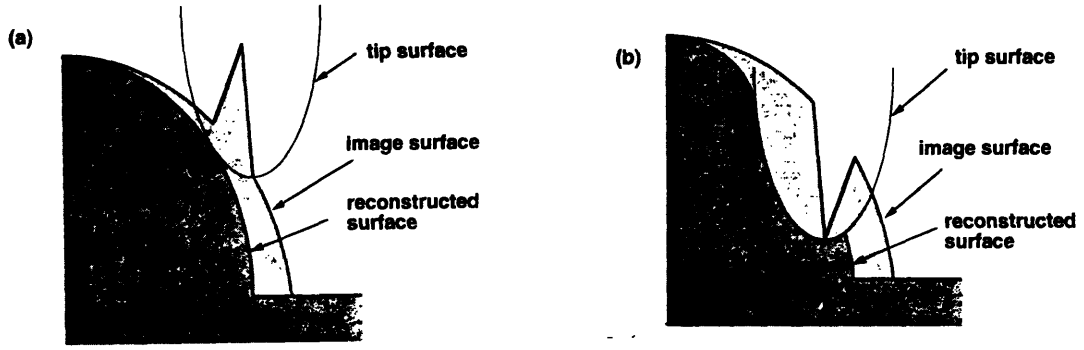


Figure 3-5: Envelop Image Analysis with Noise[5]

the convolution result, and this error will cause a sine error or the so called Abbé principle.

Another problem comes from the image noise. Although the envelop image analysis method can effectively remove a sharp, upward fluctuations (Figure 3-5 (a)), but for a downward fluctuation, it will produce a significant distortion in the reconstructed surface (see Figure 3-5 (b)).

To eliminate these problems, the envelop image analysis method should be applied only to the small ultimate tip area. Basically, the envelop image analysis method reconstructs the data from a scanning tip's downward convex shape part. For a tip with upward convex shape or just a straight line, the envelop image analysis method can not provide more information. From Figure 3-6, we can see that, unlike a tip with downward convex shape, for a ideal tip with only upward convex shape, the scanning profile and the reconstructed profile are exactly the same. Most SPM tips are parabolic-like only quite near the ultimate tip end. Further up, the tip surface usually goes over to an approximately downward convex conical shape or even a straight line. So, only the ultimate tip area will actually effect the reconstructed profile.

The advantage of restricting this method only to the ultimate tip area is that only the estimation or measurement of the ultimate tip shape is needed, and the deconvolution errors due to inappropriate tip information can be minimized. Also the deconvolution errors from the image noise can be reduced in some cases, too.

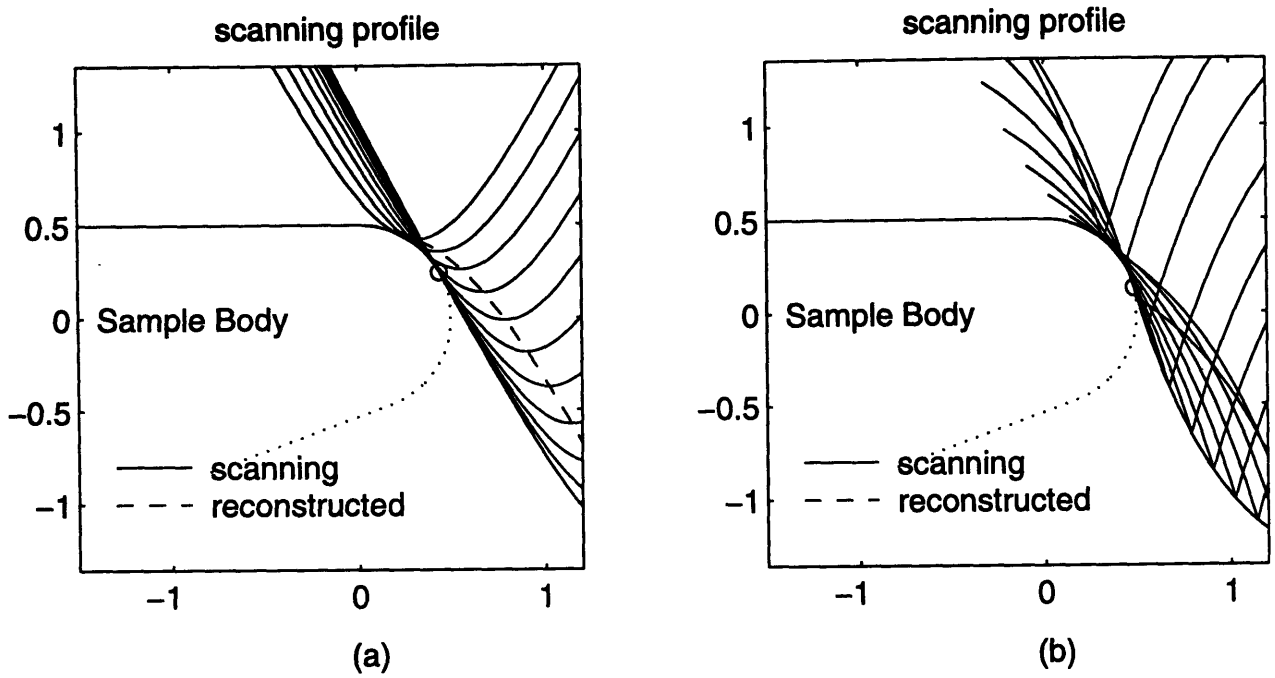


Figure 3-6: Envelop Image Analysis for a Tip with Upward Convex Shape

Figure 3-7 shows, the restricted envelop image analysis method causes less distortion to the downward fluctuation of noise compared to the original method.

3.4 Edge Identification

Once the image is reconstructed from the noised and distorted data, the next step is to find the intermediate point between the sample profile and the unreconstructable region filled by the tip shape, or the "edge".

In an image system, an edge is a boundary or contour at which a significant change occurs in some physical aspect of an image, such as the changes in image intensity. This is similar to the result of scanning a sample's edge area. While scanning tip is just "falling off" the sample edge, theoretically, there will be sudden change in vertical, and this corresponds to the intensity change if we look at the scanning data as a gray-scaled image. So, the basic idea to detect the edge in image analysis could be borrowed here.

The common methods in edge detection contain the calculation of its first or second derivatives. Consider a function $f(x)$ which represents a typical 1-D edge in image processing problem, as shown in Figure 3-8. Its first and second derivatives are

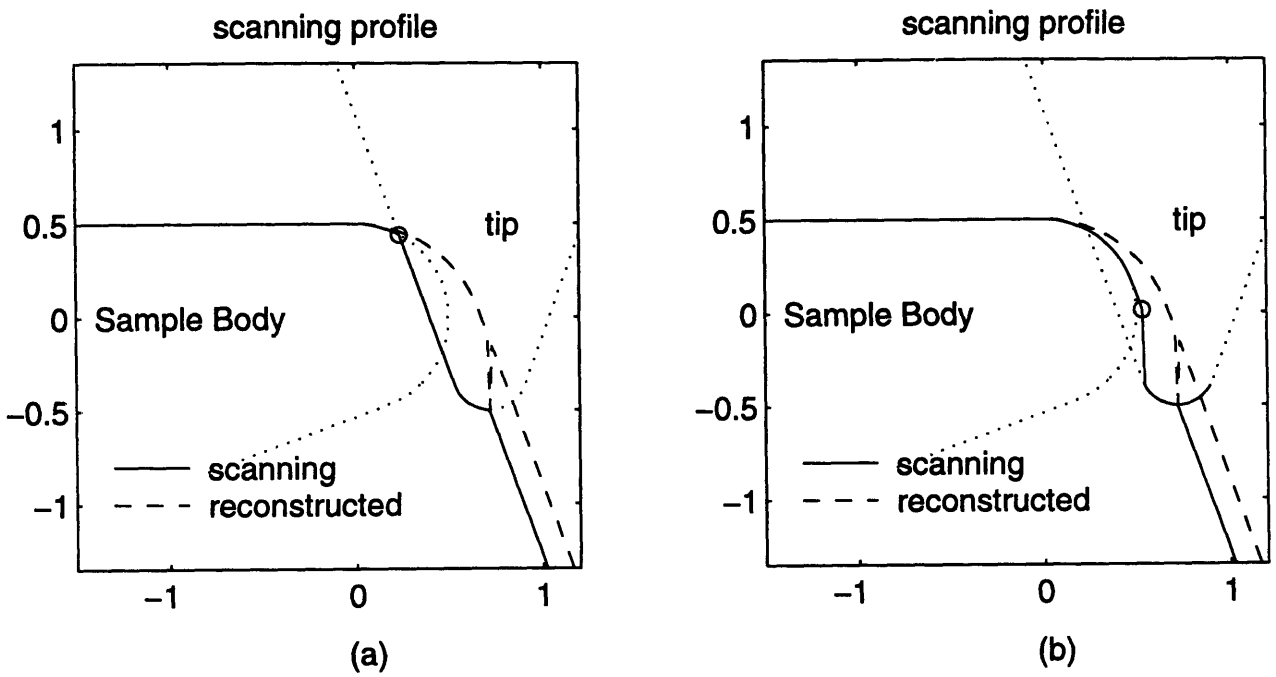


Figure 3-7: Deconvolution with Noise (a) Envelop Image Analysis (b) Localize Envelop Image Analysis to the Tip Area

also given. One way to determine the edge point x_0 is to compute the first derivative $f'(x)$, and x_0 can be determined by looking for the local extremum of $f'(x)$. Or we can calculate the second derivative $f''(x)$ and x_0 can be determined by looking for a zero crossing of $f''(x)$.

A similar approach can be used in the sample's edge identification, but here, the edge to be identified has it's physical meaning (the intermediate point between the sample edge and the reconstructed scanning tip segment), some modifications are needed.

3.4.1 First-Derivative-Based Method

In a simplified model, the sample and the scanning tip are both modeled as an arc with two straight sidewall, see Figure 3-9(a). While scanning, due to the finite size and the sidewall of the tip, the scanning result shows some distortion (the dashed line in the figure). After deconvolution according to the previous section's discussion, then the first and second derivatives of the reconstructed data are calculated and plotted in the subsequent figures in Figure 3-9(a) with circles marked at the corresponding edge point. It is shown that, to identify the edge correctly by looking at its first derivative.

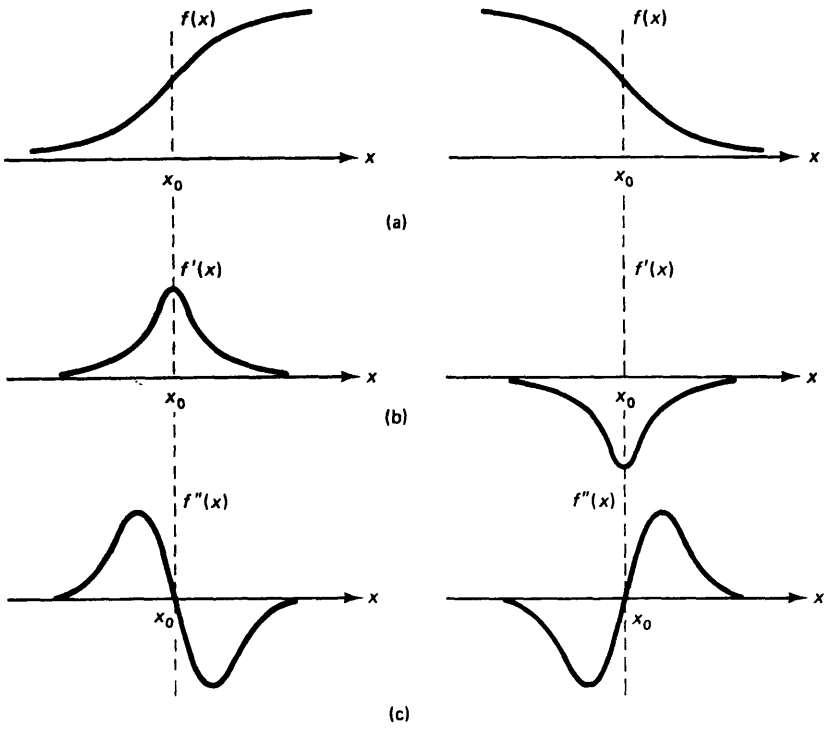


Figure 3-8: Edge Detection[10]

a threshold should be set and the edge point is the first point which exceed the threshold. If all the sizes and shapes are well known and it is noise-free, it is possible to set the threshold beforehand by calculation. But usually, the threshold has to be set by experimental method.

3.4.2 Second-Derivative-Based Method

If the second-derivative method is adopted, then the criterion is no longer the point with second derivative value crossing zero. Instead, the point with extreme value should be identified as the sample edge.(See Figure 3-9(a))

This is generally true if the sample's edge region and the scanning tip are both smooth enough (with continuous first derivative) and there is large open angle between the sidewall of the tip and the sample surface. Since the reconstructed curve is composed of the tip surface and the sample surface , at the intermediate point, there is an abrupt change in the slope and this corresponds to the extreme of its second

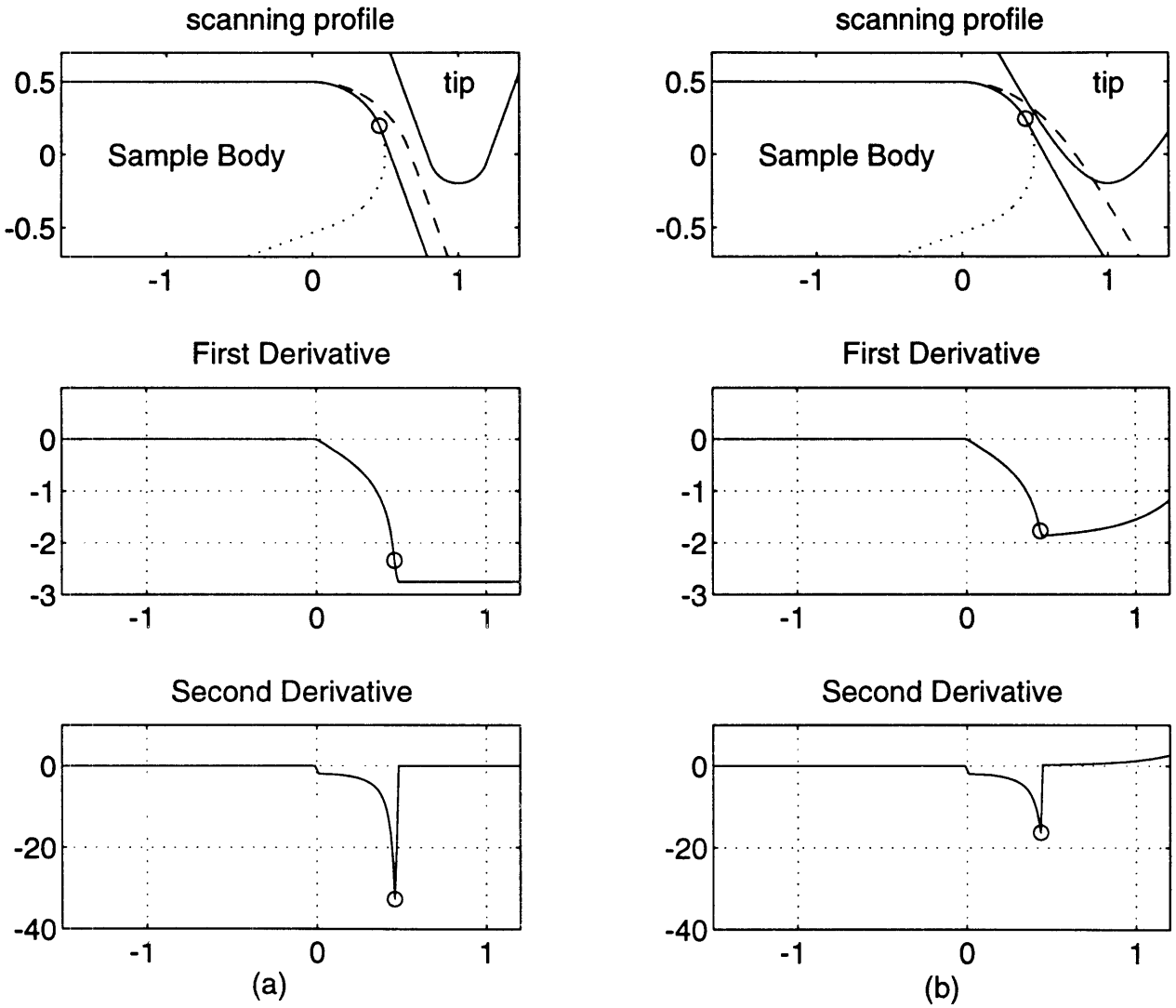


Figure 3-9: Simulation of Sample Edge Detection (a) with a Simple Tip (b) with a More Realistic Tip

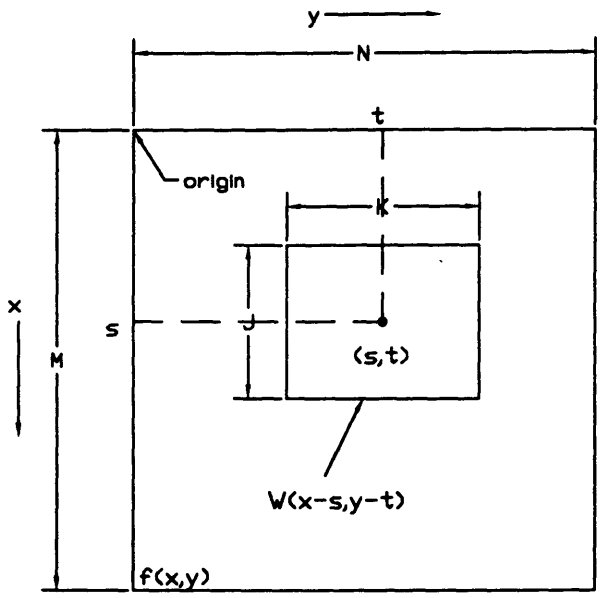


Figure 3-10: Image Matching

derivative.

The advantage of the second derivative based method over the first derivative based method is that no threshold is needed, therefore the result will not be influenced by the tip shape. But since the second derivative of the scanning data is taken, it is more sensitive to noise, and if the sample surface has some sharp feature, this method will tend to fail. To avoid this situation, the first derivative of the profile at the extreme value point sould also be calculated for the double check.

Another simulation is shown in Figure 3-9(b). Here, a more realistic tip shape is considered. After deconvolution, the edge still can be found by use of one of these methods.

One thing that should be kept in mind is, due to the sensitivity to noise, (first and second derivatives are taken here), the application of a noise reduction system prior to edge detection is very desirable.

3.5 Image Matching

As mentioned earlier, due to the inevitable error from the sample's rotation, the scanning position will not exactly correspond to the same spot before and after the

sample is flipped. This might distort the reconstructed 3-dimensional profile. To reduce this error, the images should be "shifted" back before they are combined together.

For a image $f(x, y)$ of size $M \times N$ (pixels), if we want to find a region which is matched to a subimage $w(x, y)$ of size $J \times K$, a straightforward and conventional way is to calculate the correlation of the two images :

$$c(s,t) = \sum_x \sum_y f(x,y)w(x-s, y-t) \tag{3.5}$$

where $s = 0, 1, 2, ..., M-1, t = 0, 1, 2, ..., N-1$ and it is calculated with $w(x, y)$ moving on top of $f(x, y)$. (See Figure 3-10) The maximum value of $c(s, t)$ indicates the optimal match position [3].

The concept to match two sides images of the sample is as following : ideally, the edges of the two images should be symmetric if they represent the same area of the sample. If each pixel of the images which presents the sample is assigned as one (1) and the pixel which doesn't present the sample is assigned as minus one (-1), flipping over one of the two assigned images, then they should be identical. To match these two images, a rectangular subimage along the edge is chosen and compared to another image, the matching point will be the position with maximum correlation.

If the sample is not aligned properly, or there is orientation errors after flipping the stage, the two edges might not be in the same orientation, and the image rotation is needed before calculation, otherwise, the correlation method will fail. One way to do that is to fit a straight line to the sample edge first, then take that line as a characteristic line, and rotate the image to let this line have the same orientation in the two images for matching. An example result of matching the two sides sample images is shown in Figure 3-11, where the small rectangular represents the subimage for matching.

Of course, to make this technique work, the sample edge identified should have some obvious features, so they could be compared. (e.g., if the sample edges are perfectly straight, then it is impossible to match them by this method.) Also, the

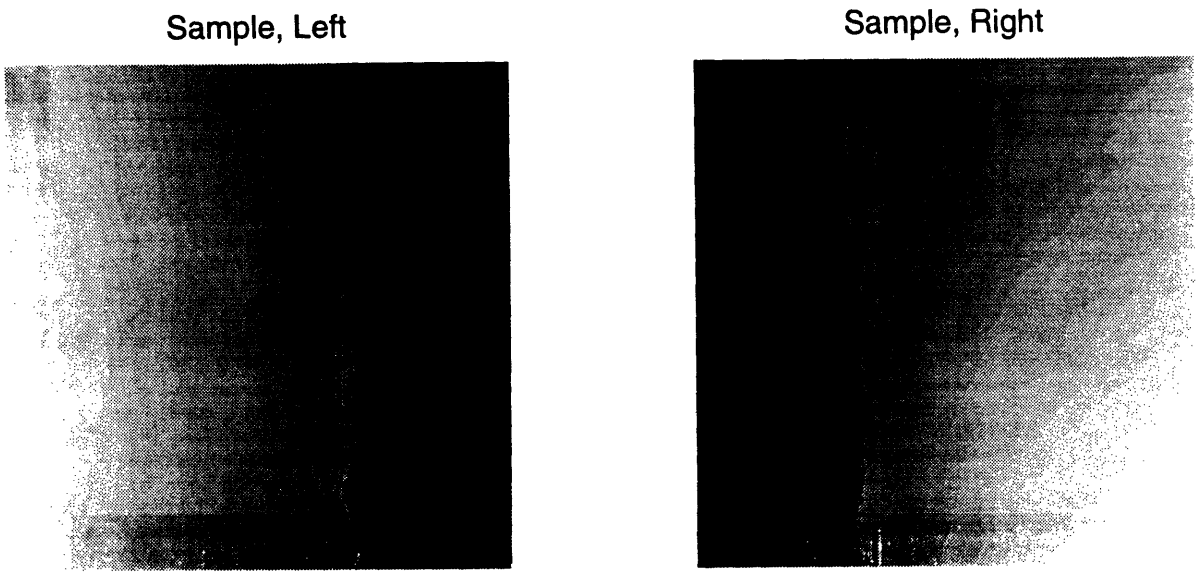


Figure 3-11: Matching of Two Sides Sample Images

offset error should be small enough to find the matching position. (If the offset is too large, e.g., larger than the image size, then again, it is impossible to correctly match the two images.). From experiment we found, in a typical $5 \times 5\mu$m area scanning, the sample does show some "wiggle" that can be used as the features for identification. [4]

3.6 Images Combination

Once the sample edge has been identified from two sides, the final step is to reconstruct the original three dimensional sample profile.

The definition of the coordinate system is depicted in Figure 3-12. [5] The global reference frame $\{R\}$ is chosen to coincide with the scanner frame $\{A\}$. Frame $\{D\}$ represents the X stage's coordinate system and moves with the stage. Frame $\{B\}$

[4]Due to the calculation time constraint, the image matching technique is not implemented in the software described in chapter 4.

[5]See appendix A for the notation and some basics of the coordinate transformation

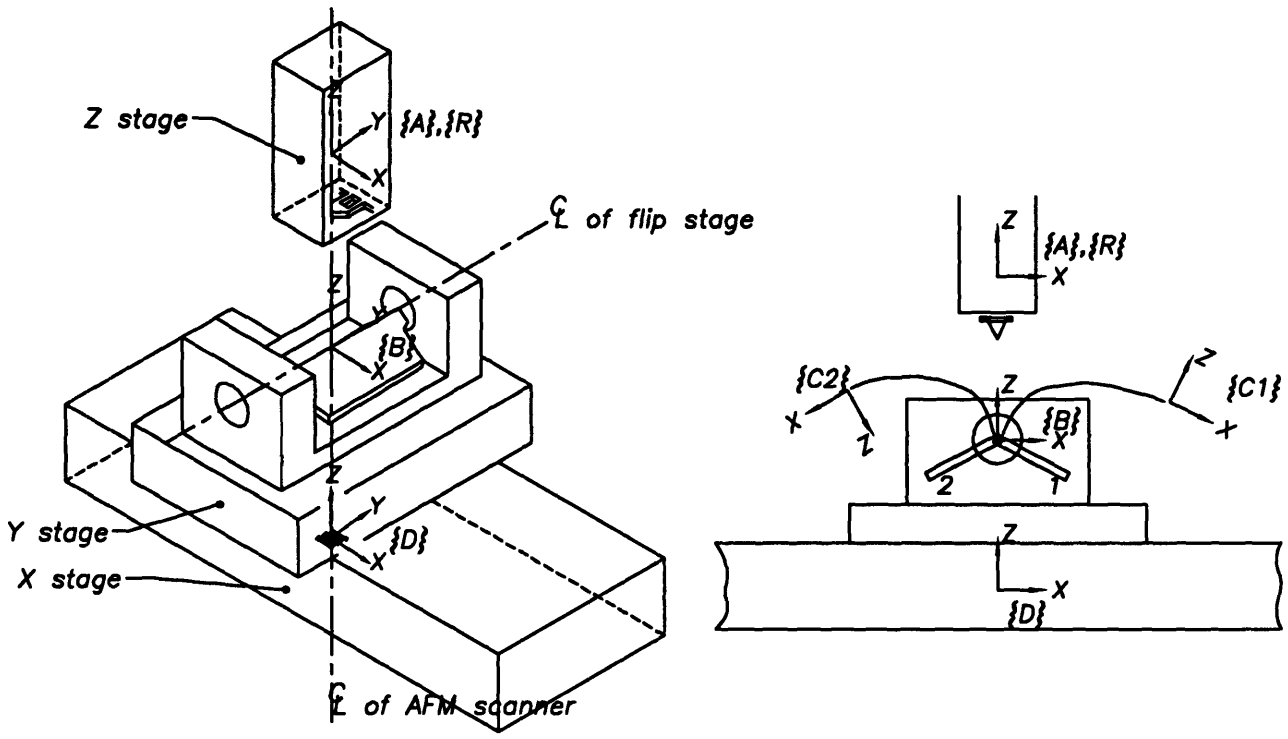


Figure 3-12: System Coordinations Definition

represents the flip stage's coordinate system, where $\hat{Y}_B$ is defined as the rotation axis of the flip stage. Frame $\{C\}$ represents the sample's coordinate system. Before rotating, frame $\{C\}$ is coincided with frame $\{B\}$. For profiling, the flip stage is rotated an angle θ_1 and $(\pi - \theta_1 - \theta_2)$ respectively to scan the two sides of a sample and bring the sample's local coordinate system frame $\{C\}$ to the corresponding position. [6] [7] The notations of frames $\{C_1\}$ and $\{C_2\}$ are used to represent the respective position and orientation of the sample coordinate system $\{C\}$.

[6]Note : in this definition, frame $\{B\}$ is fixed with respect to frame $\{D\}$ and it doesn't move along the flip stage.

[7]See Appendix B for the rotating angle θ_1 and θ_2.

3.6.1 Simplified Method

For a point P on the sample surface at position 1, it is defined with respect to frame $\{C_1\}$ by a 3×1 position vector $\mathbf{P}^{C_1}$, its description with respect to frame $\{A\}$ is : [8]

$$\mathbf{P}^A = \mathbf{R}^A_{C_1}\mathbf{P}^{C_1} + \mathbf{P}^A_{C_{1org}} \tag{3.6}$$

or

$$(\mathbf{P}^A - \mathbf{P}^A_{C_{1org}}) = \mathbf{R}^A_{C_1}\mathbf{P}^{C_1} = \mathbf{R}^A_R\mathbf{R}^R_D\mathbf{R}^D_B\mathbf{R}^B_{C_1}\mathbf{P}^{C_1} \tag{3.7}$$

where $\mathbf{P}^A$ is gathered directly from AFM, $\mathbf{P}^A_{C_{1org}}$ is the position vector of the frame $\{C_1\}$'s origin represented in frame $\{A\}$, and can be defined to be arbitrary edge point determined from the previous sections.

For a point Q on the sample's other side surface at position 2, a similar result is : [9]

$$\mathbf{Q}^A = \mathbf{R}^A_{C_2}\mathbf{Q}^{C_2} + \mathbf{Q}^A_{C_{2org}} \tag{3.8}$$

or

$$(\mathbf{Q}^A - \mathbf{Q}^A_{C_{2org}}) = \mathbf{R}^A_{C_2}\mathbf{Q}^{C_2} = \mathbf{R}^A_R\mathbf{R}^R_D\mathbf{R}^D_B\mathbf{R}^B_{C_2}\mathbf{Q}^{C_2} \tag{3.9}$$

Ideally, these coordinate systems are arranged with the relationship below :

$$\bar{\mathbf{R}}^A_R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \bar{\mathbf{R}}^R_D = \bar{\mathbf{R}}^D_B \tag{3.10}$$

[8] The 3x3 rotation matrix representation is used instead of the 4x4 homogeneous transformation matrix here. Frame $\{C\}$'s orientation is fixed once the sample is put on the flip stage, but its position is not. Frame $\{C\}$'s origin can be defined at arbitrary sample tip point directly from the scanning data. It means, what we really care about is the sample's orientation instead of the sample's absolute position. By using 3x3 rotation matrix representation, we can take apart the position terms $\mathbf{P}^A_{C_{2org}}$ and $\mathbf{Q}^A_{C_{2org}}$ and concentrate on the orientation terms. It can reduce the complexity of calculation.

[9] Here, Q instead of P is used to represent a point on the sample surface to distinguish two points on different sample sides

$$\bar{\mathbf{R}}_{C_1}^{B} = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ 0 & 1 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 \end{bmatrix} \tag{3.11}$$

and

$$\bar{\mathbf{R}}_{C_2}^{B} = \begin{bmatrix} -\cos\theta_2 & 0 & \sin\theta_2 \\ 0 & 1 & 0 \\ -\sin\theta_2 & 0 & -\cos\theta_2 \end{bmatrix} \tag{3.12}$$

where $\bar{\mathbf{R}}$ is used to represent the ideal case.

So, for point P on the sample surface, substitute these rotation matrix into equation 3.7, we can get a simple result below :

$$\Delta\mathbf{P}^{A} = (\mathbf{P}^{A} - \mathbf{P}_{C_{1org}}^{A}) = \mathbf{R}_{C_1}^{B}\mathbf{P}^{C_1}$$

then

$$\mathbf{P}^{C_1} = \mathbf{R}_{C_1}^{B\ -1}\Delta\mathbf{P}^{A} = \mathbf{R}_{B}^{C_1}\Delta\mathbf{P}^{A}$$

or

$$\begin{bmatrix} P_X^{C_1} \\ P_Y^{C_1} \\ P_Z^{C_1} \end{bmatrix} = \begin{bmatrix} \cos\theta_1\Delta P_x^{A} - \sin\theta_1\Delta P_z^{A} \\ \Delta P_y^{A} \\ \sin\theta_1\Delta P_x^{A} + \cos\theta_1\Delta P_z^{A} \end{bmatrix} \tag{3.13}$$

where $\Delta\mathbf{P}$ is calculated once some sample edge point is defined as the origin of frame $\{C_1\}$.

Similarly, the equation for point Q is :

$$\begin{bmatrix} Q_X^{C_1} \\ Q_Y^{C_1} \\ Q_Z^{C_1} \end{bmatrix} = \begin{bmatrix} -\cos\theta_1\Delta Q_x^{A} - \sin\theta_1\Delta Q_z^{A} \\ \Delta Q_y^{A} \\ \sin\theta_1\Delta P_x^{A} - \cos\theta_1\Delta P_z^{A} \end{bmatrix} \tag{3.14}$$

Assume the sample's edge points identified from two sides are the same [10], then the origins of frame $\{C_1\}$ and frame $\{C_2\}$ should be the same, too. Converting all

[10] See appendix B

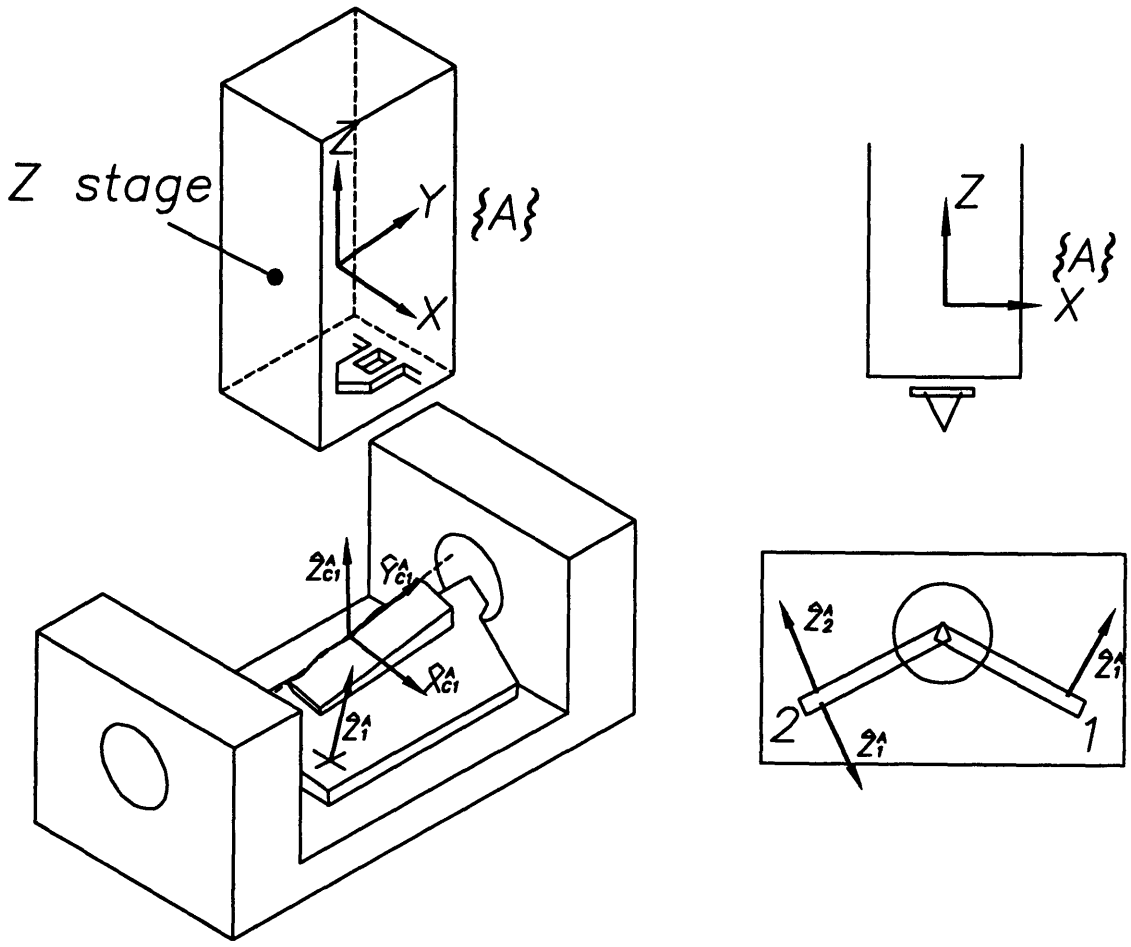


Figure 3-13: Coordinate System for Corrected Method

the data from frame $\{A\}$ to frame $\{C\}$ by use of equations 3.13 and 3.14 and putting them together, the sample profile can be reconstructed.

3.6.2 Corrected Method

In the previous section, one assumption is that the orientation of each coordinate system is perfectly aligned, also the sample is carefully placed in the desired orientation. However, it is hard to achieve especially nanoscale accuracy is required. Also there is variation of the orientation of the scanner's coordinate system everytime the tip is reloaded or the scanner is remounted.

A more accurate, yet more complicated method is presented here. It can correct the errors due to the misalignment of the sample, but it needs extra measurement except to measure the sample to acquire the information about the stage's orientation.

Figure 3-13 shows a sample on the holder. First, the unit vector $\hat{\mathbf{Z}}_1^A$ which is

normal to the reference plane of the flip stage can be calculated by scanning at least 3 points on the reference plane or a flat sample, like an optical flat sample, on the plane. Also, once the sample edge is identified, a straight line can be fitted, and it's direction is assigned as $\hat{\mathbf{Y}}^P_{C_1}$, then we can define $\hat{\mathbf{X}}^A_{C_1}$ as :

$$\hat{\mathbf{X}}^A_{C_1} = \hat{\mathbf{Y}}^A_{C_1} \times \hat{\mathbf{Z}}^A_1 \tag{3.15}$$

$$\hat{\mathbf{Z}}^A_{C_1} = \hat{\mathbf{X}}^A_{C_1} \times \hat{\mathbf{Y}}^A_{C_1} \tag{3.16}$$

Then, the rotation matrix from frame $\{C_1\}$ to frame $\{A\}$ will be :

$$\mathbf{R}^A_{C_1} = \begin{bmatrix} \hat{\mathbf{X}}^A_{C_1} & \hat{\mathbf{Y}}^A_{C_1} & \hat{\mathbf{Z}}^A_{C_1} \end{bmatrix} \tag{3.17}$$

After flipping the stage over, the other side of the reference plane is scanned, and the unit vector $\hat{\mathbf{Z}}^A_2$ can be calculated. If two sides of the reference plane are parallel, then $\hat{\mathbf{Z}}^A_1 = -\hat{\mathbf{Z}}^A_2$. Following the previous procedure, we can find the sample edge from the other side, and the unit vector $\hat{\mathbf{Y}}^A_{C2}$ along the edge. Then, again:

$$\hat{\mathbf{X}}^A_{C_2} = \hat{\mathbf{Y}}^A_{C_2} \times -\hat{\mathbf{Z}}^A_2 \tag{3.18}$$

$$\hat{\mathbf{Z}}^A_{C_2} = \hat{\mathbf{X}}^A_{C_2} \times \hat{\mathbf{Y}}^A_{C_2} \tag{3.19}$$

And the rotation matrix from frame $\{C_2\}$ to frame $\{P\}$ will be :

$$\mathbf{R}^A_{C_2} = \begin{bmatrix} \hat{\mathbf{X}}^A_{C_2} & \hat{\mathbf{Y}}^A_{C_2} & \hat{\mathbf{Z}}^A_{C_2} \end{bmatrix} \tag{3.20}$$

At the assumption of the sample edge identified from the two sides are identical, it means $\hat{\mathbf{Y}}_{C_1} = \hat{\mathbf{Y}}_{C_2}$. Then $\mathbf{R}^A_{C1}$ and $\mathbf{R}^A_{C2}$ correspond to the two rotation matrix from the sample coordinate system to the AFM's coordinate system. Substitute these matrix into equations (3.6) and (3.8), the sample profile can be reconstructed.

In this way, the errors from the sample misalignment or the stages errors could be eliminated. But extra scans to identify the reference plane direction is needed and

the relationship between both sides of the reference plane have to know beforehand.

Figure 3-14 shows one example of the reconstructed profile of the sample in 3D view. It is done by use of the simplified method and the necessary 3D information was generated by the software described in the next chapter.

3.7 Summary

In this chapter, the problem for the data processing in the three dimensional profilometer was point out first, then the feasible procedures to reconstruct the profile were proposed. Each procedures were also discussed and tested separately.

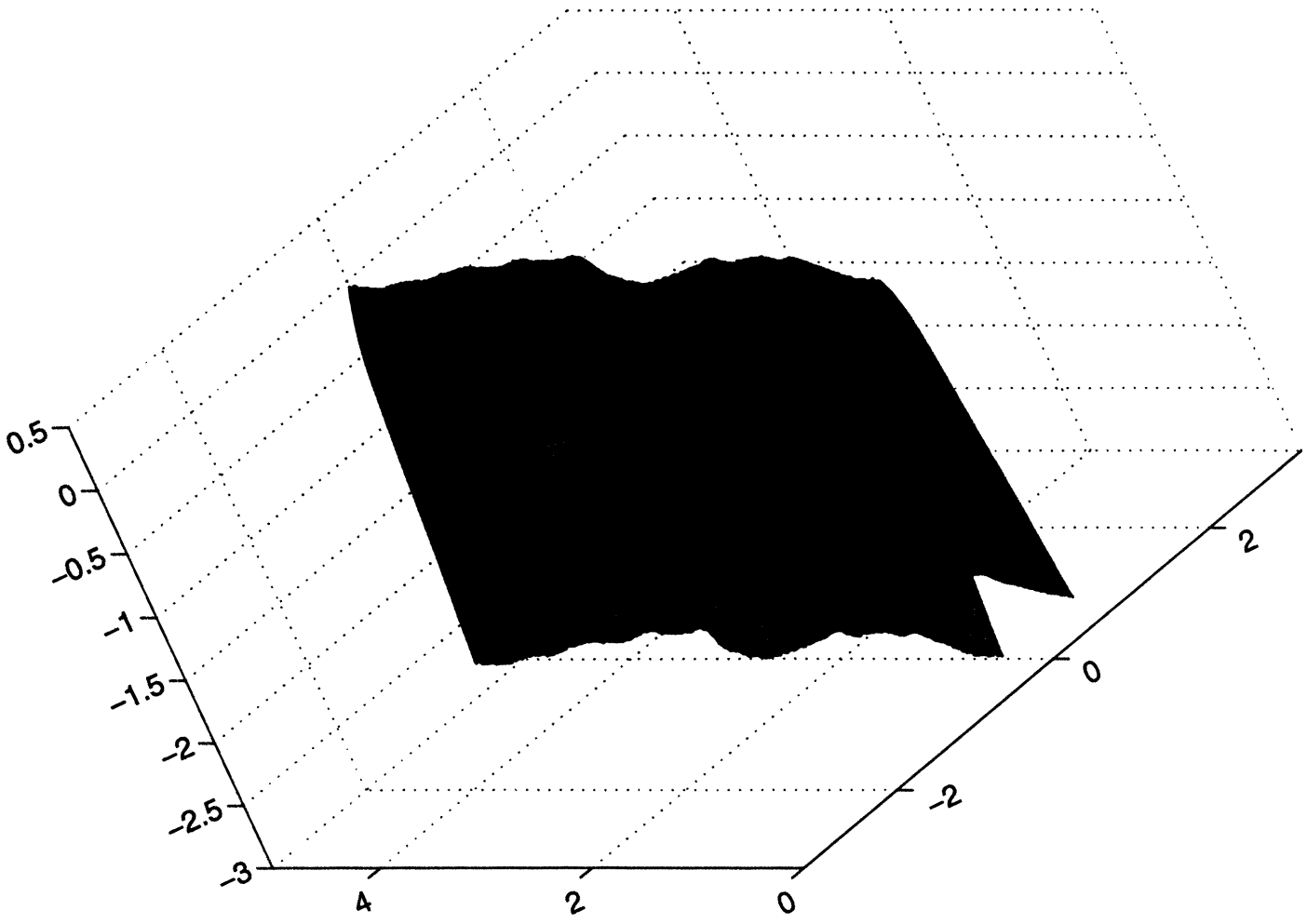


Figure 3-14: 3D Profile

Chapter 4

Software Implementation

4.1 Instroduction

To realize the procedures described in the previous chapter, a computer program was implemented. At first, the software was realized under MATLAB's environment. Basically, MATLAB is an interactive program for scientific and engineering numeric calculation. Its matrix-based data format makes it particularly suitable for signal and image processing like the work we were dealing with. Besides, Matlab's windows interface and rich 2-D/3-D graphics functions make data analysis and presentation easier than the traditional programming language. Figure 4-1 shows one user interface of this prototype program. One drawback of programming on the MATLAB environment is the slow speed compared to the other programming language like C. For this particular application, the matrices dealt with are usually 256×256 or even 512×512 in size and the computation time it costs typically takes up to 10 minutes on a 486DX2-66 PC in this program. To improve this situation while keeping the program flexible and easy for further development, C language was chosen as the developing tool and the personal computer and MS-Windows are chosen as the developing platform and the operating system.

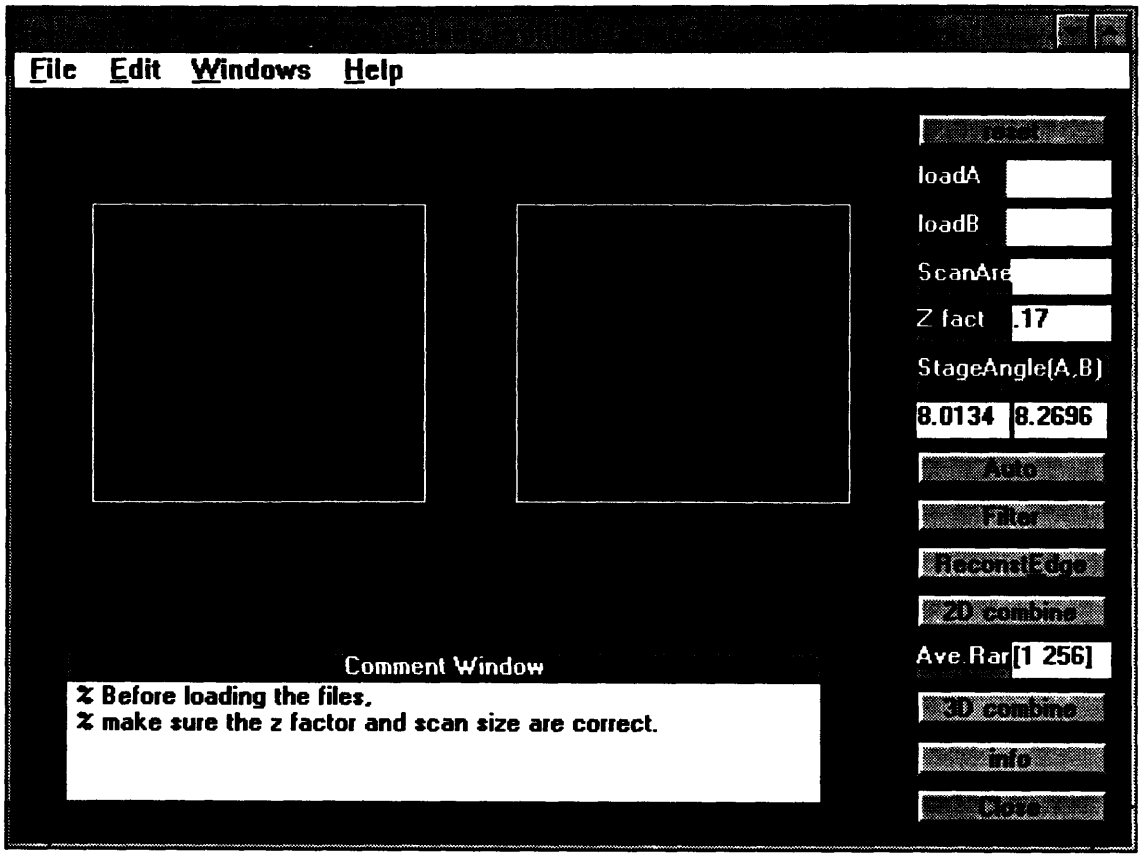


Figure 4-1: Matlab Program Interface

4.2 Data Structure

Most of the data we are facing here are vectors and matrices in natural, it is therefore important to deal with them efficiently [1].

In C language, there is a close, and elegant, correspondence between pointers and arrays. The value referenced by an expression like $a[j]$ is defined to be $*((a) + (j))$, that is, "the contents of address obtained by incrementing the pointer a by j." A consequence of this definition is that if a points to a legal data location, the array element $a[0]$ is always defined. Arrays in C are natively "zero-origin" or "zero-offset." An array declared by the statement $float\ b[4]$; has the valid references $b[0]$, $b[1]$, $b[2]$, and $b[3]$ but not $b[4]$. [2] In general, the range of an array declared by $float\ a[M]$; is $a[0..M-1]$.

One problem is that many algorithms naturally like to go from 1 to M, not from 0

[1] Parts of this paragraph are adapted from [12]

[2] Here, the notation $b[0..3]$ is introduced to indicate the index range of vector b

to M-1. To avoid this annoying conversion problem between "zero- origin" and "one-origin", the pointer to a matrix or a vector is offset one like below in the program :

```
float b[4], *bb;
bb = b-1;
```

In other words, the range of *bb* is *bb*[1..4] and *bb*[1], ..., *bb*[4] all exist.

Another problem comes from the size of a matrix. In C language definition, it cannot pass two-dimensional arrays whose size is variable and known only at run time. That is, a function defined below is not allowed in C:

```
void someroutine(a,m,n)
float a[m][n]
```

In this program, this problem was solved by defining a matrix structure below :

```
typedef struct {
int row;
int column;
float *buffer;
} MatrixF;
```

where buffer is a pointer points to the beginning of a matrix and it is unit offset, too. For a row or column vector, it is also treated as a matrix since a vector is only a special case of matrix. In this way, a new data structure *MatrixF* instead of an array is passed to a function and the function can easily get the information about the matrix size and matrix address for further calculation. [3]

4.3 User Interface

To enhence the flexibility and ease of further improvement and development, the whole programs was breaking down into one main program and several dynamic link

[3]See appendix D for the program codes listing.

library (DLL) files. The main program manages the user interface and the interaction between the other programs whiles others in charge of basic matrix manipulation, data processing and database-related functions.

Besides realizing the procedures described in chapter 3, the software provides the bridge between the user and the programs, so it has been decided to contain the features below :

1. show the two images simutaneously for comparison.
2. provide a close up view of the images.
3. represent the 2D/3D view of the final result.
4. represent the cross section view of the result.
5. provide the essential parameters and information and some of them should be allowed to be changed by the users.

The basic program outlook is shown in Figure 4-2. The interface can be divided into 5 subwindows. The main window is the parent window which contains the other child windows and some useful buttons. It is used to communicate with the user and also displays the essential information and parameters. The view windows A and B are used to display the gray scale images, and the corresponding zoom-in windows show the close-up view of the images. Those windows also show the filtered or reconstructed images according to user's option. View 2D window shows the resulting 2-D data plot.

4.4 Software Operation Procedure

The basic procedures to operate the program are as following : [4]

1. load the 2 sides images by selecting *File/Open* option.
2. choose the filtering method and the edge identification method by selecting appropriate items in *Setup* option.

[4]See appendix C for detailed description of the program functions.

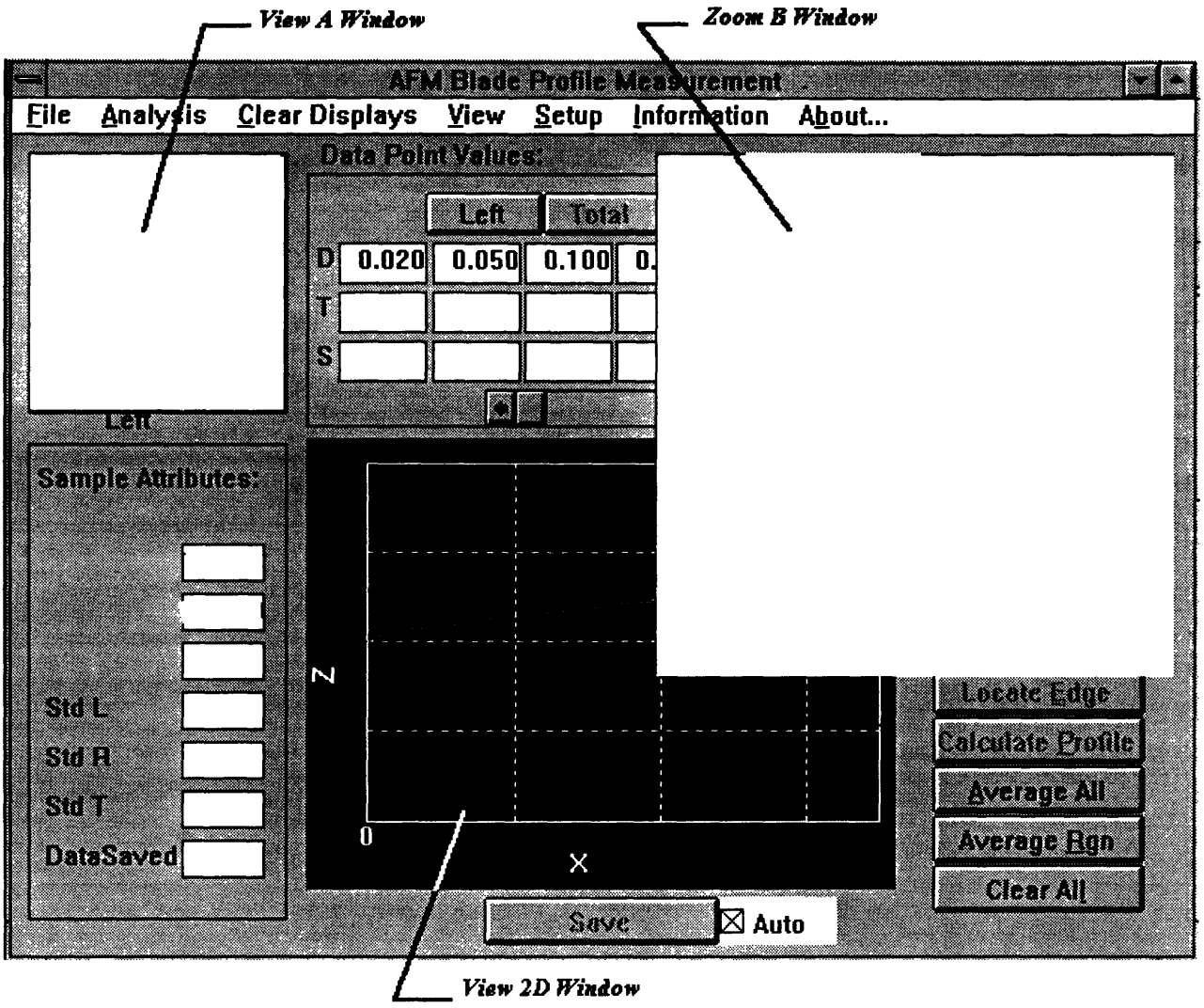


Figure 4-2: Software User Interface

For filtering, the 1-D separable median filter with window size 5×1 and the cascade 1-D adaptive Wiener filter with window size 5×1 are implemented. For sample edge identification, the first and second derivatives based methods are implemented.

3. either run the program step by step by selecting the corresponding options in *Analysis*, or click *Find Edge/2D combine* buttons to automatically run the procedures above.

4. see the corresponding trace profile in view 2D window by clicking and dragging the mouse cursor in the image view A or B window or click the *Average* button to see the average 2-D profile.

5. save 2D/3D data for further analysis.

4.5 Summary

In this chapter, the essential software implematation consideration and data structure used in the programes are discussed. The brief operation procedure is described, too. A detailed list of the programs organization and the explanation of the functions provided in this program could be found in appendix C and D.

Chapter 5

Errors Analysis

5.1 Introduction

Although the scanning-probe-microscope-based profilometer provides a ultrahigh resolution and a close-up view of dazzling three dimensional surface topographic features, it's limitation in metrology application should be kept in mind. Does it faithfully show the real surface topography ? or it is mixed with some other information ? How accurate is it ? and how to reduce the error ? Because the small scale we are dealing with, some factors not considered in other applications might have great effects to its output here. In this chapter, some of the possible error sources in this instrument are presented and discussed. Some errors might be quantitatively estimated, some are impractical to obtain.

This system can be basically divided into 3 parts: the AFM, positioning stages, and post-processing software part. It is not easy to quantify the post-processing errors, also, a large part of the post-processing errors like edge detection are directly influenced by the hardware or sensor errors, so it is not discussed here. The errors described here can be divided into the AFM-related errors and the positioning stage errors.

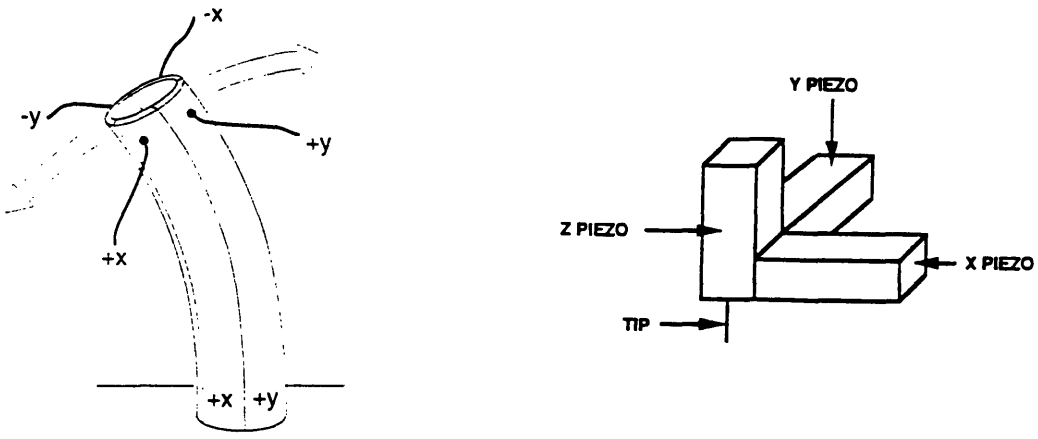


Figure 5-1: (a) Piezoelectric Tube[13] (b) Tripod Scanner [2]

5.2 AFM-Related Errors

As described in the previous chapter, an AFM is basically composed of a scanning actuator, a microstylus-mounted cantilever and some electronic components for data acquisition and gathering.

5.2.1 Piezo Actuator

For most of the scanning probe microscopes, a piezoelectric scanner is used as an extremely fine positioning stage to move the probe or the sample, and the most widely used form of piezoelectric scanner is the simple tube design introduced by Binnig and Smith[1] and is depicted in Figure 5-1(a). This consists of a thin-walled tube of hard piezoelectric which is polarized radially. Electrodes are applied to the internal and external faces of the axis, with the outer electrode of the piezo tube sectioned into four quadrants. By applying appropriate voltages on the electrodes, it can generate displacements in three dimensions. The advantages over the tripod scanner (see Figure 5-1(b)) are the higher resonant frequency, greater range and greater thermal resistance it provides. Also its smaller size can greatly simplify vibration isolation. But its geometrical arrangements and the properties of the piezos can cause several problems.

As a first approximation, the strain in a piezoelectric scanner varies linearly with applied voltage. By applying the appropriate voltages in x and y, the piezoactuator

will move linearly in the corresponding direction. Practically, the behavior of piezoelectric scanner is not so simple. It will show some nonlinearity like hysteresis and creeping. some approach is needed to eliminate those errors.

Intrinsic Nonlinearity

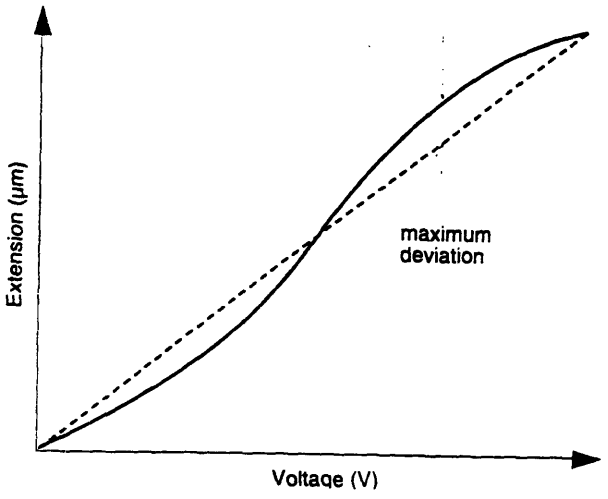
Starting from zero voltage, if we applied voltage to a scanner gradually to eliminate the other effects like hysteresis or creeping, the relationship between the applied voltage and the extension will look like that of Figure 5-2(a). This intrinsic nonlinearity will effect both in the x-y direction and in the z direction. An effective method to eliminate this error is to use independent position sensors in the x-y, and z directions as the feedback signal to correct the applied voltage or to monitor the actual movement of the tube.

Hysterisis

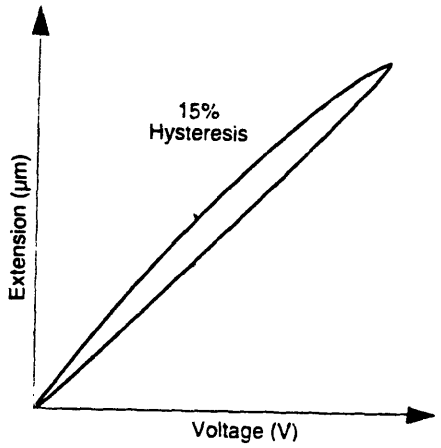
Piezoelectric ceramics also display hysteresis behavior. Suppose we start at zero applied voltage, gradually increase the voltage to some finite value, and then decrease the voltage back to zero. If we plot the extension of the ceramic as a function of the applied voltage, the descending curve doesn't retrace the ascending curve, it follows a different path, as shown in Figure 5-2(b). Most of the SPMs are configured so that in one scanning session, the data are collected in only one direction to minimize this effect in x-y directions. But again, the most effective way to eliminate this error is to use independent x-y/z position sensors to correct/monitor the tip position.

Creep

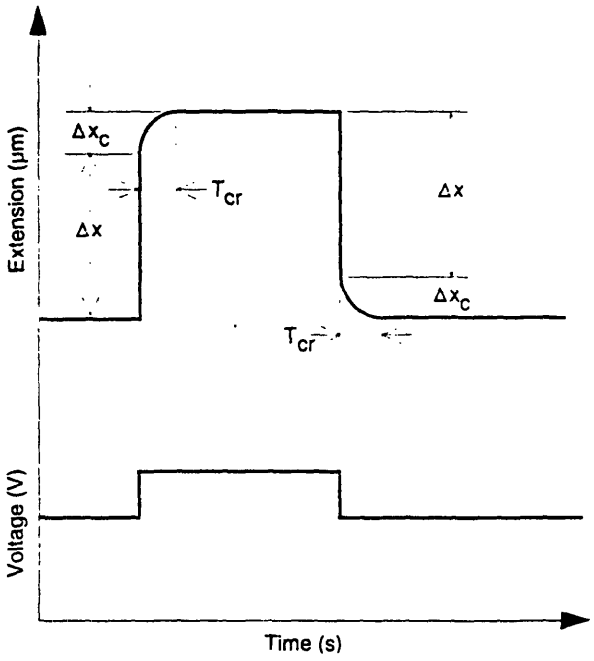
Another important nonlinear behavior in piezoelectric scanner is creep. When an abrupt change in voltage is applied, the piezoelectric material does not change dimension all at once. Instead, the dimensional change occurs in two steps : the first step takes place in less than a millisecond, the second is on a much longer time scale. The second step is known as creep. (See Figure 5-2(c).) As a result, two scans taken



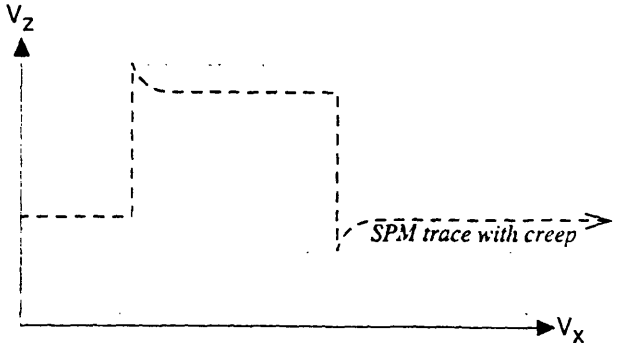
Intrinsic nonlinearity in a piezoelectric scanner



Hysteresis in a piezoelectric scanner



Creep in a piezoelectric scanner



The effects of creep on an SPM image of a step.

Figure 5-2: Nonlinearity of Piezoelectric (a) Intrinsic Nonlinearity (b) Hysteresis (c) Creep (d) The Effect of Creep to a Step Sample [13]

at different scan speeds show slightly different length scales when creep is present. So, the calibration might be needed for different scanning speed.

This nonlinear behavior also has great effects in the z direction. As shown in Figure 5-2(b), as the tip traverses the step from bottom to top, the scanner contracts immediately with a voltage corresponding to the full step height. However, over the next few seconds the scanner will continue to contract slowly as creep occurs. To keep the tip in contact with the sample, the SPM will have to apply a voltage in the other direction, counteracting the creep. A similar situation happens when the tip traverses the step from top to bottom. This error will happen while scanning the ultimate edge of a sample. When the probe touches the sample edge, there will be an abrupt change in voltage, it will cause error due to creeping. One way to eliminate this error is to adjust the scanning direction. Scanning from the sample body toward the edge can remove the error, since we don't need the information after the probe falls down the edge, the creep errors here is not important. To further reduce the errors, an independent position sensor in z direction to monitor the scanner's movement is needed.

Cross Coupling

Maybe the most terrible sin for SPMs in metrology is that its configuration deviates from the cartesian coordinate system, and causes the "coupling" problem. The term cross coupling refers to the tendency of x-axis or y-axis scanner movement to have a spurious z-axis component. It arises from several sources, but the main reason is from the geometry, because the x, y, and z movement of the piezotube are not independent, as a result, a piezoelectric tube scans in an arc, not in a plane. To eliminate this error radically, a new method to move the scanner should be proposed to provide the actuator's independent x, y, and z movement, or independent x-y/z position sensors for real time scanner correction is needed to reduce this error. For the best, an independent z position sensor to monitor the scanning tip's actual movement should be used, but due to the difficulty to measure the tip position directly, the position of the actuator is monitored. Unfortunately, this is a violation of the Abbé principle.

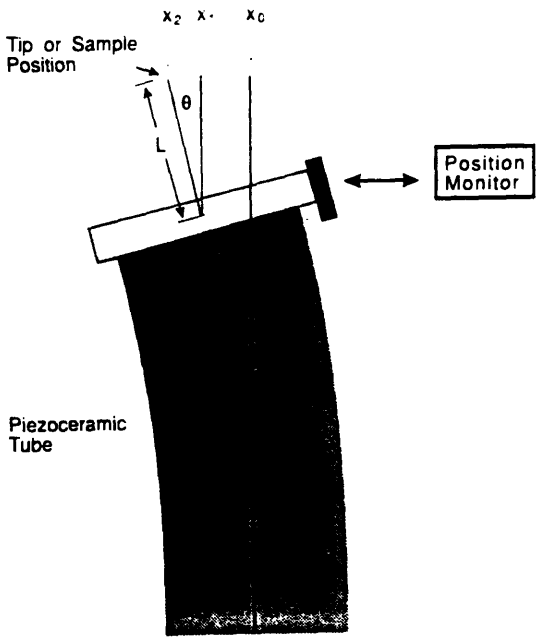


Figure 5-3: Effect of Tube Tilting on the Lateral Displacement of the Tip or Sample[4]

Figure 5-3 illustrates the effect of tilting on a position monitor. This effect applies to systems that mount the tip on the tube and to systems that mount the sample on the tube. We imagine that either the probe tip or the sample surface is a distance L from the end of the tube from the end of the tube. Before flexing the point we want to measure is at x_0. During deflection the tilt angle theta adds an extra motion x_2 to the motion x_1 detected by the position monitor. Here, θ is a function of lateral tube displacement.

While piezoelectric contains many nonlinearity above, and practically, it is hard to tell apart the errors from intrinsic nonlinearity, hysteresis, creep, or cross coupling, they are usually mixed together. But as we can see from the above discussion, one approach to obtain a linear scan from the piezoelectric is having real-time scan correction by use of some kind of position sensors. As shown in Figure 5-4(a), a linear, 9.9 micron-pitch grating is scanned without x-y and z position sensors on, and the image shows nonuniform spacing and curvature (as seen from the different gray scale of the image). With proper correction for scanner nonlinearity, this image would appear uniform and straight under virtually all combinations of scan speed. scan direction, and scanner offset, as shown in Figure 5-4(b). [1] The resulting ratio of measurement

[1]Figure 5-4 (a) and (b) are plotted in the same gray scale, but Figure 5-4(b) is much flatter than

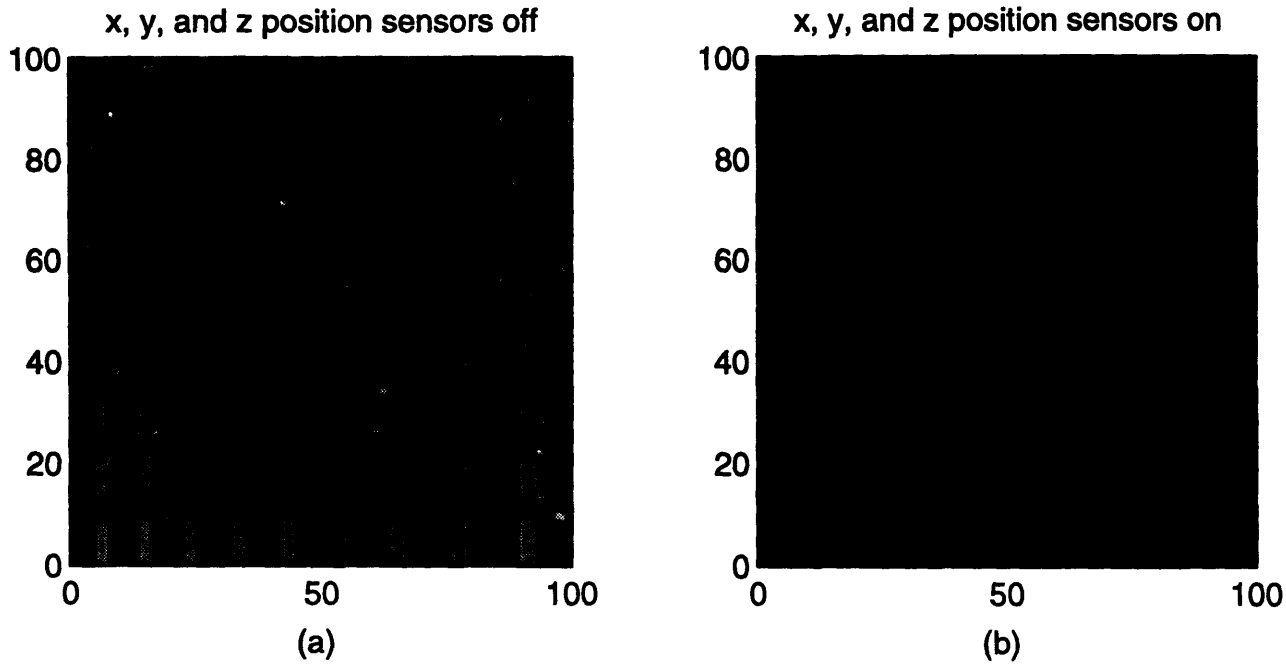


Figure 5-4: AFM Images of a Linear 9.9 μm-pitch grating : (a) Without x-y, and z Independent Position Sensors (b) With Position Sensors

error to measured distance was found to be smaller than 1%.

Figure 5-5 shows the result of scanning an optical flat sample (flat to within 20 nm over a 1" area) after fitting a first order plane to remove the tilt angle from the stage. Without z position sensor on and without any other correction, the bending error in vertical was found to result in a quadratic-like surface, and the difference from peak to valley is as large as 0.6836μm and the root mean square error (RMS) is 0.1285μm for the $90 \times 90\mu$m scanning area, while turning the position sensors on, the difference is reduced to 0.0807μm and RMS is 0.0117μm. Although the error was reduced, from Figure 5-5(b), we can still see another quadratic-like surface presented. It is mainly due to the noncollocated problem of the z position sensor. Since this is mainly a system error, further improvement is possible. If we also use the error map as the second correction, and fit a second-order surface to the data, the result is shown is Figure 5-5(c) and the difference is reduced to 0.0653μm with RMS is 0.0066μm. [2]

Figure 5-4 (a), so it might not be as clear as Figure 5-4(a) here.

[2]Here, Figure 5-5 (a), (b), and (c) are displayed in their own gray scales to show their surface features.

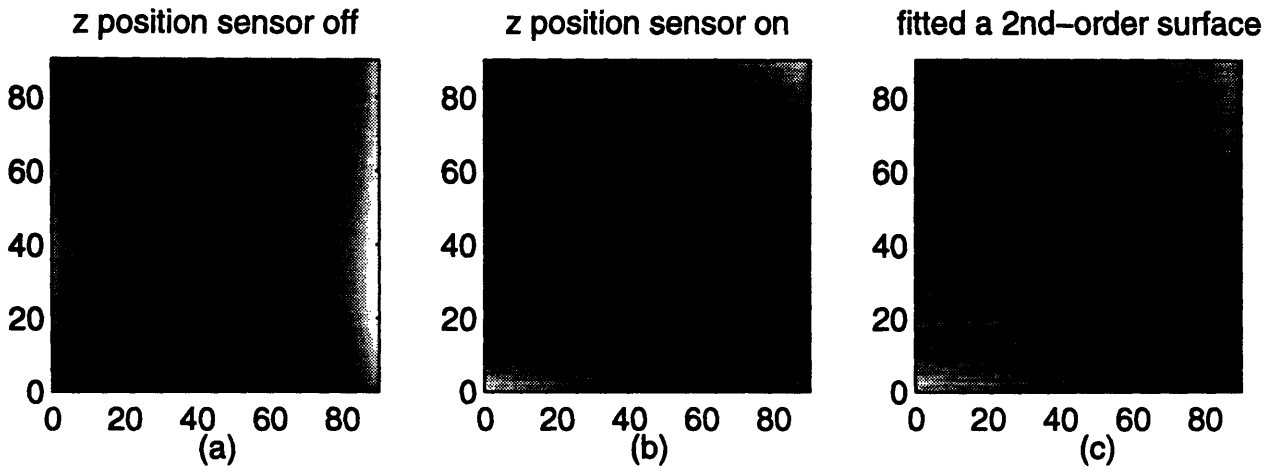


Figure 5-5: AFM Images of an Optical Flat Sample (a) Without z Position Sensor (b) With z Position Sensor (c) Fitting a Second-Order Surface to (b)

A few things to note here are, the coupling errors in the x and y directions are almost decoupled and highly quadratic. The main errors has the form below :

$$E(x, y) = A_2x^2 + A_1x + A_0 + B_2y^2 + B_1y + B_0$$

and the x error is larger than the y error. Also the average RMS roughness over a length of 90 μm in the x direction is found to be much smaller than the RMS roughness in the y direction (2.5 nm and 19.8 nm respectively) after correction. It means that the quadratic error map fits better in the x direction. These results might be due to the nonsymmetric geometric configuration of the scanner head design (i.e., error is not symmetrical in the y direction.) and the mounting position of the z position sensor (z position sensor is mounted along the x axis.)

5.2.2 Cantilever and Tip

The performance of SPM is limited by the quality of the tip used for probing the surface topographhpy. Different sites on the probe tip interact with the sample during the scan leads to a convolution of the sample features with the tip shape, and it will produce artifacts in the resulting images.

Tip convolution problems have been addressed in Chapter 3, and the "envelop

image analysis" method proposed by Keller is used to relieve the problem there. But as mentioned earlier, to deconvolute the tip shape from the false image, the tip shape must be well known first. Inappropriate estimate of the tip shape for deconvolution sometimes will induce even larger errors on the reconstructed images. It is not easy to get a quantitative measurement of the convolution errors , but one thing that should be kept in mind is, if the tip shape is more well-defined and has higher aspect-ratio , it is more possible to avoid the convolution/deconvolution errors.

Basically, there are three different types of microstyli commonly used in SPM : pyramidal silicon nitride tips, etched silicon tips, and electron-beam-deposited(EBM) tips. The pyramidal silicon nitride have a well-defined geometry with extremely smooth sidewalls with slopes of 54.7° relative to the cantilever. It's straight-line-like sidewall is good for the edge detection, also the tips produced from this method are quite uniform and the extra measurement for the tip shape can be reduced to minimum. One problem here is it's wide side angle. In order to scan the ultimate points of the sample, the sample has to be tilted 54.7°. It has physical dimensional problem in this instrument if a long range scanning is needed. Besides, it's large side angle will also lower the edge detection signal. (you can think of that this way, if the angle between the sample edge and the tip's side wall are larger, the edge is more obvious, and it is easier to identify the edge feature.)

The tip used in this instrument is the etched silicon tip. The etched silicon tips did not have such well-defined geometry, and they showed greater variation even within a single batch, this made the deconvolution hard to do. To eliminate this effect, a localized reconstruction method is used here. But the advantages of theses tips in comparison with the pyramidal tips described above are their higher aspect ratio and lower tip radius. A typical tip could have radius as small as 10 nm and the aspect ratio as high as 5 in the ultimate tip area.

Another type of tip commercially available is the EBI tip. It is produced by focusing the electron beam of a scanning electron microscope on the end of a conventional SPM tip covered first with a metal or carbon layer to create a conductive lever surface. The beam position is held fixed at the end point of the tip for several minutes

while electron beam deposition takes place. This results in a needle-like deposit which grows with irradiation time. The EBD tips offers an extremely high aspect ratio in combination with a small tip radius, smooth side walls and a fairly well-defined geometry. An added advantage of the EBD tips is the ability to characterize every tip in the electron microscope immediately after its fabrication. This allows the in-situ characterization of the various special tip shapes that can be produced by this method. If high accuracy is needed, the EBD tips might be a better choice.

Except to the tip shape, tips might also be flexed, distorted or even worn during scanning. Sometimes a double tip situation might also happen and these will all effect the scanning result. To guarantee the accuracy of scanning result, the frequent examination of the tip shape is needed.

5.3 Positioning Stages

As described in chapter 3, to transfer the data gathered from AFM to the sample's coordinate system, the equations below are applied :

$$(\mathbf{P}^A - \mathbf{P}^A_{C_{1org}}) = \mathbf{R}^A_{C_1}\mathbf{P}^{C_1} = \mathbf{R}^A_R\mathbf{R}^R_D\mathbf{R}^D_B\mathbf{R}^B_{C_1}\mathbf{P}^{C_1} \tag{5.1}$$

$$(\mathbf{Q}^A - \mathbf{Q}^A_{C_{2org}}) = \mathbf{R}^A_{C_2}\mathbf{Q}^{C_2} = \mathbf{R}^A_R\mathbf{R}^R_D\mathbf{R}^D_B\mathbf{R}^B_{C_2}\mathbf{Q}^{C_2} \tag{5.2}$$

where $\mathbf{P}^A_{C_{1org}}$ and $\mathbf{P}^A_{C_{2org}}$ are the position of the sample coordinate system $\{C\}$'s origin at position 1 and 2 with respect to the scanner coordinate system $\{A\}$, and are determined from the post processing described in chapter 3.

All rigid bodies have three rotational $(\epsilon_x, \epsilon_y, \epsilon_z)$ and three translational $(\delta_x, \delta_y, \delta_z)$ errors components. Those errors will cause the distortion to the reconstruction of the original three dimensional sample profile. To effectively control those errors, a homogeneous transformation matrix based error analysis was calculated.

For a small angular error, the higher order terms can be neglected. Then, the actual rotation matrix from the reference frame $\{R\}$ to the scanner frame $\{A\}$ will

be :

$$\mathbf{R}_R^A = \begin{bmatrix} 1 & \epsilon_z(A) & -\epsilon_y(A) \\ -\epsilon_z(A) & 1 & \epsilon_x(A) \\ \epsilon_y(A) & -\epsilon_x(A) & 1 \end{bmatrix}$$

where the errors are functions of the scanner's orientation. Everytime the tip is reloaded or the scanner is remounted, it's orientation will change a little bit, and $\epsilon_x(A)$, $\epsilon_y(A)$, and $\epsilon_z(A)$ correspond to it's roll, pitch, and yaw errors respectively.

Here, the translational errors $(\delta_x, \delta_y, \delta_z)$ is not taken into account. Basically, the stages and AFM's absolute position errors will not effect the determination of frame $\{C\}$'s origin in x and z direction since the tip is always brought to the sample's ultimate edge for scanning and frame $\{C\}$'s origin is determined from the relative position to frame $\{A\}$ by post-processing. The key thing here is the sample's orientation errors. But one thing to note is, the error in Y direction after flipping the sample over should be calculated to consider the Y shifting effect, although it might be able to correct that by image matching technique. To simplify the calculation, it is ignored for now and will be considered later.

The actual rotation matrix from x-y stages to reference frame $\{R\}$ is :

$$\mathbf{R}_D^R = \begin{bmatrix} 1 & -\epsilon_z(D) & \epsilon_y(D) \\ \epsilon_z(D) & 1 & -\epsilon_x(D) \\ -\epsilon_y(D) & \epsilon_x(D) & 1 \end{bmatrix}$$

where the errors are a function of the x-y stage's position and orientation.

The actual rotation matrix from the flip stage to the x-y stages is :

$$\mathbf{R}_B^D = \begin{bmatrix} 1 & -\epsilon_z(B) & \epsilon_y(B) \\ \epsilon_z(B) & 1 & -\epsilon_x(B) \\ -\epsilon_y(B) & \epsilon_x(B) & 1 \end{bmatrix}$$

and the errors are functions of the flip stage's position and orientation.

As to the sample's frame $\{C_1\}$, it is reached by rotating an angle θ_1 with respect

to $\hat{Y}_B$. So, in addition to the sample's orientation errors $(\epsilon_x(C), \epsilon_y(C), \epsilon_z(C))$, there is another rotational error $\delta\theta_1$. The actual rotation matrix from the sample's frame $\{C_1\}$ to the flip stage's frame $\{B\}$ will be :

$$\mathbf{R}^B_{C_1} = \begin{bmatrix} \cos(\theta_1+\delta\theta_1) & 0 & \sin(\theta_1+\delta\theta_1) \\ 0 & 1 & 0 \\ -\sin(\theta_1+\delta\theta_1) & 0 & \cos(\theta_1+\delta\theta_1) \end{bmatrix} \cdot \begin{bmatrix} 1 & -\epsilon_z(C) & \epsilon_y(C) \\ \epsilon_z(C) & 1 & -\epsilon_x(C) \\ -\epsilon_y(C) & \epsilon_x(C) & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos\theta_1 - \delta\theta_1\sin\theta_1 - \epsilon_y(C)\sin\theta_1 & \epsilon_x(C)\sin\theta_1 - \epsilon_z(C)\cos\theta_1 & \epsilon_y(C)\cos\theta_1 + \sin\theta_1 + \delta\theta_1\cos\theta_1 \\ \epsilon_z(C) & 1 & -\epsilon_x(C) \\ -\sin\theta_1 - \delta\theta_1\cos\theta_1 - \epsilon_y(C)\cos\theta_1 & \epsilon_z(C)\sin\theta_1 + \epsilon_x(C)\cos\theta_1 & -\epsilon_y(C)\sin\theta_1 + \cos\theta_1 - \delta\theta_1\sin\theta_1 \end{bmatrix}$$

where the errors are functions of the sample's orientation and the flip stage's rotation angle θ_1.

Similarly, at position 2:

$$\mathbf{R}^B_{C2} =$$

$$\begin{bmatrix} -\cos\theta_2 + \delta\theta_2\sin\theta_2 - \epsilon_y(C)\sin\theta_2 & \epsilon_z(C)\cos\theta_2 + \epsilon_x(C)\sin\theta_2 & -\epsilon_y(C)\cos\theta_2 + \sin\theta_2 + \delta\theta_2\cos\theta_2 \\ \epsilon_z(C) & 1 & -\epsilon_x(C) \\ -\sin\theta_2 - \delta\theta_2\cos\theta_2 - \epsilon_y(C)\cos\theta_2 & \epsilon_z(C)\sin\theta_2 - \epsilon_x(C)\cos\theta_2 & -\epsilon_y(C)\sin\theta_2 - \cos\theta_2 + \delta\theta_2\sin\theta_2 \end{bmatrix}$$

The actual sample position with respect to frame $\{A\}$ is represented by the following matrix multiplication :

$$\mathbf{P}^A_{actual} = \mathbf{R}^A_R\mathbf{R}^R_D\mathbf{R}^D_B\mathbf{R}^B_{C_1}\mathbf{P}^{C_1} + \mathbf{P}^A_{C_1org}$$

While in ideal case :

$$\mathbf{P}^A_{ideal} = \bar{\mathbf{R}}^A_R\bar{\mathbf{R}}^R_D\bar{\mathbf{R}}^D_B\bar{\mathbf{R}}^B_{C_1}\mathbf{P}^{C1} + \mathbf{P}^A_{C_1org}$$

where $\bar{\mathbf{R}}^A_R$, $\bar{\mathbf{R}}^R_D$, $\bar{\mathbf{R}}^D_B$, and $\bar{\mathbf{R}}^B_{C_1}$ are defined in section 3.6.1. So, the transformation

errors due to the orientation errors are given by

$$\begin{bmatrix} \Delta P_x^A \\ \Delta P_y^A \\ \Delta P_z^A \end{bmatrix} = \begin{bmatrix} P_x^A \\ P_y^A \\ P_z^A \end{bmatrix}_{actual} - \begin{bmatrix} P_x^A \\ P_y^A \\ P_z^A \end{bmatrix}_{ideal}$$

To simplify the comparison of the errors with the true sample profile, it is better to represent those errors in the sample's frame $\{C_1\}$:

$$\Delta \mathbf{P}^{C_1} = \mathbf{R}_A^{C_1} \Delta \mathbf{P}^A$$

Evaluating the equation above while neglecting second and higher order terms, we can get the respective error terms, and it can be written in the form of the error gains matrix as shown in Table 5.1. The error gains are the coefficients of the respective errors. For point P on the sample at position 1 and $\mathbf{P}^{C_1} = (P_x^{C_1}, P_y^{C_1}, P_z^{C_1})^T$, the corresponding x-directional error ΔP_x due to frame $\{A\}$'s roll error $\epsilon_x(A)$ will be $\Delta P_x = sin\theta_1 P_y \times \epsilon_x(A)$. The error gains matrix for the sample coordinate frame $\{C_2\}$ at position 2 is also shown in Table 5.2.

Y Shifting Errors

From the beginning, the Y shifting error of rotating the sample from position 1 to position 2 was neglected. It needs to be considered, too. Ideally, the sample's frame $\{C\}$'s origin should be on the frame $\{B\}$'s rotation axis, but it is hard to achieve, especially the sample edge is not a perfect straight line. Assume the sample's frame $\{C\}$'s origin position with respect to frame $\{B\}$ is :

$$\mathbf{P}_{Corg}^B = \begin{bmatrix} \delta_x(C) \\ P_{Corg,y}^B \\ \delta_z(C) \end{bmatrix}$$

	ΔPx	ΔPy	ΔPz
Frame {A} errors			
εx(A)	sin(θ1)Py	-sin(θ1)Px+cos(θ1)Pz	-cos(θ1)Py
εy(A)	-Pz	0	Px
εz(A)	cos(θ1)Py	-cos(θ1)Px-sin(θ1)Pz	sin(θ1)Py
Frame {D} errors			
εx(D)	-sin(θ1)Py	sin(θ1)Px-cos(θ1)Pz	cos(θ1)Py
εy(D)	Pz	0	-Px
εz(D)	-cos(θ1)Py	cos(θ1)Px+sin(θ1)Pz	-sin(θ1)Py
Frame {B} errors			
εx(B)	-sin(θ1)Py	sin(θ1)Px-cos(θ1)Pz	cos(θ1)Py
εy(B)	Pz	0	-Px
εz(B)	-cos(θ1)Py	cos(θ1)Px+sin(θ1)Pz	-sin(θ1)Py
Frame {C} errors			
εx(C)	0	-Pz	-Py
εy(C)	Pz	0	-Px
εz(C)	-Py	Px	0
Rotation Error			
δθ1	Pz	0	-Px

Table 5.1: Error Gains for Position 1

	ΔQx	ΔQy	ΔQz
Frame {A} errors			
εx(A)	sin(θ2)Qy	-sin(θ2)Qx-cos(θ2)Qz	cos(θ2)Qy
εy(A)	-Qz	0	Qx
εz(A)	-cos(θ2)Qy	cos(θ2)Qx-sin(θ2)Qz	sin(θ2)Qy
Frame {D} errors			
εx(D)	-sin(θ2)Qy	sin(θ2)Qx+cos(θ2)Qz	-cos(θ2)Qy
εy(D)	Qz	0	-Qx
εz(D)	cos(θ2)Qy	-cos(θ2)Qx+sin(θ2)Qz	-sin(θ2)Qy
Frame {B} errors			
εx(B)	-sin(θ2)Qy	sin(θ2)Qx+cos(θ2)Qz	-cos(θ2)Qy
εy(B)	Qz	0	-Qx
εz(B)	cos(θ2)Qy	-cos(θ2)Qx+sin(θ2)Qz	-sin(θ2)Qy
Frame {C} errors			
εx(C)	0	-Qz	Qy
εy(C)	Qz	0	-Qx
εz(C)	-Qy	Qx	0
Rotation Error			
δθ2	-Qz	0	Qx

Table 5.2: Error Gains for Position 2

where $P^B_{Corg,y}$ instead of $\delta_y(C)$ is used to represent that this value is not necessary to be a small number.

Then, after it rotates an angle θ_1 to position 1, the origin represented in the reference frame $\{R\}$ should be :

$$\mathbf{P}^R_{C_1org} = \mathbf{R}^R_D(\mathbf{R}^D_B(\mathbf{R}_1\mathbf{P}^B_{Corg}) + \mathbf{P}^D_{Borg}) + \mathbf{P}^R_{Dorg} \tag{5.3}$$

where $\mathbf{P}^R_{Dorg} = (P^R_{Dorg,x}, P^R_{Dorg,y}, P^R_{Dorg,z})^T$, $\mathbf{P}^D_{Borg} = (P^D_{Borg,x}, P^D_{Borg,y}, P^D_{Borg,z})^T$, and

$$\mathbf{R}_1 = \begin{bmatrix} \cos(\theta_1 + \delta\theta_1) & 0 & \sin(\theta_1 + \delta\theta_1) \\ 0 & 1 & 0 \\ -\sin(\theta_1 + \delta\theta_1) & 0 & \cos(\theta_1 + \delta\theta_1) \end{bmatrix}$$

Again, those are not required to be small numbers. And

$$\mathbf{P}^R_{C_2org} = \mathbf{R}^R_D(\mathbf{R}^D_B(\mathbf{R}_2\mathbf{P}^B_{Corg}) + \mathbf{P}^D_{Borg}) + \mathbf{P}^R_{Dorg} \tag{5.4}$$

where

$$\mathbf{R}_2 = \begin{bmatrix} \cos(\theta_2 + \delta\theta_2) & 0 & \sin(\theta_2 + \delta\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2 + \delta\theta_2) & 0 & \cos(\theta_2 + \delta\theta_2) \end{bmatrix}$$

Then the shifting of the origin is :

$$\Delta^R_{shifting} = \mathbf{P}^R_{C_2org} - \mathbf{P}^R_{C_1org} \tag{5.5}$$

If we neglect the higher order terms, also, except the flip stage, every stage is fixed at the same position and orientation for sample at position 1 and 2, then

$$\Delta^R_{shifting} = \mathbf{R}^R_D\mathbf{R}^D_B(\mathbf{R}^B_{C_2} - \mathbf{R}^B_{C_1})\mathbf{P}^B_{Corg}$$

But usually, when the sample is rotated from position 1 to position 2, it is needed to move the x stage to close the sample edge to the tip, so the error terms associated

with frame $\{D\}$ will not the same as in Equations (5.3) and (5.4). So those equations should be adjusted to :

$$\mathbf{P}^R_{C_1org} = \mathbf{R}^R_{D_1}(\mathbf{R}^{D_1}_B \mathbf{R}^B_{C_1} \mathbf{P}^B_{Corg} + \mathbf{P}^{D_1}_{Borg}) + \mathbf{P}^R_{D_1org} \tag{5.6}$$

where $\mathbf{P}^R_{D_1org} = (P^R_{D_1org,x}, P^R_{D_1org,y}, P^R_{D_1org,z})^T$ and $\mathbf{P}^{D_1}_{Borg} = (P^{D_1}_{Borg,x}, P^{D_1}_{Borg,y}, P^{D_1}_{Borg,z})^T$. And

$$\mathbf{P}^R_{C_2org} = \mathbf{R}^R_{D_2}(\mathbf{R}^{D_2}_B \mathbf{R}^B_{C_2} \mathbf{P}^B_{Corg} + \mathbf{P}^{D_2}_{Borg}) + \mathbf{P}^R_{D_2org} \tag{5.7}$$

where $\mathbf{P}^R_{D_2org} = (P^R_{D_2org,x}, P^R_{D_2org,y}, P^R_{D_2org,z})^T$ and $\mathbf{P}^{D_2}_{Borg} = (P^{D_2}_{Borg,x}, P^{D_2}_{Borg,y}, P^{D_2}_{Borg,z})^T = \mathbf{P}^{D_1}_{Borg}$ [3]

Neglecting the second and higher order terms, then :

$$\Delta^A_{shifting} \simeq \Delta^R_{shifting} = \mathbf{P}^R_{C_2org} - \mathbf{P}^R_{C_1org}$$

the shifting error in the y direction will be :

$$\Delta_{shifting,y} = P^R_{C_2org,y} - P^R_{C_1org,y}$$

or

$$\Delta_{shifting,y} = P^R_{D_2org,y} - P^R_{D_1org,y} + (\epsilon_x(D_1) - \epsilon_x(D_2))P^D_{Borg,z} + \textit{higher order terms.}$$

As we can see, the errors from the rotation axis errors or the sample's orientation errors are all second order or higher terms. The real factors dominating the y shifting error is the x stage's roll error $\epsilon_x(D)$ and the flip stage's clearance in the y direction $(P^R_{D_2org,y} - P^R_{D_1org,y}.)$ To reduce the y shifting error, in addition to reduce the flip stage's clearance, the sample edge offset in the x direction at position 1 and 2 had better be small enough, so the movement of the x stage is not necessary or can be compensated by use of the piezotube offset, then $\epsilon_x(D_1) = \epsilon_x(D_2)$ since the stage

[3]Since frame $\{B\}$ is always fixed with respect to frame $\{D\}$ according the definition in chapter 3. then $\mathbf{R}^{D_1}_B = \mathbf{R}^{D_2}_B = \mathbf{R}^D_B$ and $\mathbf{P}^{D_1}_{Borg} = \mathbf{P}^{D_2}_{Borg} = \mathbf{P}^D_{Borg}$

didn't move during the two scanning sessions.

Finally, if we also take into account the sample tip's identification errors, then $\Delta_{edge,x}$ and $\Delta_{edge,z}$ should be included to the total errors at position 1 and 2 where $\Delta_{edge,x}$ and $\Delta_{edge,z}$ represent the corresponding edge identification errors in the x and z direction.

5.4 Thermal Drift

The relative start and stop x-y positions of the tip and sample should remain the same during the scan. However, due to thermal fluctuations, the image appears to drift away and/or become distorted in the direction of the drift. That is, if the scanning is repeated many times at the same position, the resulting images tend to shift in some direction. Thermal drift may be from the piezoelectric tube or from the stages, and might cause problems at low scan speeds since it takes longer time to finish the scan and the thermal drift effect might appear to be significant. It is possible to use software correction to remove the thermal drift, but usually the thermal drift depends heavily on the temperature gradients in the scan area, so it is difficult to correct that on time.

A simple test has been done on the prototype of this instrument by scanning the 1 μm pitch grating at the same location continuously and recording the drift in position. The thermal drift is calculated as :

$$thermal\ drift = \frac{|\Delta X(or\ \Delta Y, \Delta Z)|}{(elapsed\ time)}$$

and the tested result was roughly 12 nm/min in the x direction and 4 nm/min in the y direction.

To eliminate the thermal drift error, material with low coefficients of thermal expansion and acceptable dimensional stability properties should be chosen and the temperature control is needed. A faster scanning rate will help, too, since it takes less time to finish a scan.

5.5 Summary

In this chapter, the main error sources from this instrument were reviewed and some of them are quantitatively measured or estimated. A qualitative analysis of the errors from the positioning stages was given here, too.

Chapter 6

Conclusion

The purpose of this thesis work was to build a prototype system of a nano-precision three dimensional profilometer. The task has been accomplished by designing a customized Atomic Force Microscope with a high precision sample stage.[7] Data processing algorithms and techniques particularly required for this kind of resolution were developed to reconstruct the original profile. The software for this three dimensional profilometer is written, and is emphasized on the modularization and flexibility for the further development. Also the user interface is provided for ease of use. The system's errors sources are also discussed.

For a measurement instrument, there are three important parameters to tell how good it is, they are accuracy, repeatability, and resolution. Accuracy is how close the measurement to the true value, repeatability is how consistent the results are on repeated measurements, and resolution is the smallest quantity the instrument can measure or distinguish. These parameters for this instrument has been tested and the results can be found in [18].

Although this system's performance is proved to be satisfactory, but still need to be improved in several areas. First, the system's dynamic effects should be analyzed or modeled to understand their effects on the data. Basically, in this thesis work, all the data analyzed are assumed to be in the ideal steady states. Actually, this needs to be verified. While the scanning tip of the AFM is moving on the sample surface, there are many dynamic effects involved. The cantilever-like spring might

be distorted due to the different forces acting on it and while the tip is just falling off the sample's ultimate edge, it will cause vibration, too. The sample and the tip's interaction might also distort the image.

Another problem needed to investigate is the scanning speed. This is a common problem in all of the scanning probe microscopes. Although SPM's invention has greatly improved the measurement resolution, one main drawback in SPM is the time to acquire a certain image. In light and electron microscopes, the time to acquire an image is independent of the image size. This is not the case in SPM. The maximum scanning speed of a SPM is mainly limited by its resonance frequency. The scan process can be understood as excitation of the Z-axis of the SPM, because the feed-back loop, that control the Z-axis, has to follow the surface features of the sample. Therefore the excitation frequency of the Z-axis is directly proportional to the chosen scan-speed. The result of this dependency is the fact that small microscopes with a small scanner and a small scan range may use high scan-speed. More rigorous mechanical system and control electronics limitations are necessary for higher scanning speed.

Again, the prototype of this instrument has been proven to have better performance than the traditional stylus-type profilometer, its three dimensional ability also breaks through the limit of the optical and electron microscopes. The different types of scanning probe microscopes could make the SPM-based profilometer even more diverse and could be used in a wide variety of disciplines.

Appendix A

Coordinate Transformation

To represent the relative position and orientation of a rigid body in three- dimensional space with respect to a given coordinate system, a 4×4 matrix is needed. A homogeneous transformation from the coordinate system $\{B\}$ to the coordinate system $\{A\}$ is denoted by a 4×4 homogeneous transformation matrix (HTM) T_B^A:

$$\mathbf{T}_B^A = \begin{bmatrix} \mathbf{R}_B^A & \vec{\mathbf{P}}_{Borg}^A \\ 0 \;\; 0 \;\; 0 & 1 \end{bmatrix} \tag{A.1}$$

where $\mathbf{R}_B^A$ is a 3×3 rotation matrix describing $\{B\}$ relative to $\{A\}$, and

$$\mathbf{R}_B^A = \begin{bmatrix} \hat{\mathbf{X}}_B^A & \hat{\mathbf{Y}}_B^A & \hat{\mathbf{Z}}_B^A \end{bmatrix}$$

where $\hat{\mathbf{X}}_B$, $\hat{\mathbf{Y}}_B$, and $\hat{\mathbf{Z}}_B$ are the unit vectors giving the principal directions of coordinate system $\{B\}$. When written in terms of the coordinate system $\{A\}$, they are called $\hat{\mathbf{X}}_B^A$, $\hat{\mathbf{Y}}_B^A$, and $\hat{\mathbf{Z}}_B^A$ respectively. $\vec{\mathbf{P}}_{Borg}^A$ is the position vector of the origin of frame $\{B\}$ with respect to frame $\{A\}$.

For a position defined with respect to frame $\{B\}$ by the vector $\vec{\mathbf{P}}^B$(see Figure A-1), its description with respect to frame $\{A\}$ is :

$$\vec{\mathbf{P}}^A = \mathbf{R}_B^A \vec{\mathbf{P}}^B + \vec{\mathbf{P}}_{Borg}^A \tag{A.2}$$

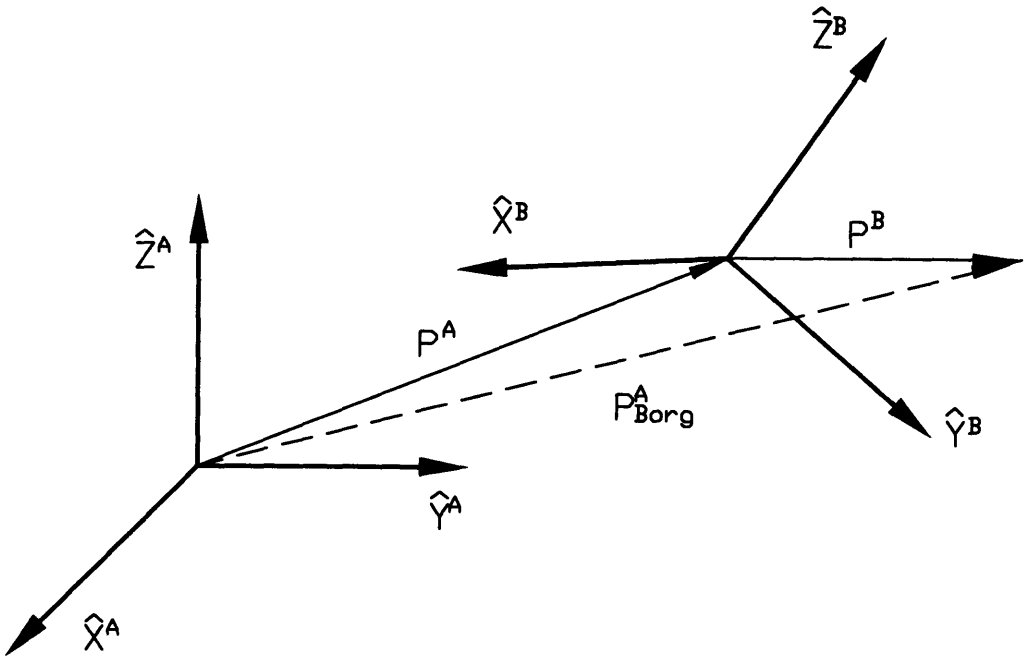


Figure A-1: Coordinate Transformation

or

$$\begin{bmatrix} \vec{\mathbf{P}}^A \\ 1 \end{bmatrix} = \mathbf{T}_B^A \begin{bmatrix} \vec{\mathbf{P}}^B \\ 1 \end{bmatrix} \tag{A.3}$$

If there are more than two coordinate systems involved, for example, a coordinate system $\{C\}$ as the intermediate frame between $\{A\}$ and $\{B\}$, then they have the relationship below :

$$\mathbf{T}_B^A = \mathbf{T}_C^A \mathbf{T}_B^C \tag{A.4}$$

Appendix B

Optimal Sample Tilt Angle for Scanning

Due to the geometrical restriction, certain regions of the sample's ultimate edge area are inaccessible, so the image contains no information about them. As shown in Figure B-1, the envelop image analysis method could only reconstruct the regions touched by the scanning tip surface and connect them with a segment of the tip surface. So the sample's edge that we are trying to identify will be the farthest point of the sample to be traced by the tip, or the intermediate point between the sample surface and the scanning tip surface formed by a group of tip surface.

Basically, there are two factors that will effect the range of the sample edge area that can be recovered. One is the sample's tilt angle, another is the scanning tip's shape.

Assume the sample's edge could be modeled as an arc. If the sample is tilted an angle α_1 with respect to horizontal at one side and α_2 at the other side, and the scanning tip has an angle β_1 and β_2 with respect to vertical at both sides of it's steepest sidewall. As shown in Figure B-1, the ultimate point the scanner tip can trace will have an angle $\theta_1 = \alpha_1 - \beta_1$ and $\theta_2 = \alpha_2 - \beta_2$.

To recover the three dimensional profile from two separate surface topography, one assumption is made that the sample edge points identified from both sides are the same point. It means, $\theta_1 = -\theta_2$, and usually it is more convenient to take $\theta_1 = \theta_2 = 0$.

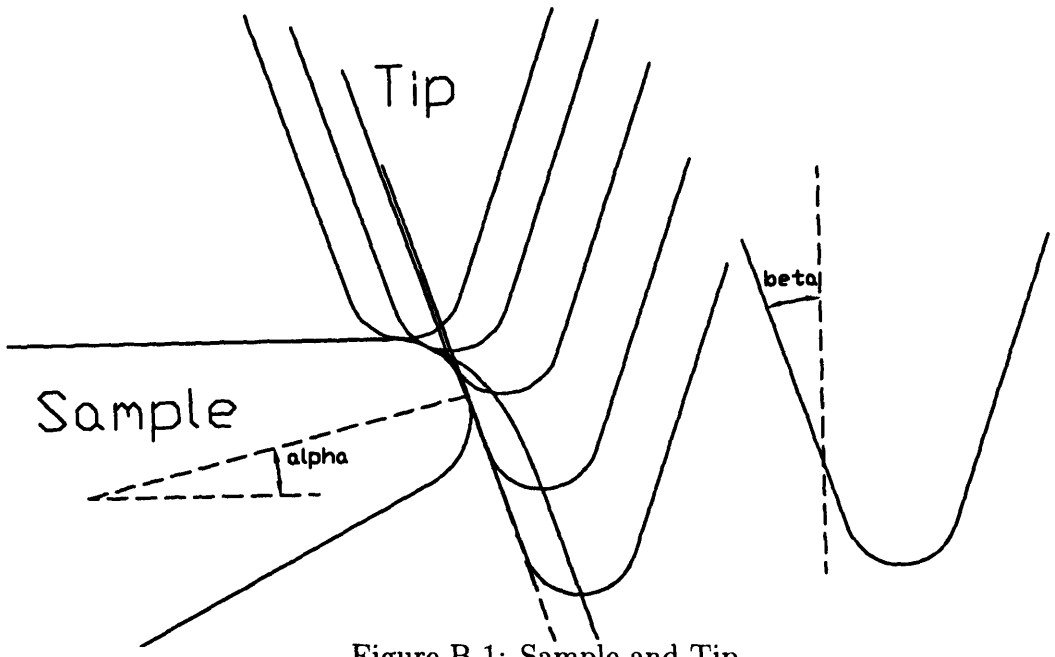


Figure B-1: Sample and Tip

In this experiment, the scanning tip's sidewall angle $\beta_1 \cong \beta_2 \cong 10°$. To minimize the errors from the edge points position, the sample tilt angle should be set to 10°.

Appendix C

Software Functions

C.1 Programming Environment

The software was developed under Borland C/C++ 4.5's ide development tools, and the operating system is MS-Windows 3.1.

The software can mainly be divided into several projects as below : [1]

1. afmproj.ide : the Windows execution file project. It deals with user interface and communicates with other programs. It contains,

   - afmproj5.c : C source code.
   - afmproj.rc : Windows resource script file.
   - afmproj.def : Windows module definition file.
   - idapi.lib : Library for database access.
   - sela_dlg.rc : Windows resource script file for data selection windows.
   - the related header files (*.h).

   Output : afmproj.exe

2. matrxlib.ide : a DLL project for matrix-manipulation-related functions.

[1] Refer to Borland C++ for Windows Manual and the related Windows programming reference for further explanation.

- matrxlib.c : C source code.
- matrxlib.def : Windows module definition file.
- the related header files (*.h).

Output : matrxlib.dll

3. afmlib.ide : a DLL project for profilometer-related functions.

   - afmlib.c : C source code.
   - afmlib.def : Windows module definition file.
   - the related header files (*.h).

   Output : afmlib.dll

4. filesel.ide : a DLL project for file selection related functions

   - filesel2.c : C source code
   - filesel2.rc : Windows resource script file.
   - filesel2.def : Windows module definition file.
   - the related header files (*.h).

   Output : filesel.dll

Because the AFM's scanning parameters like scanning rate, scanning size, datafile names ... etc, are stored in Paradox-readable database files, to access those data, Borland's database engine provides some useful DLL files to facilitate the programming. The DLL files needed to run this software include :

- dbg.dll : database related DLL
- idapi01.dll : database-related DLL
- idasci01.dll : database-related DLL
- idabat01.dll : database-related DLL
- iddbas01.dll : database-related DLL
- idodbc01.dll : database-related DLL

- idpdx01.dll : database-related DLL
- idqry01.dll : database-related DLL
- idr10009.dll : database-related DLL
- ild01.dll : databse-related DLL

C.2 Software Operation

The software operation instruction is addressed in depth here.

First, the procedure to install and run the program is as following :

1. copy file "afmproj.exe" and the related DLL files to the same direction.
2. create a root directory "\imgfiles" and put the database file "scantbl.db" and all of the corresponding image files in this directory.
3. run the program under MS-Windows 3.1 or later.

Once the program is run, a main window will pop up as shown in Figure C-1, it contains view A, view B, zoom A, zoom B, view 2D subwindows, and some other buttons and parameters as shown in this figure.

some important functions of this programs are addressed below :

- *File/Open* : pop up a sample selection window as shown in Figure C-2. Select the data files needed for analysis by scrolling the light bar either in "Sample name selections" or "Comments" field. The images will be loaded and displayed in view A and view B windows.
- *File/Save* : save the 2D average thickness data. The default file name is "afm.dat".
- *File/Save 3D* : save the 3D profile data. So far, 3D profile display function has not been implemented in this program yet, but it can save the 3D data and present the 3D profile in other programs like MATLAB. Example commands of

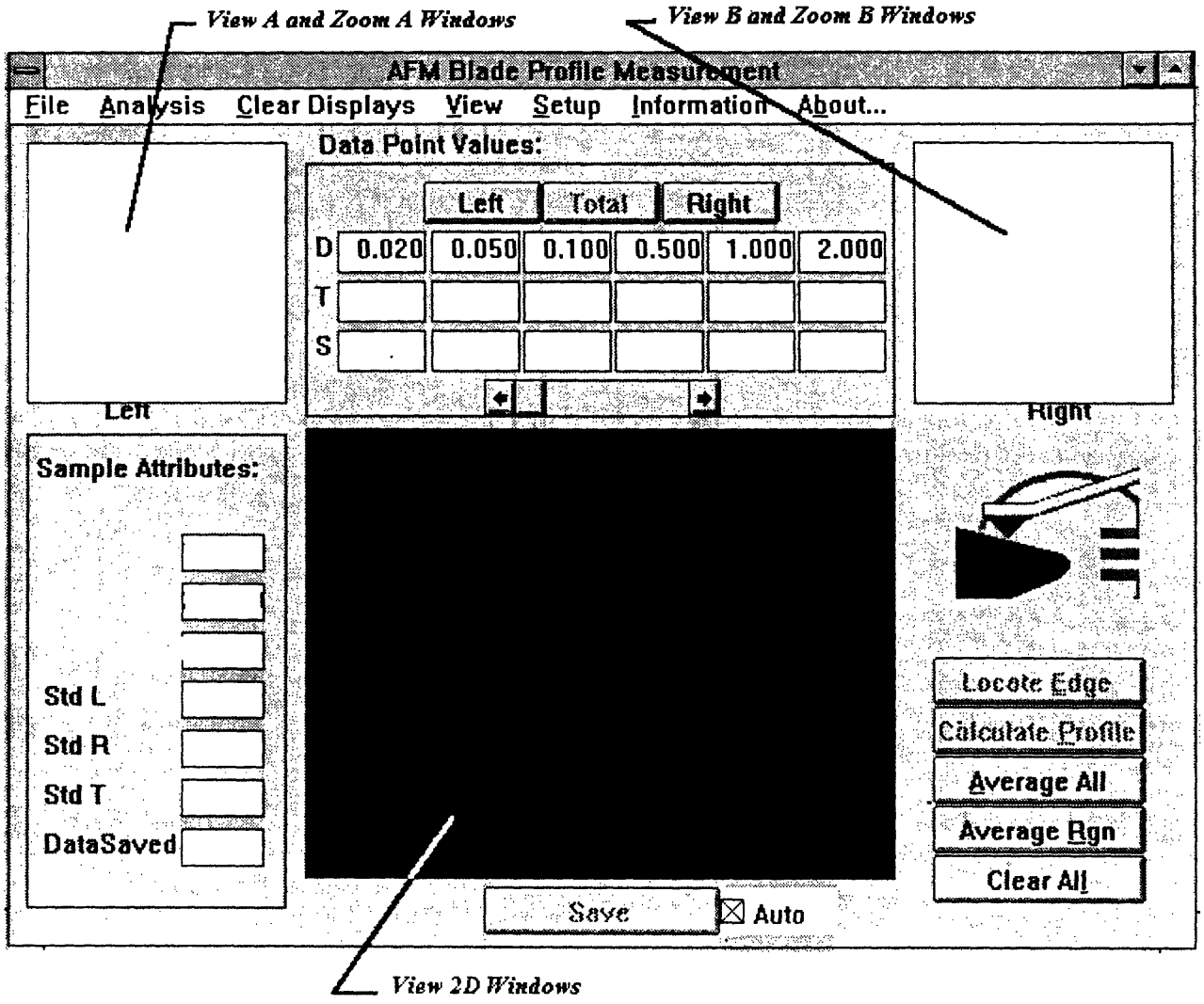


Figure C-1: user interface

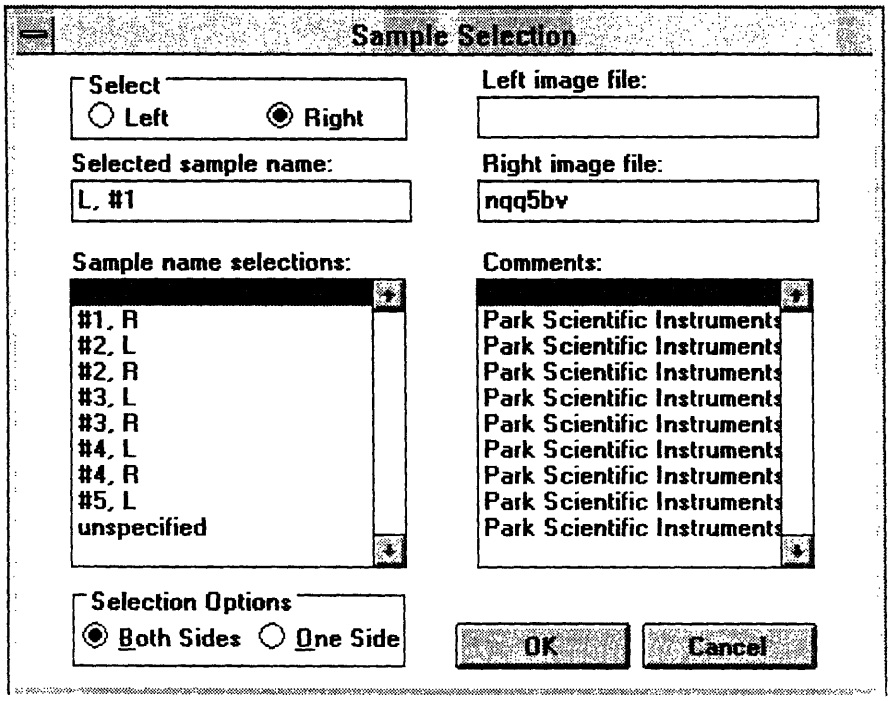


Figure C-2: File Selection Window

showing the 3D profile from MATLAB is described below, here, the data saved is named "afm3d.dat", and the original image size is assumed to be 256 × 256.

≫load afm3d.dat

≫[m,n]=sizes(afm3d);

≫XX=reshape(afm3d(:,1), m/256, 256))';

≫YY=reshape(afm3d(:,2), m/256, 256))';

≫ZZA=reshape(afm3d(:,3), m/256, 256))';

≫ZZB=reshape(afm3d(:,4), m/256, 256))';

≫mesh(ZZA, XX, -YY);

≫hold on;

≫mesh(ZZB, XX, -YY);

- *Analysis* : contain *Filter*, *Reconstruction*, *2D profile*, *2D Avg profile* and *3D profile* selections. Choose the appropriate one for analysis. Based on the process states, some of the functions might be disabled.

- *Clear Displays* : clear the corresponding windows.

- *View* : change the data shown in the subwindows.

- *Setup* : pop up a setting window as shown in Figure C-3 to change the parameters setting in this program. The setting can be changed permanently, temporarily or can be canceled by pressing *Save*, *OK(not save)* or *Cancel* respectively. [2]

  - *Filter* : set the filtering method.

  - *Recon_Edge* : set the edge identification method

  - *Tilt angle* : set the sample's tilt angle

  - *First derivative* : set the parameters for the first derivative based edge identification method.

    To accelerate the edge identification and reduce the possibility of misidentification, the edge to be searched is restricted to a small region of the sample edge area by two parameters *threshold_ ratio* and *offset* in the program as shown in Figure C-4. The parameter *slope1* set the criterion of the slope for edge detected.

  - *Second derivative* : set the parameters for second derivative based edge identification method.

- *Information* : pop up the information window which shows the image data's basic information. (See Figure C-5)

- *Locate Edge* Button : find the sample edge and show the corresponding images in the view windows

- *Calculate Profile* Button : calculate the 2D profile and show that in the view 2D window.

[2] To restore the default value, quit the program and delete the file "afmparam.def". The default value will be reinstalled automatically next time the program is run.

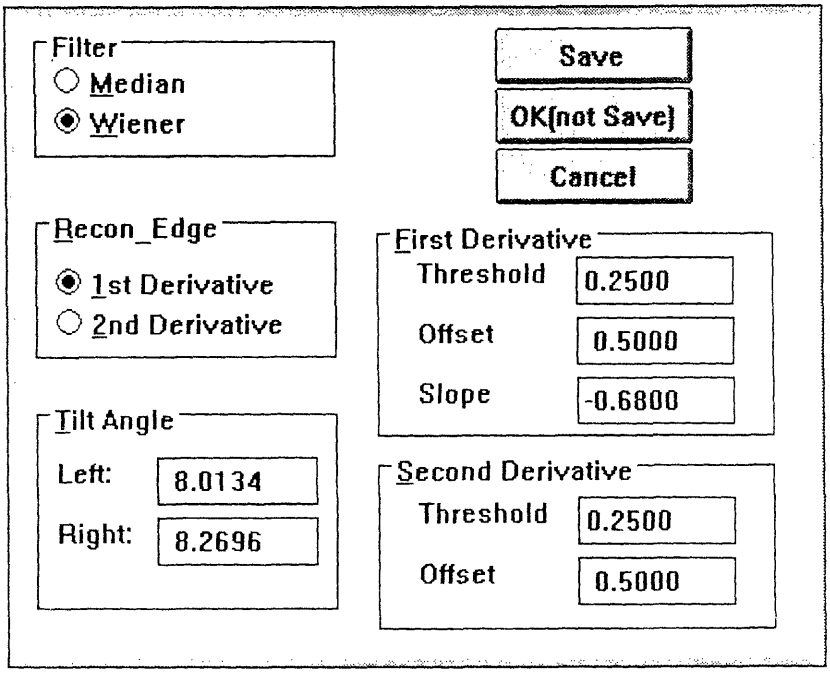


Figure C-3: Setting Window

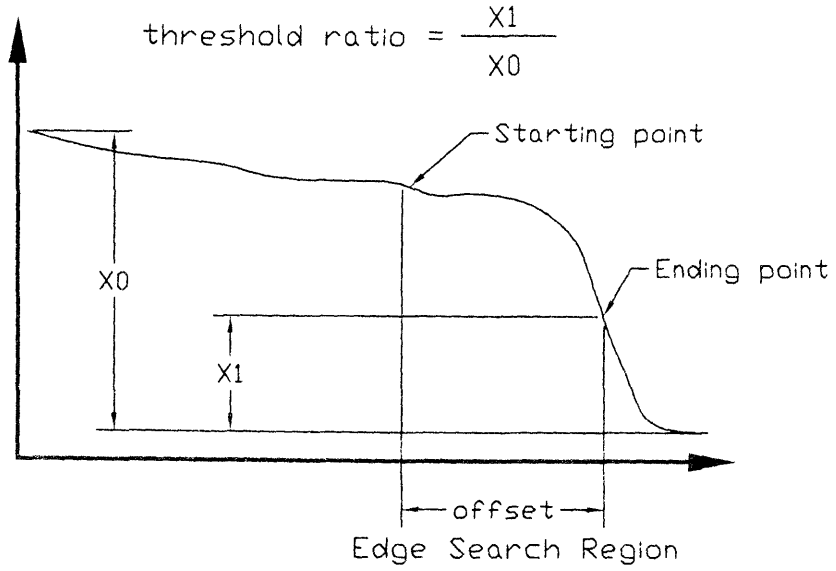


Figure C-4: Parameters for Edge Identification

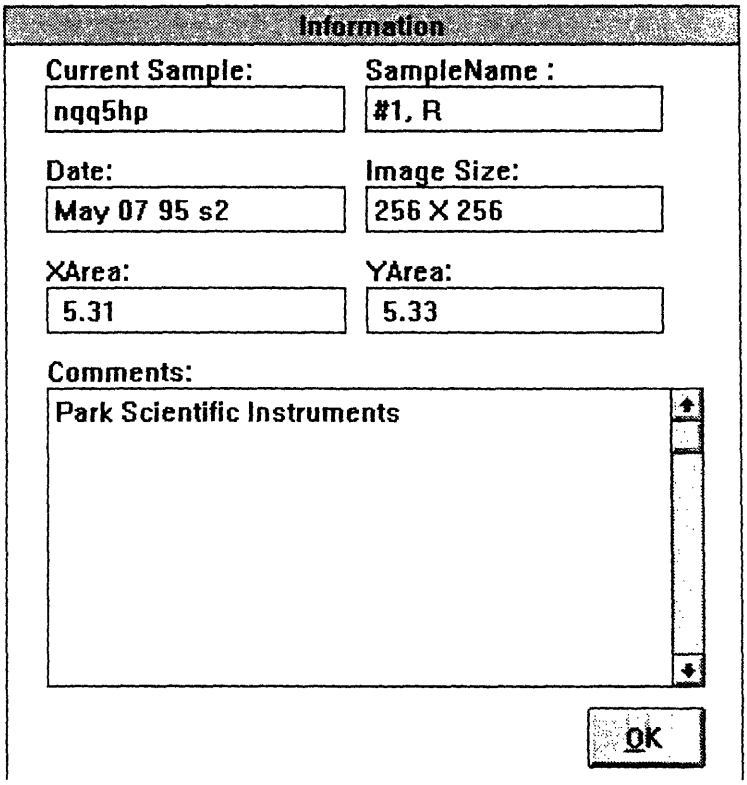


Figure C-5: Information Window

- *Average All* button : calculate the average thickness of the sample
- *Average Rgn* button : calculate the average thickness of a region of the sample. The region is defined as follows :

  press the "shift" key while clicking the right button of the mouse on the view A or view B windows and drag the mouse, the region is defined once the mouse button is released.
- *Clear All* button : clear all the subwindows.
- *Save* button : the same as the *file/save*. If *Auto* is checked, the program will automatically save the data everytime a new 2D profile is generated.
- *Left/Total/Right* button : switch the parameters shown in the screen according to which button is chosen.
- the view and zoom windows can be toggled by double clicking the right button of the mouse on the view or zoom windows.

- once the edge is identified, each cross section of the sample can be displayed on the view 2D windows by clicking the left button of the mouse on the view A or B windows and move the mouse cursor. Click again to disable this function.

Appendix D

List of Program Codes

D.1 afmproj.ide

D.1.1 afmproj5.def

```
NAME          afmproj5
DESCRIPTION   'AFM Blade Measurement Windows application'
EXETYPE       WINDOWS
STUB          WINSTUB.EXE'
CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE MULTIPLE
HEAPSIZE      2048
STACKSIZE     19600
EXPORTS       WNDPROC
              CHILDPROCA                                              10
              CHILDPROCB
              CHILDPROCMAIN
              SetupDlg
              SaveAsDlg
              Save3DDlg
IMPORTS       CALLFILEDLG = FILESEL.CALLFILEDLG
              SPLITPATHFILE = FILESEL.SPLITPATHFILE
              COMBINEPATHFILE = FILESEL.COMBINEPATHFILE
              NEW_MATRIXF = MATRXLIB.NEW_MATRIXF
              OLD_MATRIXF = MATRXLIB.OLD_MATRIXF                      20
              NEW_MATRIXI = MATRXLIB.NEW_MATRIXI
              OLD_MATRIXI = MATRXLIB.OLD_MATRIXI
              MEDFILT2 = AFMLIB.MEDFILT2
              WIENER2 = AFMLIB.WIENER2
              REC_EDG1 = AFMLIB.REC_EDG1
              REC_EDG2 = AFMLIB.REC_EDG2
              FLIPLR = MATRXLIB.FLIPLR
              COMBINE2D = AFMLIB.COMBINE2D
              COMBINE3D = AFMLIB.COMBINE3D
              AVERAGE = MATRXLIB.AVERAGE                              30
```

```
TABLE = MATRXLIB.TABLE
STD = MATRXLIB.STD
FVEC_SUB_FVEC = MATRXLIB.FVEC_SUB_FVEC
```

D.1.2 afmproj5.h

```
/*************************************************************
        Header File : afmproj.h
        Description : header files for afmproj.c

        Written by C.J. Chiu
        Dec. 1994
 *************************************************************/
#define SIDEA                   0
#define SIDEB                   1

#define IDM_INFORMATIONA        1
#define IDM_INFORMATIONB        4

#define IDM_QUIT                10

#define IDM_OPENIMAGEA          101
#define IDM_OPENIMAGEB          102
#define IDM_OPENPROJECT         103
#define IDM_SAVEAS              104
#define IDM_SAVE                105
#define IDM_SAVE3D              106
#define IDM_FILECLOSE           107

#define IDM_FILTER              201
#define IDM_SETUP               202
#define IDM_RECONSTRUCT         204
#define IDM_2DPROFILE           207
#define IDM_2DAVGPROFILE        208
#define IDM_3DPROFILE           209

#define ID_MEDIANFILTER         220
#define ID_WIENERFILTER         221
#define ID_RECONS1              230
#define ID_RECONS2              231
#define ID_TILTA                240
#define ID_TILTB                241
#define ID_THRESHOLD1           250
#define ID_OFFSET1              251
#define ID_SLOPE1               252
#define ID_THRESHOLD2           255
#define ID_OFFSET2              256
#define ID_SETUPSAVE            260


#define IDM_CLEARFRAMEA         301
#define IDM_CLEARFRAMEB         302
```

```
#define IDM_CLEARMAINFRAME 304
#define IDM_CLEARALL       305

#define IDM_VIEWRAWA       401
#define IDM_VIEWRAWB       402
#define IDM_VIEWFILTEREDA  403
#define IDM_VIEWFILTEREDB  404
#define IDM_VIEWRECONSTRUCTA 405
#define IDM_VIEWRECONSTRUCTB 406
#define IDM_VIEW2DPROFILE  410
#define IDM_VIEW2DAVGPROFILE 411
#define IDM_VIEW3DPROFILE  412

#define IDM_HELPABOUT      503

#define INFO_CANCEL_BUTTON 600
#define INFO_OK_BUTTON     605


#define LOCATEDGE_BUTTON   1001
#define CALCPROFILE_BUTTON 1002
#define CLEARALL_BUTTON    1003
#define AVGALL_BUTTON      1004
#define AVGRGN_BUTTON      1005
#define LEFT_BUTTON        1006
#define TOTAL_BUTTON       1007
#define RIGHT_BUTTON       1008
#define SAVE_BUTTON        1009

#define PI 3.1415962

#define AUTOSAVE_BUTTON    1010
#define ID_FILENAME        1011
#define SCROLLBAR          1012
#define ID_FILENAME3D      1013

#define ATTRIBUTE0         5000
#define ATTRIBUTE1         5001
#define ATTRIBUTE2         5002
#define ATTRIBUTE3         5003
#define ATTRIBUTE4         5004
#define ATTRIBUTE5         5005
#define ATTRIBUTE6         5006

#define DDATAPOINT1        5101
#define TDATAPOINT1        5201
#define SDATAPOINT1        5301
#define DATAPTGROUP        5501
#define ATTRBGROUP         5502

#define DLI_CANCELBUTTON   6000

#define S_PROGRAMCAPTION      5
```

```
#define S'PRINTDRIVERPROBLEM   7

#define IMGSIZE                65536


#define CURSAMPLE'EF           7000
#define SESSION'EF             7005
#define DATE'EF                7010
#define TIPSHARPNESS'EF        7015                                    110
#define TIPRADIUS'EF           7020
#define IMAGESIZE'EF           7025
#define COMMENTS'MEF           7030

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);
long FAR PASCAL ChildProcA (HWND, WORD, WORD, LONG);
long FAR PASCAL ChildZoomA (HWND, WORD, WORD, LONG);
long FAR PASCAL ChildProcB (HWND, WORD, WORD, LONG);
long FAR PASCAL ChildZoomB (HWND, WORD, WORD, LONG);
void OutBlock(HWND, POINT, POINT);                                     120
void InBlock(HWND, POINT, POINT, int);
long FAR PASCAL ChildProcMain (HWND, WORD, WORD, LONG);
long FAR PASCAL ChildInfoProc (HWND, WORD, WORD, LONG);

HGLOBAL LoadRawData(int hFileIn, HGLOBAL hgMemRaw);
void LoadMatrixF(int row, int column, MatrixF 'huge * A, HGLOBAL hgMemRaw,
                 float *maxA, float *minA);
void DisplayFile(HDC hdcMem, MatrixF 'huge *A, float maxA, float minA,
                 int row, int column);
BOOL FAR PASCAL 'export SaveAsDlg(HWND, WORD, WORD, LONG);             130
BOOL FAR PASCAL 'export SetupDlg(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL 'export Save3DDlg(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL 'export AboutDlgProc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL 'export SelectA'DlgProc (HWND, UINT, UINT, LONG);
void TableOpen (HWND hInformation);
```

D.1.3 afmproj5.c

```
/*****************************************************
  afmproj.c AFM Blade Measurement Application
  Note : the table : "scantbl.db" and related image data files
  should be placed under c:"imgfiles

  Written by C.J. Chiu.
  May, 1995
  *****************************************************/

#include ¡windows.h¿   /* window's header file */                      10
#include ¡direct.h¿    /* for getcwd() function */
#include ¡string.h¿      /* needed for Strtok() function */
#include ¡commdlg.h¿
#include ¡stdlib.h¿
#include ¡windowsx.h¿  /* window macro */
```

```
#include ¡math.h¿
#include "filesel2.h"  /* header file for dll */
#include "matrxlib.h"  /* header file for matrix operations */
#include "afmlib.h"    /* header file for afm-related functions */
#include "afmproj5.h"  /* header file for this program */
#include "snipit.h"    /* header for IDAPI / BDE */
#include "sela_dlg.h"

/* Global Variables */

/* Logical Palette global variables */
HPALETTE      hOldPal, hOldPal2, hPal=NULL;
LPLOGPALETTE lPal;

/* windows handles */
HWND  hStaticRectMain, hWnd;
HWND  hStaticRectA, hStaticRectB, hInformation, hZoomA, hZoomB;

/* program states variable */
BOOL   bOpenA = FALSE, bOpenB = FALSE;
BOOL   bFilteredA = FALSE, bFilteredB = FALSE;
BOOL   bReconstructedA = FALSE, bReconstructedB = FALSE;
BOOL   bCombine2D=FALSE, b2DAvgProfile=FALSE, bCombine3D=FALSE;
BOOL   bZoomA = FALSE, bZoomB = FALSE;
BOOL   bViewRawA = FALSE, bViewRawB = FALSE;
BOOL   bViewFilteredA = FALSE, bViewFilteredB = FALSE ;
BOOL    bViewReconstructedA = FALSE, bViewReconstructedB = FALSE;
BOOL   bViewMain=FALSE, bView2DProfile = FALSE;
BOOL   bView2DAvgProfile = FALSE, bView3DProfile = FALSE;
BOOL   bDataFile=FALSE, bSaved=FALSE, bFileName, bAutoSave=TRUE;
BOOL   bSelSideA = TRUE, bSelSideB = FALSE, bBoth = TRUE, bCancelSelDlg = FALSE;

/* trace line position on view A and B windows */
int traceA=1, traceB=1, avgRangeLo=1, avgRangeHi=1;

//xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  signed int _huge  * hpMemRaw;
signed int _huge  * hpMemRawA;
signed int _huge  * hpMemRawB;

static HGLOBAL   hgMemRaw, hgMemRawA, hgMemRawB;

/* windows handles */
HDC hdcMemA, hdcMemB;
HBITMAP hAFMLOGO, hOldBmpA, hBmpRawA, hBmpFilteredA, hBmpRecA, hBmpNowA;
HBITMAP hOldBmpB, hBmpRawB, hBmpFilteredB, hBmpRecB, hBmpNowB;

/*
 * Global variables used to contain data obtained from Paradox Table (scantbl.db)
 * via Borland Database Engine functions
 */
char     szDbKey[50];              // "Name" field
char     szSampleName[32];         // "Sample name field
char     szImageFile[32];          // Image filename
```

```
char    szComment[255];         // Comment field                                    70
INT16   nXPoints;               // Number of X points in image (256 or 512)
INT16   nYPoints;               // Number of Y points in image (256 or 512)
FLOAT   fZUnitsPerAngs;         // Z units per angstrom
FLOAT   fXArea, fYArea;

/* Variables to hold constants from paradox for side A of blade */
char     szDbKeyA[50];          // "Name" field
char     szSampleNameA[32];     // "Sample name field
char     szImageFileA[32];      // Image filename
char     szCommentA[255];       // Comment field                                    80
INT16   nXPointsA;              // Number of X points in image (256 or 512)
INT16   nYPointsA;              // Number of Y points in image (256 or 512)
FLOAT   fZUnitsPerAngsA;        // Z units per angstrom
FLOAT   fXAreaA, fYAreaA;       // dimension of image A

    /* Variables to hold constants from paradox for side B of blade */
char     szDbKeyB[50];          // "Name" field
char     szSampleNameB[32];     // "Sample name field
char     szImageFileB[32];      // Image filename
char     szCommentB[255];       // Comment field                                    90
INT16   nXPointsB;              // Number of X points in image (256 or 512)
INT16   nYPointsB;              // Number of Y points in image (256 or 512)
FLOAT   fZUnitsPerAngsB;        // Z units per angstrom
FLOAT   fXAreaB, fYAreaB;

/* A for left side image, B for right image */
MatrixF _huge *AA;
MatrixF _huge *BB;
MatrixF _huge *A=NULL;
MatrixF _huge *B=NULL;                                                           100

MatrixF _huge *revB=NULL;
MatrixF _huge *revBB=NULL;
/* reconstructed 2D profile */
MatrixF _huge *nPx=NULL,  _huge *nPAz=NULL,  _huge *nPBz=NULL;
MatrixF _huge *nPAz_bar, _huge *nPBz_bar;
/* 3D profile  */
MatrixF _huge *XX=NULL,  _huge *YY=NULL;
MatrixF _huge *ZZA=NULL, _huge *ZZB=NULL;
/* blade edge */                                                                 110
MatrixI _huge *edgeptA;
MatrixI _huge *edgeptB;

/* Norm data */
float _huge NormDist[] = {0.02, 0.05, 0.1, 0.5, 1, 2, 4, 8, 16};
MatrixF _huge *NormThicknessA, _huge *NormThicknessB, _huge *NormThicknessAB;
MatrixF _huge *NormAvgThicknessA, _huge *NormAvgThicknessB, _huge *NormAvgThicknessAB;
MatrixF _huge *stdA, _huge *stdB, _huge *stdAB;
MatrixF _huge *SharpA, _huge *SharpB, _huge *SharpAB;
float stdSharpA, stdSharpB, stdSharpAB, SharpA_bar, SharpB_bar, SharpAB_bar;    120

/* images size */
int      COLUMNA, COLUMNB, ROWA, ROWB;
```

```
/* default setting */
float thetaA = 8.0134, thetaB = 8.2696;
float offset1=0.5, offset2=0.5;
float threshold1=0.25,threshold2=0.25;
float slope1=-0.68;
static WORD    filterSelect=ID_WIENERFILTER, reconSelect=ID_RECONS1;
char    filename[64], filename3D[64];

/*
 *  WinMain -- Window procedure for processing messages to and from the
 *             parent window.  This section controls the initiation of all the
 *             application's functions.
 */
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine,
                            int nCmdShow)

{
  MSG msg;              /* a message structure */
  WNDCLASS wndclass;    /* window class structure */
  char cBuf[128];

  /* fill in class data for new class */
  if (!hPrevInstance) {
    wndclass.style      = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance   = hInstance;
    wndclass.hIcon       = LoadIcon (hInstance, "MyIcon");
    wndclass.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wndclass.hbrBackground     = GetStockObject (LTGRAY_BRUSH);
    wndclass.lpszMenuName      = "MyMenu";
    wndclass.lpszClassName     = "MyClass";
    /* register new class */
    if (!RegisterClass (&wndclass))
      return (0);       /* quit if can't register class */

    /* Change the class definition for the 'A' child window */
    wndclass.lpfnWndProc       = ChildProcA;
    wndclass.hIcon             = NULL;
    wndclass.hCursor           = LoadCursor(NULL, IDC_CROSS);
    wndclass.hbrBackground     = GetStockObject(WHITE_BRUSH);
    wndclass.lpszClassName     = "AChildClass";
    /* register the 'A' child window class */
    if (!RegisterClass (&wndclass)){
      MessageBox (hWnd, "Could not register the 'A' child class", "Message", MB_OK);
      return (0);    /* quit if can't register */
    }

    /* Change the class definition for the 'B' child window */
    wndclass.lpfnWndProc       = ChildProcB;
    wndclass.lpszClassName     = "BChildClass";
    /* register the 'B' child window class */
```

```
  if (!RegisterClass(&wndclass)){
    MessageBox(hWnd, "Could not register the 'B' child class", "Message", MB_OK);
    return(0);
  }

  /* Change the class definition for the 'ZoomA' child window */
  wndclass.lpfnWndProc       = ChildZoomA;
  wndclass.lpszClassName     = "ZoomAClass";
  /* register the 'ZoomA' child window class */
  if (!RegisterClass(&wndclass)){
    MessageBox(hWnd, "Could not register the 'ZoomA' child class", "Message", MB_OK);
    return(0);
  }

  /* Change the class definition for the 'ZoomB' child window */
  wndclass.lpfnWndProc       = ChildZoomB;
  wndclass.lpszClassName     = "ZoomBClass";
  /* register the 'ZoomB' child window class */
  if (!RegisterClass(&wndclass)){
    MessageBox(hWnd, "Could not register the 'ZoomA' child class", "Message", MB_OK);
    return(0);
  }

  /* Change the class definition for the 'Main' child window */
  wndclass.lpfnWndProc       = ChildProcMain;
  wndclass.hCursor           = LoadCursor(NULL, IDC_ARROW);
  wndclass.hbrBackground     = GetStockObject(BLACK_BRUSH);
  wndclass.lpszClassName     = "MainChildClass";
  /* register the 'Main' child window class */
  if (!RegisterClass(&wndclass)){
    MessageBox(hWnd, "Could not register the 'Main' child class", "Message", MB_OK);
    return(0);
  }

  /* Change the class definition for the 'ChildInfo' child window */
  wndclass.lpfnWndProc         = ChildInfoProc;
  wndclass.hbrBackground     = GetStockObject(WHITE_BRUSH);
  wndclass.lpszMenuName      = "InfoMenu";
  wndclass.lpszClassName     = "ChildInfoClass";
  /* register the 'Info' child window class */
  if (!RegisterClass(&wndclass)){
    MessageBox(hWnd, "Could not register the 'Info' child class", "Message", MB_OK);
    return(0);
  }
}

hAFMLOGO = LoadBitmap(hInstance,"AFMLOGO");
LoadString (hInstance, S_PROGRAMCAPTION, cBuf, sizeof (cBuf));
/* create the program window */
hWnd = CreateWindow("MyClass", cBuf ,
                    WS_OVERLAPPEDWINDOW | WS_MAXIMIZE,// | WS_CLIPCHILDREN,
                    0, 0, 640, 500, NULL, NULL, hInstance, NULL);

hStaticRectMain = CreateWindow ("MainChildClass", "",
```

```
                                WS_CHILD | WS_BORDER | WS_VISIBLE ,
                                0, 0, 0, 0, hWnd, NULL, hInstance, NULL);
  hStaticRectA = CreateWindow ("AChildClass", "",
                               WS_CHILD | WS_BORDER |WS_VISIBLE ,
                               0, 0, 0, 0, hWnd, NULL, hInstance, NULL);
  hZoomA = CreateWindow ("ZoomAClass", "",
                         WS_CHILD | WS_BORDER ,//|WS_VISIBLE,
                         0, 0, 0, 0, hWnd, NULL, hInstance, NULL);
  hZoomB = CreateWindow ("ZoomBClass", "",                                    240
                         WS_CHILD | WS_BORDER ,
                         0, 0, 0, 0,   hWnd, NULL, hInstance, NULL);
  hStaticRectB = CreateWindow ("BChildClass", "",
                               WS_CHILD | WS_BORDER | WS_VISIBLE ,
                               0, 0, 0, 0, hWnd, NULL, hInstance, NULL);

  ShowWindow (hWnd, nCmdShow);        /* display the window */
  UpdateWindow(hWnd);

  while (GetMessage (&msg, NULL, NULL, NULL)) { /* message loop */            250
    TranslateMessage (&msg);     /* translate keyboard messages */
    DispatchMessage (&msg);    /* send message to WndProc() */
  }
  DeleteObject(hAFMLOGO);
  return(msg.wParam);                                  /* quit */
}


/* WndProc() is a custom function for processing messages from Windows */
long FAR PASCAL WndProc (HWND hWnd, WORD wMessage, WORD wParam, LONG lParam)   260
{
  static       int cxwin, cywin;
  HDC   hdc;
  HPEN    hPen, hOldPen;
  HBITMAP      hOldBmp;
  static HMENU hMenu;

  // Logical Palette  variables
    int nBW;

  PAINTSTRUCT   ps;                                                           270

  FILEOPENDATA FAR * lpFOD;

  static       HWND    hButton[10], hScrl;
  static       HWND    hDataPtGroup, hAttrbGroup;
  static       HWND    hDdatapoint[6], hTdatapoint[6], hSdatapoint[6];
  static       HWND    hAttrb[7];
  static       HANDLE  hInstance, hInst;
  static       HDC     hLogoDC;
  static       BOOL    bNotCancel = TRUE;                                     280
  static       int     hRawFileIn, hRawFileBIn, scrlPos =0, maxPos=0, minPos=0;
  static       int     NormNumber = 6;
  /*********************************************************/
  static       FILE    *tmpfile, *outfile;
```

```
static       DataSavedNo=0;
int   ch;
char  cBuf[256];    //cBuf2[128]
MatrixF tmpMatrixFA, tmpMatrixFB, tmpMatrixFAA, tmpMatrixFBB, tmpMatrixFAB;
                                                                          290
static float maxA, maxB, minA, minB;
float _huge *ptsA, _huge *ptsB, _huge *ptsC;
float _huge *ptsAA, _huge *ptsBB;
float _huge *ptsXX, _huge *ptsYY, _huge *ptsZZA, _huge *ptsZZB;
float tmpA, tmpB, tmp, tmpAB;
int itmp;

static float XUNIT, YUNIT, NORMGRID = 0.02;
register int i,j;
                                                                          300
OFSTRUCT      of;           /* OpenFile () structure */

static       FARPROC lpfnAboutDlgProc;
static       FARPROC lpfnSelectA_DlgProc;
static       FARPROC lpfnSaveAs;
static       FARPROC lpfnSetupDlg;
static       FARPROC lpfnSave3D;
static       WORD LTRSelect=TOTAL_BUTTON;

switch (wMessage){          /* process windows messages */                310
case WM_CREATE:    /* program is just starting */
  if ( (tmpfile=fopen("afmparam.def", "r")) == NULL )
    MessageBox(NULL, "Cannot open afmparam.def, use default values", "Warning", MB_OK);
  else { /* read in the setting values */
    fscanf(tmpfile, "%d", &itmp);
    if (itmp==1) filterSelect=ID_MEDIANFILTER;
    else if (itmp=2) filterSelect=ID_WIENERFILTER;
    fscanf(tmpfile, "%d", &itmp);
    if (itmp==1) reconSelect=ID_RECONS1;
    else if (itmp==2) reconSelect=ID_RECONS2;                             320

    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) thetaA=tmp;
    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) thetaB=tmp;

    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) threshold1=tmp;
    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) offset1=tmp;                                               330
    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) slope1=tmp;
    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) threshold2=tmp;
    itmp=fscanf(tmpfile, "%f", &tmp);
    if (~itmp) offset2=tmp;
    fclose(tmpfile);
  }
```

```
// Create the logical palette for the application                          340
lPal = (LPLOGPALETTE) farmalloc(  sizeof(LOGPALETTE)+
                                  sizeof(PALETTEENTRY)*64);
lPal -> palVersion = 0x300;
lPal -> palNumEntries = 64;
for(i=0; i<64; i++){
  if(i==63)
    nBW = i*4;
  else
    nBW = 4*i;
  lPal->palPalEntry [i].peRed =                                           350
    lPal->palPalEntry [i].peGreen =
      lPal->palPalEntry [i].peBlue = nBW;
  lPal->palPalEntry [i].peFlags = 0;
}
/* procedure setup */
lpfnSelectA_DlgProc = MakeProcInstance((FARPROC) SelectA_DlgProc, hInstance);
hInstance = ((LPCREATESTRUCT) lParam)->hInstance;
lpfnAboutDlgProc = MakeProcInstance((FARPROC) AboutDlgProc, hInstance);
lpfnSaveAs = MakeProcInstance(SaveAsDlg, hInstance);
lpfnSave3D = MakeProcInstance(Save3DDlg, hInstance);                      360
lpfnSetupDlg = MakeProcInstance(SetupDlg, hInstance);

/* display the controls */
hScrl = CreateWindow("scrollbar", NULL,
                     WS_CHILD|WS_VISIBLE|SBS_HORZ,
                     0, 0, 0, 0, hWnd, SCROLLBAR, hInstance, NULL);
SetScrollRange(hScrl, SB_CTL, minPos, maxPos, FALSE);
SetScrollPos(hScrl, SB_CTL, scrlPos, FALSE);

hDataPtGroup = CreateWindow ("BUTTON", "",                                370
                             WS_CHILD | BS_GROUPBOX | WS_VISIBLE,
                             0, 0, 0, 0, hWnd, NULL, hInstance, NULL);
hAttrbGroup = CreateWindow ("BUTTON", "",
                            WS_CHILD | BS_GROUPBOX | WS_VISIBLE,
                            0, 0, 0, 0, hWnd, NULL, hInstance, NULL);
hButton[0] = CreateWindow ("BUTTON", "Locate &Edge",
                           WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE,
                           0, 0, 0, 0, hWnd, LOCATEDGE_BUTTON, hInstance, NULL);
hButton[1] = CreateWindow ("BUTTON", "Calculate &Profile",
                           WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE,         380
                           0, 0, 0, 0, hWnd, CALCPROFILE_BUTTON, hInstance, NULL);
hButton[2] = CreateWindow ("BUTTON", "&Average All",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, AVGALL_BUTTON, hInstance, NULL);
hButton[3] = CreateWindow ("BUTTON", "Average &Rgn",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, AVGRGN_BUTTON, hInstance, NULL);
hButton[4] = CreateWindow ("BUTTON", "Clear Al&l",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, CLEARALL_BUTTON, hInstance, NULL);390
hButton[5] = CreateWindow ("BUTTON", "Left",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, LEFT_BUTTON, hInstance, NULL);
```

```
hButton[6] = CreateWindow ("BUTTON", "Total",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, TOTAL_BUTTON, hInstance, NULL);
hButton[7] = CreateWindow ("BUTTON", "Right",
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, RIGHT_BUTTON, hInstance, NULL);
hButton[8] = CreateWindow ("BUTTON", "Save",                                    400
                           WS_CHILD | BS_PUSHBUTTON |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, SAVE_BUTTON, hInstance, NULL);
hButton[9] = CreateWindow ("BUTTON", "Auto",
                           WS_CHILD | BS_AUTOCHECKBOX |WS_VISIBLE,
                           0, 0, 0, 0, hWnd, AUTOSAVE_BUTTON, hInstance, NULL);

for (i=0; i<7; i++) {
  hAttrb[i] = CreateWindow ("static","",
                            WS_CHILD | SS_CENTER | WS_VISIBLE | WS_BORDER,
                            0, 0, 0, 0,    hWnd, ATTRIBUTE0+i, hInstance, NULL);
}

for (i=0; i<6; i++){
  hDdatapoint[i] = CreateWindow ("Static","",
                                 WS_CHILD | SS_RIGHT | WS_VISIBLE | WS_BORDER,
                                 0, 0, 0, 0, hWnd, DDATAPOINT1+i, hInstance, NULL);
  sprintf(cBuf, "%4.3f", NormDist[i]);
  SetWindowText(hDdatapoint[i],  cBuf);
  hTdatapoint[i] = CreateWindow ("Static","",
                                 WS_CHILD | SS_RIGHT | WS_VISIBLE | WS_BORDER,420
                                 0, 0, 0, 0, hWnd, TDATAPOINT1+i, hInstance, NULL);
  hSdatapoint[i] = CreateWindow ("Static","",
                                 WS_CHILD | SS_RIGHT | WS_VISIBLE | WS_BORDER,
                                 0, 0, 0, 0, hWnd, SDATAPOINT1+i, hInstance, NULL);
}

/* Create the Information Child Window */
hInformation = CreateWindow("ChildInfoClass", "Information" , WS_OVERLAPPED,
                            120, 60, 370, 395, hWnd, NULL, hInstance, NULL);
                                                                                430
/* Set Menu Items to their state (i.e.  Enabled or Disabled) upon start up*/
hMenu = GetMenu(hWnd);
EnableWindow(hButton[1], FALSE);
EnableWindow(hButton[0], FALSE);
EnableWindow(hButton[2], FALSE);
EnableWindow(hButton[3], FALSE);
EnableWindow(hButton[4], FALSE);
EnableWindow(hButton[6], FALSE);
EnableWindow(hButton[8], FALSE);
SendMessage(hButton[9], BM_SETCHECK, 1, 0L);                                    440

EnableMenuItem(hMenu, IDM_FILECLOSE, MF_GRAYED);
EnableMenuItem(hMenu, IDM_SAVE, MF_GRAYED);
EnableMenuItem(hMenu, IDM_SAVEAS, MF_GRAYED);
EnableMenuItem(hMenu, IDM_SAVE3D, MF_GRAYED);

EnableMenuItem(hMenu, 1, MF_GRAYED|MF_BYPOSITION);
```

```
  EnableMenuItem(hMenu, 2, MF_GRAYED|MF_BYPOSITION);
  EnableMenuItem(hMenu, 3, MF_GRAYED|MF_BYPOSITION);
  EnableMenuItem(hMenu, 5, MF_GRAYED|MF_BYPOSITION);                    450
  DrawMenuBar(hWnd);
  break;
case WM_COMMAND:
  switch (wParam){
  case IDM_OPENPROJECT:
    bCancelSelDlg = FALSE;
    // Dialog proceedure to select working image file for side
     // 'A' of the current sample.
       DialogBox(hInstance, "SelectA_DlgBox", hWnd, lpfnSelectA_DlgProc);
    if(bCancelSelDlg){                                                  460
     MessageBox(hWnd,"No file Selected.", "Message" , MB_ICONINFORMATION | MB_OK);
     return 0;
    }
    if(!bBoth){
     if((lstrcmp(szImageFileA, "")) && (lstrcmp(szImageFileB, ""))){
       MessageBox(hWnd, "Error in selecting files. Please try again.",
                  "Message" , MB_ICONHAND | MB_OK);
       break;
     }
     else if (lstrcmp(szImageFileA, "")){                               470
       SendMessage(hWnd, WM_COMMAND, IDM_OPENIMAGEA, 0L);
       break;
     }
     else if (lstrcmp(szImageFileB, "")){
       SendMessage(hWnd, WM_COMMAND, IDM_OPENIMAGEB, 0L);
       break;
     }
    }
    else {
     SendMessage(hWnd, WM_COMMAND, IDM_OPENIMAGEA, 0L);                 480
     SendMessage(hWnd, WM_COMMAND, IDM_OPENIMAGEB, 0L);
    }
    bSaved=FALSE;
    return 0;
  case IDM_OPENIMAGEA:
    /*getcwd (cBuf, 128);  /* get current directory */
    lstrcpy (cBuf, "C:\\IMGFILES");
    lpFOD = (FILEOPENDATA FAR *)GlobalAllocPtr(GHND, sizeof(FILEOPENDATA));
    memset(lpFOD->szPath, '\0', sizeof(lpFOD->szPath));
    memset(lpFOD->szFile, '\0', sizeof(lpFOD->szFile));                 490
    lstrcpy (lpFOD->szPath, cBuf);  /* initialize */
    lstrcpy (lpFOD->szFile, szImageFileA);
    lstrcpy (lpFOD->szCaption, "Open Image File A");
    /* call dlg box in dll */
    if (bOpenA)
     SendMessage(hWnd, WM_COMMAND, IDM_FILECLOSE, 1L);
    UpdateWindow(hWnd);
    if (bNotCancel) { /* if a file was selected */
     /* combine path and file */
     memset(cBuf, '\0', sizeof(cBuf));                                  500
     CombinePathFile (cBuf, lpFOD->szPath, lpFOD->szFile);
```

```
hRawFileIn = (OpenFile( cBuf, &of, OF_PARSE));
if (hRawFileIn != -1){
  SetCursor(LoadCursor (NULL, IDC_WAIT));
  // Enter BDE Information code here
    lstrcpy(szSampleName, szSampleNameA);
  lstrcpy(szImageFile, szImageFileA);
  TableOpen(hWnd);
  lstrcpy(szDbKeyA, szDbKey);
  lstrcpy(szCommentA, szComment);                                    510
  nXPointsA = nXPoints;
  nYPointsA = nYPoints;
  fXAreaA = fXArea;
  fYAreaA = fYArea;
  ROWA = nYPointsA;
  COLUMNA = nXPointsA;
  XUNIT = fXAreaA/(COLUMNA-1);
  YUNIT = fYAreaA/(ROWA-1);
  NORMGRID = ceil(XUNIT*100)/100;
  avgRangeLo = 1;                                                    520
  avgRangeHi = ROWA;
  fZUnitsPerAngsA = fZUnitsPerAngs;
  //  end of BDE Information
    /* create matrix for processing*/
    A = new_MatrixF(ROWA, COLUMNA);
  AA = new_MatrixF(ROWA, COLUMNA);
  edgeptA = new_MatrixI(ROWA, 1);

  hRawFileIn = (OpenFile( cBuf, &of, OF_READ));
  if(hRawFileIn != -1){                                              530
    /* load data */
    bOpenA = TRUE;

    hgMemRawA = LoadRawData(hRawFileIn, hgMemRawA);
    LoadMatrixF(ROWA, COLUMNA, A, hgMemRawA, &maxA, &minA);
    GlobalFree(hgMemRawA);

    hdc = GetDC(hStaticRectA);
    // hdcMemA is not deleted until file A is closed
      hdcMemA = CreateCompatibleDC(hdc);                             540
    hBmpRawA = CreateCompatibleBitmap(hdc, ROWA, COLUMNA);
    hOldBmpA = SelectObject(hdcMemA, hBmpRawA);

    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemA, hPal, FALSE);
    RealizePalette(hdcMemA);
    DisplayFile(hdcMemA, A, maxA, minA, ROWA, COLUMNA);

    SelectPalette(hdcMemA, hOldPal, FALSE);
    DeleteObject(hPal);                                              550
    ReleaseDC(hStaticRectA, hdc);
    /* display on view A window */
    SendMessage(hWnd, WM_COMMAND, IDM_VIEWRAWA, 0L);
    /* set up the appropriate menu states */
    EnableMenuItem(hMenu, 1, MF_ENABLED|MF_BYPOSITION);
```

```
        EnableMenuItem(hMenu, 2, MF_ENABLED|MF_BYPOSITION);
        EnableMenuItem(hMenu, 3, MF_ENABLED|MF_BYPOSITION);
        EnableMenuItem(hMenu, 5, MF_ENABLED|MF_BYPOSITION);
        EnableMenuItem(hMenu, IDM_FILTER, MF_ENABLED);
        EnableMenuItem(hMenu, IDM_CLEARFRAMEA, MF_ENABLED);                    560
        EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);

        EnableMenuItem(hMenu, IDM_VIEWRAWA, MF_ENABLED);
        if (bOpenB) {
          EnableWindow(hButton[0], TRUE);
          EnableWindow(hButton[1], TRUE);
          EnableWindow(hButton[4], TRUE);
          EnableMenuItem(hMenu, IDM_2DPROFILE, MF_ENABLED);
        }
        DrawMenuBar(hWnd);                                                     570
      }
      else
        MessageBox( hWnd, "Unable to read file. Please try again.", "Message" ,
                    MB_ICONHAND | MB_OK);
      SetCursor(LoadCursor (NULL, IDC_WAIT));
    }
    else
      MessageBox( hWnd, "Unable to open selected file. Please try again.",
                  "Message" , MB_ICONHAND | MB_OK);
  }                                                                            580
  else
    MessageBox(hWnd, "No file was selected", "Message", MB_ICONINFORMATION | MB_OK);
  GlobalFreePtr(lpFOD);
  return 0;
case IDM_OPENIMAGEB:
  /*getcwd (cBuf, 128);  /* get current directory */
  lstrcpy (cBuf, "C:\\IMGFILES");
  lpFOD = (FILEOPENDATA FAR *)GlobalAllocPtr(GHND, sizeof(FILEOPENDATA));
  memset(lpFOD->szPath, '\0', sizeof(lpFOD->szPath));
  memset(lpFOD->szFile, '\0', sizeof(lpFOD->szFile));                          590
  lstrcpy (lpFOD->szPath, cBuf);  /* initialize */
  lstrcpy (lpFOD->szFile, szImageFileB);
  lstrcpy (lpFOD->szCaption, "Open Image File B");
  /* call dlg box in dll */
  if(bOpenB) SendMessage(hWnd, WM_COMMAND, IDM_FILECLOSE, 2L);
  UpdateWindow(hWnd);
  if (bNotCancel){ /* if a file was selected */
    /* combine path and file */
    memset(cBuf, '\0', sizeof(cBuf));
    CombinePathFile (cBuf, lpFOD->szPath, lpFOD->szFile);                      600
    hRawFileBIn = (OpenFile( cBuf, &of, OF_PARSE));
    if (hRawFileBIn != -1){
      hRawFileBIn = (OpenFile( cBuf, &of, OF_READ));
      if(hRawFileBIn != -1){
        SetCursor(LoadCursor(NULL, IDC_WAIT));
        // Enter BDE Information code here
          lstrcpy(szSampleName, szSampleNameB);
        lstrcpy(szImageFile, szImageFileB);
        TableOpen(hWnd);
```

```
lstrcpy(szDbKeyB, szDbKey);
lstrcpy(szCommentB, szComment);
nXPointsB = nXPoints;
nYPointsB = nYPoints;
fXAreaB = fXArea;
fYAreaB = fYArea;
ROWB = nYPointsB;
COLUMNB = nXPointsB;
XUNIT = fXAreaB/(COLUMNB-1);
YUNIT = fYAreaB/(ROWB-1);
NORMGRID = ceil(XUNIT*100)/100;
fZUnitsPerAngsB = fZUnitsPerAngs;
//  end of BDE Information
avgRangeLo = 1;
avgRangeHi = ROWB;
/* B is reversed to have the same orientation as A */
B = new_MatrixF(ROWB, COLUMNB);
revB = new_MatrixF(ROWB, COLUMNB);

BB = new_MatrixF(ROWB, COLUMNB);
revBB = new_MatrixF(ROWB, COLUMNB);
edgeptB = new_MatrixI(ROWB, 1);

bOpenB = TRUE;
bFilteredB = FALSE;

hgMemRawB = LoadRawData(hRawFileBIn, hgMemRawB);
LoadMatrixF(ROWB, COLUMNB, B, hgMemRawB, &maxB, &minB);
GlobalFree(hgMemRaw);

hdc = GetDC(hStaticRectB);
hdcMemB = CreateCompatibleDC(hdc);
hBmpRawB = CreateCompatibleBitmap(hdc, COLUMNB, ROWB);
hOldBmpB = SelectObject(hdcMemB, hBmpRawB);

hPal = CreatePalette(lPal);
hOldPal = SelectPalette(hdcMemB, hPal, FALSE);
RealizePalette(hdcMemB);
DisplayFile(hdcMemB, B, maxB, minB, ROWB, COLUMNB);
SelectPalette(hdcMemB, hOldPal, FALSE);
DeleteObject(hPal);
ReleaseDC(hStaticRectB, hdc);

SendMessage(hWnd, WM_COMMAND, IDM_VIEWRAWB, 0L);

EnableMenuItem(hMenu, 1, MF_ENABLED|MF_BYPOSITION);
EnableMenuItem(hMenu, 2, MF_ENABLED|MF_BYPOSITION);
EnableMenuItem(hMenu, 3, MF_ENABLED|MF_BYPOSITION);
EnableMenuItem(hMenu, 5, MF_ENABLED|MF_BYPOSITION);
EnableMenuItem(hMenu, IDM_CLEARFRAMEB, MF_ENABLED);
EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);

EnableMenuItem(hMenu, IDM_VIEWRAWB, MF_ENABLED);
if (bOpenA) {
```

```
        EnableWindow(hButton[0], TRUE);
        EnableWindow(hButton[1], TRUE);
        EnableWindow(hButton[4], TRUE);
        EnableMenuItem(hMenu, IDM_2DPROFILE, MF_ENABLED);
      }
      DrawMenuBar(hWnd);
    }                                                                    670
    else
      MessageBox( hWnd, "Unable to read file. Please try again.", "Message" ,
                  MB_ICONHAND | MB_OK);
  }
  else
    MessageBox( hWnd, "Unable to open selected file. Please try again.", "Message" ,
                MB_ICONHAND | MB_OK);
  SetCursor(LoadCursor (NULL, IDC_ARROW));
 }
 else                                                                    680
  MessageBox(hWnd, "No file was selected", "Message", MB_ICONINFORMATION | MB_OK);
 GlobalFreePtr(lpFOD);
 return 0;
 /* save the norm data */
case IDM_SAVE:
 for (i=1; i<=NormThicknessAB->column; i++)
  fprintf(tmpfile, "%f \t", *(NormAvgThicknessAB->buffer+i));
 fprintf(tmpfile, "%f \t", 0);
 for (i=1; i<=NormThicknessB->column; i++)
  fprintf(tmpfile, "%f \t", *(NormAvgThicknessB->buffer+i));             690
 fprintf(tmpfile, "%f \t", 0);
 for (i=1; i<=NormThicknessA->column; i++)
  fprintf(tmpfile, "%f \t", *(NormAvgThicknessA->buffer+i));
 fprintf(tmpfile, "%f \t", 0);

 for (i=1; i<=stdAB->column; i++)
  fprintf(tmpfile, "%f \t", *(stdAB->buffer+i));
 fprintf(tmpfile, "%f \t", 0);
 for (i=1; i<=stdB->column; i++)
  fprintf(tmpfile, "%f \t", *(stdB->buffer+i));                          700
 fprintf(tmpfile, "%f \t", 0);
 for (i=1; i<=stdA->column; i++)
  fprintf(tmpfile, "%f \t", *(stdA->buffer+i));
 fprintf(tmpfile, "%f \t", 0);
 fprintf(tmpfile, "%f \t %f \t %f \t", SharpAB_bar, SharpB_bar, SharpA_bar);
 fprintf(tmpfile, "%f \t %f \t %f \t", stdSharpAB, stdSharpB, stdSharpA);
 fprintf(tmpfile, "\n");
 fclose(tmpfile);
 if ( (tmpfile=fopen("afm.dat", "a")) == NULL) {
  MessageBox(NULL, "Cannot append afm.dat", "Warning", MB_OK);           710
  return 0;
 }
 bSaved = TRUE;
 ++DataSavedNo;
 sprintf(cBuf, "%d", DataSavedNo);
 SetWindowText(hAttrb[6], cBuf);
```

```
  EnableWindow(hButton[8], FALSE);
  EnableMenuItem(hMenu, IDM_SAVE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_SAVEAS, MF_ENABLED);                    720
  return 0;
  /* save the norm data to another file */
case IDM_SAVEAS:
  if ( DialogBox(hInstance, "SaveAsBox", hWnd, lpfnSaveAs) ) {
    if ( (outfile=fopen(filename, "w")) != NULL ) {
      fclose(tmpfile);
      bDataFile=FALSE;
      if ( (tmpfile=fopen("afm.dat", "r")) == NULL ) {
        MessageBox(NULL, "Cannot open afm.dat", "Warning", MB_OK);
        return 0;                                                   730
      }
      else {
        SetCursor(LoadCursor (NULL, IDC_WAIT));
        while ((ch=getc(tmpfile))!=EOF) putc(ch, outfile);
        fclose(tmpfile);
        fclose(outfile);
        bDataFile=FALSE;
        DataSavedNo=0;
        sprintf(cBuf, "%d", DataSavedNo);
        SetWindowText(hAttrb[6], cBuf);                             740
        EnableMenuItem(hMenu, IDM_SAVEAS, MF_GRAYED);
        SetCursor(LoadCursor (NULL, IDC_ARROW));
      }
    }
    else {
      MessageBox(NULL, "Invalid, Cannot Open File", "Warning", MB_OK);
    }
  }
  return 0;
  /* save 3D data to a file */                                      750
case IDM_SAVE3D:
  if ( DialogBox(hInstance, "Save3DDlg", hWnd, lpfnSave3D) ) {
    if ( (outfile=fopen(filename3D, "w")) != NULL ) {
      SetCursor( LoadCursor (NULL, IDC_WAIT));
      ptsXX=XX->buffer;
      ptsYY=YY->buffer;
      ptsZZA=ZZA->buffer;
      ptsZZB=ZZB->buffer;

      for (i=1;i<=XX->row;i++)                                      760
        for (j=1; j<=XX->column; j++){
          ++ptsXX; ++ptsYY; ++ptsZZA; ++ptsZZB;
          fprintf(outfile, "%f %f %f %f \n", *ptsXX, *ptsYY, *ptsZZA, *ptsZZB);
        }
      fclose(outfile);
      SetCursor(LoadCursor (NULL, IDC_ARROW));
    }
    else
      MessageBox(NULL, "Invalid, Cannot Open File", "Warning", MB_OK);
  }                                                                 770
  return 0;
```

```
// IDM_FILECLOSE calls IDM_CLEARxxx first to clear the image from
  // from the screen, then clean the data from memory.
  case IDM_FILECLOSE:
    // clear data A from memory if lParam = 1
      // clear data B from memory if lParam = 2;
// clear both from memory if lParam = 0;

if (bOpenA&&(lParam==0||lParam==1)){                                    780
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARFRAMEA, 1L);
  bOpenA = FALSE;
  bFilteredA = FALSE;
  bReconstructedA = FALSE;
  SelectObject(hdcMemA, hOldBmpA);
  DeleteObject(hBmpRawA);
  DeleteObject(hBmpFilteredA);
  DeleteObject(hBmpRecA);
  DeleteDC(hdcMemA);
  _lclose(hRawFileIn);                                                  790

  nXPointsA = 0;
  nYPointsA = 0;
  fZUnitsPerAngsA = 0;
  // free memory
  if(!old_MatrixF(A)) MessageBox (hWnd, "Could not Free A", "Test",
                                  MB_OK|MB_ICONEXCLAMATION);
  if(!old_MatrixF(AA))MessageBox (hWnd, "Could not Free AA", "Test",
                                  MB_OK|MB_ICONEXCLAMATION);
  if(!old_MatrixI(edgeptA))MessageBox (hWnd, "Could not Free dgeptA", "      800
                    Test", MB_OK|MB_ICONEXCLAMATION);
}

if (bOpenB&&(lParam==0||lParam==2)){
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARFRAMEB, 0L);
  bOpenB = FALSE;
  bFilteredB = FALSE;
  bReconstructedB = FALSE;
  SelectObject(hdcMemB, hOldBmpB);
  DeleteObject(hBmpRawB);                                               810
  DeleteObject(hBmpFilteredB);
  DeleteObject(hBmpRecB);
  DeleteDC(hdcMemB);
  _lclose(hRawFileBIn);

  nXPointsB = 0;
  nYPointsB = 0;
  fZUnitsPerAngsB = 0;
  // free memory
  if(!old_MatrixF(B))MessageBox (hWnd, "Could not Free B", "Test",      820
                                 MB_OK|MB_ICONEXCLAMATION);
  if(!old_MatrixF(revB))MessageBox (hWnd, "Could not Free revB", "Test",
                                    MB_OK|MB_ICONEXCLAMATION);
  if(!old_MatrixF(BB))MessageBox (hWnd, "Could not Free BB", "Test",
                                  MB_OK|MB_ICONEXCLAMATION);
```

```
  if(!old_MatrixF(revBB))MessageBox (hWnd, "Could not Free revBB", "Test",
                                     MB_OK|MB_ICONEXCLAMATION);
  if(!old_MatrixI(edgeptB))MessageBox (hWnd, "Could not Free edgeptB", "Test",
                                       MB_OK|MB_ICONEXCLAMATION);
}

if (bCombine2D){
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARMAINFRAME, 0L);
  bCombine2D = FALSE;
  NormNumber=6;
  scrlPos = NormNumber-6;
  maxPos = 0; minPos=0;
  SetScrollRange(hScrl, SB_CTL, minPos, maxPos, FALSE);
  SetScrollPos(hScrl, SB_CTL, scrlPos, TRUE);
  sprintf(cBuf, "%s", "");
  for (i=1; i<=6; i++){
    SetWindowText(hSdatapoint[i-1], cBuf);
    SetWindowText(hTdatapoint[i-1], cBuf);
  }
  for (i=1; i<=6; i++)
    SetWindowText(hAttrb[i-1], cBuf);
  for (i=1;i<=6; i++){
    sprintf(cBuf, "%4.3f", NormDist[i-1]);
    SetWindowText(hDdatapoint[i-1], cBuf);
  }

  old_MatrixF(nPx);
  old_MatrixF(nPAz);
  old_MatrixF(nPBz);
  old_MatrixF(NormThicknessA);
  old_MatrixF(NormThicknessB);
  old_MatrixF(NormThicknessAB);
  old_MatrixF(nPAz_bar);
  old_MatrixF(nPBz_bar);
  old_MatrixF(NormAvgThicknessA);
  old_MatrixF(NormAvgThicknessB);
  old_MatrixF(NormAvgThicknessAB);
  old_MatrixF(stdA);
  old_MatrixF(stdB);
  old_MatrixF(stdAB);
  old_MatrixF(SharpA);
  old_MatrixF(SharpB);
  old_MatrixF(SharpAB);
  b2DAvgProfile = FALSE;

  if (bCombine3D){
    old_MatrixF(XX);
    old_MatrixF(YY);
    old_MatrixF(ZZA);
    old_MatrixF(ZZB);
    bCombine3D=FALSE;
  }
}
```

```
  for (i=0; i<5; i++)                                                    880
    EnableWindow(hButton[i], FALSE);

  EnableMenuItem(hMenu, IDM_FILECLOSE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_SAVE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_SAVE3D, MF_GRAYED);

  EnableMenuItem(hMenu, IDM_RECONSTRUCT, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_2DPROFILE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_2DAVGPROFILE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_3DPROFILE, MF_GRAYED);                       890

  EnableMenuItem(hMenu, IDM_VIEWRAWA, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEWRAWB, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEWFILTEREDA, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEWFILTEREDB, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEWRECONSTRUCTA, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEWRECONSTRUCTB, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEW2DPROFILE, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_VIEW2DAVGPROFILE, MF_GRAYED);
                                                                         900
  if (DataSavedNo==0)
    EnableMenuItem(hMenu, IDM_SAVEAS, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_FILTER, MF_GRAYED);
  EnableMenuItem(hMenu, IDM_RECONSTRUCT, MF_GRAYED);
  EnableMenuItem(hMenu, 1, MF_GRAYED|MF_BYPOSITION);
  EnableMenuItem(hMenu, 2, MF_GRAYED|MF_BYPOSITION);
  EnableMenuItem(hMenu, 3, MF_GRAYED|MF_BYPOSITION);
  EnableMenuItem(hMenu, 5, MF_GRAYED|MF_BYPOSITION);
  DrawMenuBar(hWnd);
  return 0;                                                              910
  /* apply filter */
case IDM_FILTER:
  SetCursor( LoadCursor (NULL, IDC_WAIT));
  if(bOpenA && !bFilteredA){
    /* filter twice */
    if (filterSelect==ID_MEDIANFILTER){
      medfilt2(A, AA, 3, 1);
      medfilt2(AA, A, 1, 3);
    }
    else if (filterSelect==ID_WIENERFILTER) {                            920
      wiener2(A, AA, 5, 1);
      wiener2(AA, A, 1, 5);
    }

    bFilteredA = TRUE;
    /* display result */
    hdc = GetDC(hStaticRectA);
    hBmpFilteredA = CreateCompatibleBitmap(hdc, COLUMNA, ROWA);
    SelectObject(hdcMemA, hBmpFilteredA);
    hPal = CreatePalette(lPal);                                          930
    hOldPal = SelectPalette(hdcMemA, hPal, FALSE);
    RealizePalette(hdcMemA);
    DisplayFile(hdcMemA, A, maxA, minA, ROWA, COLUMNA);
```

```
    SelectPalette(hdcMemA, hOldPal, FALSE);
    DeleteObject(hPal);
    ReleaseDC(hStaticRectA, hdc);
    SendMessage(hWnd, WM_COMMAND, IDM_VIEWFILTEREDA, 0L);
    EnableMenuItem(hMenu, IDM_VIEWFILTEREDA, MF_ENABLED);
  }
  if(bOpenB && !bFilteredB){                                          940
    if (filterSelect==ID_MEDIANFILTER) {
      medfilt2(B, BB, 3, 1);
      medfilt2(BB, B, 1, 3);
    }
    else if (filterSelect==ID_WIENERFILTER) {
      wiener2(B, BB, 5, 1);
      wiener2(BB, B, 1, 5);
    }

    bFilteredB = TRUE;                                                950

    hdc = GetDC(hStaticRectB);
    hBmpFilteredB = CreateCompatibleBitmap(hdc, COLUMNB, ROWB);
    SelectObject(hdcMemB, hBmpFilteredB);
    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemB, hPal, FALSE);
    RealizePalette(hdcMemB);
    DisplayFile(hdcMemB, B, maxB, minB, ROWB, COLUMNB);
    SelectPalette(hdcMemB, hOldPal, FALSE);
    DeleteObject(hPal);                                               960
    ReleaseDC(hStaticRectB, hdc);
    SendMessage(hWnd, WM_COMMAND, IDM_VIEWFILTEREDB, 0L);
    EnableMenuItem(hMenu, IDM_VIEWFILTEREDB, MF_ENABLED);
  }

  //if(bFilteredA && bFilteredB ){
    EnableMenuItem(hMenu, IDM_FILTER, MF_GRAYED );
    EnableMenuItem(hMenu, IDM_RECONSTRUCT, MF_ENABLED);
    //}
  SetCursor(LoadCursor (NULL, IDC_ARROW));                            970
  return 0;
  /* reconstruct the image and find the blade edge */
case IDM_RECONSTRUCT:
  SetCursor( LoadCursor (NULL, IDC_WAIT));
  if (bOpenA && bFilteredA) {
    /* reconstruction and edge-finding */
    if (reconSelect==ID_RECONS1)
      rec_edg1(A, AA, edgeptA, XUNIT, threshold1, offset1,slope1);
    else if (reconSelect==ID_RECONS2)
      rec_edg2(A, AA, edgeptA, XUNIT, threshold2, offset2);           980

    bReconstructedA = TRUE;
    /* display result */
    hdc = GetDC(hStaticRectA);
    hBmpRecA = CreateCompatibleBitmap(hdc, COLUMNA, ROWA);
    SelectObject(hdcMemA, hBmpRecA);
    hBmpNowA=hBmpRecA;
```

```
    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemA, hPal, FALSE);
    RealizePalette(hdcMemA);                                              990
    DisplayFile(hdcMemA, AA, maxA, minA, ROWA, COLUMNA);
    SelectPalette(hdcMemA, hOldPal, FALSE);
    DeleteObject(hPal);
    ReleaseDC(hStaticRectA, hdc);
    SendMessage(hWnd, WM_COMMAND, IDM_VIEWRECONSTRUCTA, 0L);
  }

  if (bOpenB && bFilteredB) {
    /* reverse matrix before doing reconstruction */
    fliplr(B, revB);                                                      1000
    if (reconSelect==ID_RECONS1)
      rec_edg1(revB, revBB, edgeptB, XUNIT, threshold1, offset1, slope1);
    else if (reconSelect==ID_RECONS2)
      rec_edg2(revB, revBB, edgeptB, XUNIT, threshold2, offset2);
    /* recover the matrix and edge */
    fliplr(revBB, BB);
    for (i=1; i<=BB->row; i++)
      *(edgeptB->buffer+i) = BB->column - *(edgeptB->buffer+i) + 1;
    bReconstructedB = TRUE;
    hdc = GetDC(hStaticRectB);                                            1010
    hBmpRecB = CreateCompatibleBitmap(hdc, COLUMNB, ROWB);
    SelectObject(hdcMemB, hBmpRecB);
    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemB, hPal, FALSE);
    RealizePalette(hdcMemB);
    DisplayFile(hdcMemB, BB, maxB, minB, ROWB, COLUMNB);
    //for (i=1; i<=ROW; i++)
      //    SetPixel(hdcMemB, *(edgeptB->buffer+i)-1, ROW-i, RGB(0, 255, 0));
    SelectPalette(hdcMemB, hOldPal, FALSE);
    DeleteObject(hPal);                                                   1020
    ReleaseDC(hStaticRectB, hdc);
    SendMessage(hWnd, WM_COMMAND, IDM_VIEWRECONSTRUCTB, 0L);
  }

  SendMessage(hWnd, WM_COMMAND, IDM_VIEWRECONSTRUCTA, 0L);
  SendMessage(hWnd, WM_COMMAND, IDM_VIEWRECONSTRUCTB, 0L);

  EnableMenuItem(hMenu, IDM_RECONSTRUCT, MF_GRAYED);
  if (bReconstructedA&&bReconstructedB)
    EnableMenuItem(hMenu, IDM_2DPROFILE, MF_ENABLED|MF_BYCOMMAND);        1030

  EnableMenuItem(hMenu, IDM_VIEWRECONSTRUCTA, MF_ENABLED|MF_BYCOMMAND);
  EnableMenuItem(hMenu, IDM_VIEWRECONSTRUCTB, MF_ENABLED|MF_BYCOMMAND);
  EnableWindow(hButton[0], FALSE);
  SetCursor(LoadCursor (NULL, IDC_ARROW));
  return 0;
  /* calculate 2D profile */
case IDM_2DPROFILE:
  if (ROWA != ROWB) {
    MessageBox(NULL, "Image A and B have different sizes !", "Warning", MB_OK);  1040
    return 0;
```

```
}
if (!bFilteredA || !bFilteredB)
  SendMessage(hWnd, WM_COMMAND, IDM_FILTER, 0L);
if (!bReconstructedA || !bReconstructedB)
  SendMessage(hWnd, WM_COMMAND, IDM_RECONSTRUCT, 0L);

SetCursor(LoadCursor (NULL, IDC_WAIT));
/* combine the 2 sides images */
combine2D(AA, BB, edgeptA, edgeptB, thetaA, thetaB, XUNIT, NORMGRID, &nPx, &nPAz, &nPBz
bCombine2D = TRUE;

tmp = *(nPx->buffer+nPx->column);
for (i=1; i<=9; i++) {
  if (NormDist[i-1]>tmp){
    NormNumber = i-1;
    break;
  }
}
maxPos=NormNumber-6;                                                    1060
SetScrollRange(hScrl, SB_CTL, minPos, maxPos, TRUE);

NormThicknessA=new_MatrixF(ROWA, NormNumber);
NormThicknessB=new_MatrixF(ROWA, NormNumber);
NormThicknessAB=new_MatrixF(ROWA, NormNumber);
SharpA=new_MatrixF(ROWA, 1);
SharpB=new_MatrixF(ROWA, 1);
SharpAB=new_MatrixF(ROWA, 1);
for (i=1; i<=ROWA; i++) {
  ptsA=MatElm(NormThicknessA, i, 0);                                    1070
  ptsB=MatElm(NormThicknessB, i, 0);
  ptsC=MatElm(NormThicknessAB, i, 0);
  ptsAA=MatElm(nPAz, i, 0);
  ptsBB=MatElm(nPBz, i, 0);
  table(nPx->buffer, ptsAA, nPx->column, NormDist-1, ptsA, NormNumber);
  table(nPx->buffer, ptsBB, nPx->column, NormDist-1, ptsB, NormNumber);
  FVec_Sub_FVec(ptsB, ptsA, ptsC, NormNumber);
  *(SharpA->buffer+i)=atan((*(ptsA+1))*(-1)/NormDist[1])*180/PI;
  *(SharpB->buffer+i)=atan((*(ptsB+1)/NormDist[1]))*180/PI;
  *(SharpAB->buffer+i)=*(SharpA->buffer+i)+ *(SharpB->buffer+i);        1080
}
nPAz_bar = new_MatrixF(1, nPx->column);
nPBz_bar = new_MatrixF(1, nPx->column);

NormAvgThicknessA=new_MatrixF(1, NormNumber);
NormAvgThicknessB=new_MatrixF(1, NormNumber);
NormAvgThicknessAB=new_MatrixF(1, NormNumber);
stdA=new_MatrixF(1, NormNumber);
stdB=new_MatrixF(1, NormNumber);
stdAB=new_MatrixF(1, NormNumber);                                       1090

SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
SendMessage(hWnd, WM_COMMAND, IDM_VIEWRECONSTRUCTA, 0L);
EnableMenuItem(hMenu, IDM_2DPROFILE, MF_GRAYED);
EnableMenuItem(hMenu, IDM_3DPROFILE, MF_ENABLED);
```

```
  EnableMenuItem(hMenu, IDM_VIEW2DPROFILE, MF_ENABLED);
  EnableMenuItem(hMenu, IDM_2DAVGPROFILE, MF_ENABLED);
  EnableMenuItem(hMenu, IDM_VIEW2DAVGPROFILE, MF_GRAYED);
  SetCursor(LoadCursor (NULL, IDC_ARROW));
  EnableWindow(hButton[1], FALSE);                                        1100
  EnableWindow(hButton[2], TRUE);
  EnableWindow(hButton[3], TRUE);
  return 0;
  /* calculate 2D average profile */
case IDM_2DAVGPROFILE:
  if (!bDataFile) {
    if ((tmpfile=fopen("afm.dat", "w")) == NULL )
      MessageBox(NULL, "Cannot open tmp.dat", "Warning", MB_OK);
    else {
      EnableMenuItem(hMenu, IDM_SAVEAS, MF_ENABLED);                      1110
      bDataFile = TRUE;
    }
  }

  SetCursor(LoadCursor(NULL, IDC_WAIT));
  tmpMatrixFA.buffer = MatElm(nPAz, avgRangeLo, 1) - 1;
  tmpMatrixFA.row = avgRangeHi-avgRangeLo+1;
  tmpMatrixFA.column = nPAz_bar->column;
  average(&tmpMatrixFA, nPAz_bar);
                                                                          1120
  tmpMatrixFB.buffer = MatElm(nPBz, avgRangeLo, 1) - 1;
  tmpMatrixFB.row = avgRangeHi-avgRangeLo+1;
  tmpMatrixFB.column = nPBz_bar->column;
  average(&tmpMatrixFB, nPBz_bar);

  tmpMatrixFAA.buffer = MatElm(NormThicknessA, avgRangeLo, 1) - 1;
  tmpMatrixFAA.row = avgRangeHi-avgRangeLo+1;
  tmpMatrixFAA.column = NormAvgThicknessA->column;
  average(&tmpMatrixFAA, NormAvgThicknessA);
  std(&tmpMatrixFAA, stdA);                                               1130

  tmpMatrixFBB.buffer = MatElm(NormThicknessB, avgRangeLo, 1) - 1;
  tmpMatrixFBB.row = avgRangeHi-avgRangeLo+1;
  tmpMatrixFBB.column = NormAvgThicknessB->column;
  average(&tmpMatrixFBB, NormAvgThicknessB);
  std(&tmpMatrixFBB, stdB);

  tmpMatrixFAB.buffer = MatElm(NormThicknessAB, avgRangeLo, 1) - 1;
  tmpMatrixFAB.row = avgRangeHi-avgRangeLo+1;
  tmpMatrixFAB.column = NormAvgThicknessAB->column;                       1140
  average(&tmpMatrixFAB, NormAvgThicknessAB);
  std(&tmpMatrixFAB, stdAB);

  SharpA_bar=0; SharpB_bar=0; SharpAB_bar=0;tmpA=0; tmpB=0;
  for (i=avgRangeLo; i<=avgRangeHi; i++) {
    SharpA_bar += *(SharpA->buffer+i);
    SharpB_bar += *(SharpB->buffer+i);
    SharpAB_bar += *(SharpAB->buffer+i);
  }
```

```
  SharpA_bar /= (avgRangeHi-avgRangeLo+1);                                    1150
  SharpB_bar /= (avgRangeHi-avgRangeLo+1);
  SharpAB_bar /= (avgRangeHi-avgRangeLo+1);
  for (i=avgRangeLo; i<=avgRangeHi; i++) {
    tmpA += (*(SharpA->buffer+i)*(*(SharpA->buffer+i))
             -2*(*(SharpA->buffer+i))*SharpA_bar
             +SharpA_bar*SharpA_bar);
    tmpB += (*(SharpB->buffer+i)*(*(SharpB->buffer+i))
             -2*(*(SharpB->buffer+i))*SharpB_bar
             +SharpB_bar*SharpB_bar);
    tmpAB += (*(SharpAB->buffer+i)*(*(SharpAB->buffer+i))                     1160
              -2*(*(SharpAB->buffer+i))*SharpAB_bar
              +SharpAB_bar*SharpAB_bar);
  }
  stdSharpA = sqrt(tmpA)/(avgRangeHi-avgRangeLo);
  stdSharpB = sqrt(tmpB)/(avgRangeHi-avgRangeLo);
  stdSharpAB = sqrt(tmpAB)/(avgRangeHi-avgRangeLo);

  SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DAVGPROFILE, 0L);
  b2DAvgProfile=TRUE;
  EnableMenuItem(hMenu, IDM_2DAVGPROFILE, MF_GRAYED);                         1170
  EnableMenuItem(hMenu, IDM_VIEW2DAVGPROFILE, MF_ENABLED);

  if ((avgRangeLo==1)&&(avgRangeHi==ROWA)&&(!bSaved)){
    EnableWindow(hButton[8], TRUE);
    EnableMenuItem(hMenu, IDM_SAVE, MF_ENABLED);
    if ( bAutoSave)
      SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0L);
  }
  SetCursor(LoadCursor(NULL, IDC_ARROW));
  return 0;                                                                   1180
  /* calculate 3D profile data */
case IDM_3DPROFILE:
  /* allcate memory for 3D presentation */
  XX = new_MatrixF(ROWA, nPx->column);
  YY = new_MatrixF(ROWA, nPx->column);
  ZZA = new_MatrixF(ROWA, nPx->column);
  ZZB = new_MatrixF(ROWA, nPx->column);

  combine3D(AA, BB, nPx, nPAz, nPBz, edgeptA, edgeptB, thetaA, thetaB,
            XX, YY, ZZA, ZZB, XUNIT, YUNIT );                                 1190
  EnableMenuItem(hMenu, IDM_SAVE3D, MF_ENABLED);
  bCombine3D=TRUE;
  //       SendMessage(hWnd, WM_COMMAND, IDM_VIEW3DPROFILE, 0L);
  return 0;
  /* IDM_CLEARFRAMEA only clear frame A and do not remove the data
     from the memory */
case IDM_CLEARFRAMEA:
  bZoomA = FALSE;
  bViewFilteredA = FALSE;
  bViewReconstructedA = FALSE;                                                1200
  bViewRawA = FALSE;
  hBmpNowA = NULL;
  EnableMenuItem(hMenu, IDM_CLEARFRAMEA, MF_GRAYED);
```

```
  DrawMenuBar(hWnd);
  ShowWindow(hZoomA, SW_HIDE);
  ShowWindow(hStaticRectA, SW_SHOWNORMAL);
  InvalidateRect(hStaticRectA, NULL, TRUE);
  break;
case IDM_CLEARFRAMEB:
  bZoomB = FALSE;                                                    1210
  bViewFilteredB = FALSE;
  bViewReconstructedB = FALSE;
  bViewRawB = FALSE;
  hBmpNowB = NULL;
  EnableMenuItem(hMenu, IDM_CLEARFRAMEB, MF_GRAYED);
  DrawMenuBar(hWnd);
  ShowWindow(hZoomB, SW_HIDE);
  ShowWindow(hStaticRectB, SW_SHOWNORMAL);
  InvalidateRect(hStaticRectB, NULL, TRUE);
  return 0;                                                          1220
case IDM_CLEARMAINFRAME:
  bView2DAvgProfile = FALSE;
  bView2DProfile = FALSE;
  bView3DProfile = FALSE;
  EnableMenuItem(hMenu, IDM_CLEARMAINFRAME, MF_GRAYED);
  DrawMenuBar(hWnd);
  InvalidateRect(hStaticRectMain, NULL, TRUE);
  return 0;
case IDM_CLEARALL:
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARFRAMEA, 0L);                1230
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARFRAMEB, 0L);
  SendMessage(hWnd, WM_COMMAND, IDM_CLEARMAINFRAME, 0L);
  EnableMenuItem(hMenu, IDM_CLEARALL, MF_GRAYED);
  DrawMenuBar(hWnd);
  return 0;
  /* display the raw A data on window */
case IDM_VIEWRAWA:
  bViewRawA = TRUE;
  bViewFilteredA = FALSE;
  bViewReconstructedA = FALSE;                                       1240

  hBmpNowA = hBmpRawA;
  if (!bZoomA){
    InvalidateRect(hStaticRectA, NULL, TRUE);
    EnableMenuItem(hMenu, IDM_CLEARFRAMEA, MF_ENABLED);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);
    DrawMenuBar(hWnd);
  }
  if (bZoomA){
    InvalidateRect(hZoomA, NULL, FALSE);                             1250
  }
  return 0;
case IDM_VIEWRAWB:
  bViewRawB = TRUE;
  bViewFilteredB = FALSE;
  bViewReconstructedB = FALSE;
```

```
  hBmpNowB = hBmpRawB;
  if (!bZoomB) {
    InvalidateRect(hStaticRectB, NULL, TRUE);                          1260
    EnableMenuItem(hMenu, IDM_CLEARFRAMEB, MF_ENABLED|MF_BYCOMMAND);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED|MF_BYCOMMAND);
    DrawMenuBar(hWnd);
  }
  if (bZoomB) {
    InvalidateRect(hZoomB, NULL, FALSE);
  }
  return 0;
case IDM_VIEWFILTEREDA:
  bViewFilteredA = TRUE;                                               1270
  bViewRawA = FALSE;
  bViewReconstructedA = FALSE;

  hBmpNowA = hBmpFilteredA;
  if (!bZoomA){
    InvalidateRect(hStaticRectA, NULL, FALSE);
    EnableMenuItem(hMenu, IDM_CLEARFRAMEA, MF_ENABLED|MF_BYCOMMAND);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED|MF_BYCOMMAND);
    DrawMenuBar(hWnd);
  }                                                                    1280
  if (bZoomA){
    InvalidateRect(hZoomA, NULL, FALSE);
    EnableWindow(hStaticRectMain, FALSE);
  }
  return 0;
case IDM_VIEWFILTEREDB:
  bViewFilteredB = TRUE;
  bViewRawB = FALSE;
  bViewReconstructedB = FALSE;
                                                                       1290
  hBmpNowB = hBmpFilteredB;
  if (!bZoomB) {
    InvalidateRect(hStaticRectB, NULL, TRUE);
    EnableMenuItem(hMenu, IDM_CLEARFRAMEB, MF_ENABLED|MF_BYCOMMAND);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED|MF_BYCOMMAND);
    DrawMenuBar(hWnd);
  }
  if (bZoomB) InvalidateRect(hZoomB, NULL, FALSE);
  return 0;
case IDM_VIEWRECONSTRUCTA:                                             1300
  bViewReconstructedA = TRUE;
  bViewRawA = FALSE;
  bViewFilteredA = FALSE;

  hBmpNowA = hBmpRecA;
  if (!bZoomA){
    InvalidateRect(hStaticRectA, NULL, FALSE);
    EnableMenuItem(hMenu, IDM_CLEARFRAMEA, MF_ENABLED);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);
    DrawMenuBar(hWnd);                                                 1310
  }
```

```
  if (bZoomA){
    InvalidateRect(hZoomA, NULL, FALSE);
    //EnableWindow(hStaticRectMain, FALSE);
    //EnableWindow(hStaticRectA, FALSE);
  }
  return 0;
case IDM_VIEWRECONSTRUCTB:
  bViewReconstructedB = TRUE;
  bViewRawB = FALSE;                                                    1320
  bViewFilteredB = FALSE;

  hBmpNowB = hBmpRecB;
  if (!bZoomB){
    InvalidateRect(hStaticRectB, NULL, FALSE);
    EnableMenuItem(hMenu, IDM_CLEARFRAMEB, MF_ENABLED);
    EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);
    DrawMenuBar(hWnd);
  }
  if (bZoomB/* && bSelFrameB*/){                                        1330
    InvalidateRect(hZoomB, NULL, FALSE);
  }
  return 0;
case IDM_VIEW2DPROFILE:
  bViewMain = TRUE;
  bView2DProfile = TRUE;
  InvalidateRect(hStaticRectMain, NULL, TRUE);

  if (LTRSelect==LEFT_BUTTON) {
    ptsA = MatElm(NormThicknessA, traceA, 1)-1;                         1340
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(ptsA+i));
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);
    }
  }
  else if (LTRSelect==RIGHT_BUTTON) {
    ptsA = MatElm(NormThicknessB, traceA, 1)-1;
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(ptsA+i));
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);                   1350
    }
  }
  else if (LTRSelect==TOTAL_BUTTON) {
    ptsA = MatElm(NormThicknessAB, traceA, 1)-1;
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(ptsA+i));
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);
    }
  }
  sprintf(cBuf, "%4.3f", *(SharpA->buffer+traceA));                     1360
  SetWindowText(hAttrb[0], cBuf);
  sprintf(cBuf, "%4.3f", *(SharpB->buffer+traceA));
  SetWindowText(hAttrb[1], cBuf);
  sprintf(cBuf, "%4.3f", *(SharpAB->buffer+traceA));
  SetWindowText(hAttrb[2], cBuf);
```

```
  sprintf(cBuf, "%s", "");
  for (i=1; i<=6; i++)
    SetWindowText(hSdatapoint[i-1], cBuf);
  SetWindowText(hAttrb[3], cBuf);                                    1370
  SetWindowText(hAttrb[4], cBuf);
  SetWindowText(hAttrb[5], cBuf);

  EnableMenuItem(hMenu, IDM_CLEARMAINFRAME, MF_ENABLED);
  EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);
  bView2DAvgProfile = FALSE;
  bView3DProfile = FALSE;
  return 0;
case IDM_VIEW2DAVGPROFILE:
  bView2DAvgProfile = TRUE;                                          1380
  InvalidateRect(hStaticRectMain, NULL, TRUE);
  if (LTRSelect==LEFT_BUTTON) {
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(NormAvgThicknessA->buffer+i));
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);
      sprintf(cBuf, "%4.3f", *(stdA->buffer+i));
      SetWindowText(hSdatapoint[i-scrlPos-1], cBuf);
    }
  }
  else if (LTRSelect==RIGHT_BUTTON) {                                1390
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(NormAvgThicknessB->buffer+i));
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);
      sprintf(cBuf, "%4.3f", *(stdB->buffer+i));
      SetWindowText(hSdatapoint[i-scrlPos-1], cBuf);
    }
  }
  else if (LTRSelect==TOTAL_BUTTON) {
    for (i=scrlPos+1; i<=scrlPos+6; i++) {
      sprintf(cBuf, "%4.3f", *(NormAvgThicknessAB->buffer+i));       1400
      SetWindowText(hTdatapoint[i-scrlPos-1],  cBuf);
      sprintf(cBuf, "%4.3f", *(stdAB->buffer+i));
      SetWindowText(hSdatapoint[i-scrlPos-1], cBuf);
    }
  }
  sprintf(cBuf, "%4.3f", SharpA_bar);
  SetWindowText(hAttrb[0], cBuf);
  sprintf(cBuf, "%4.3f", stdSharpA);
  SetWindowText(hAttrb[3], cBuf);
                                                                     1410
  sprintf(cBuf, "%4.3f", SharpB_bar);
  SetWindowText(hAttrb[1], cBuf);
  sprintf(cBuf, "%4.3f", stdSharpB);
  SetWindowText(hAttrb[4], cBuf);

  sprintf(cBuf, "%4.3f", SharpAB_bar);
  SetWindowText(hAttrb[2], cBuf);
  sprintf(cBuf, "%4.3f", stdSharpAB);
  SetWindowText(hAttrb[5], cBuf);
```

```
                                                                    1420
  EnableMenuItem(hMenu, IDM_CLEARMAINFRAME, MF_ENABLED);
  EnableMenuItem(hMenu, IDM_CLEARALL, MF_ENABLED);
  DrawMenuBar(hWnd);
  bViewMain = TRUE;
  return 0;
  /* popup setup window */
case IDM_SETUP:
  DialogBox(hInstance, "SetupDlg", hWnd, lpfnSetupDlg);
  return 0;
  /* popup information window */                                    1430
case IDM_INFORMATIONA:
  lstrcpy(szDbKey, szDbKeyA);
  lstrcpy(szImageFile, szImageFileA);
  lstrcpy(szSampleName, szSampleNameA);
  lstrcpy(szComment, szCommentA);
  nXPoints = nXPointsA;
  nYPoints = nYPointsA;
  fXArea = fXAreaA;
  fYArea = fYAreaA;
  fZUnitsPerAngs = fZUnitsPerAngsA;                                 1440
  ShowWindow(hInformation, SW_SHOWNORMAL);
  return 0;
case IDM_INFORMATIONB:
  lstrcpy(szDbKey, szDbKeyB);
  lstrcpy(szImageFile, szImageFileB);
  lstrcpy(szSampleName, szSampleNameB);
  lstrcpy(szComment, szCommentB);
  nXPoints = nXPointsB;
  nYPoints = nYPointsB;
  fXArea = fXAreaB;                                                 1450
  fYArea = fYAreaB;
  fZUnitsPerAngs = fZUnitsPerAngsB;
  ShowWindow(hInformation, SW_SHOWNORMAL);
  return 0;
  /* popup about window */
case IDM_HELPABOUT:
  hInstance = GetWindowWord (hWnd, GWW_HINSTANCE);
  DialogBox(hInstance, "AboutBox", hWnd, lpfnAboutDlgProc);
  return 0;
  // IDM_QUIT is the same as WM_DESTROY                             1460
  case IDM_QUIT:
    DestroyWindow(hWnd);
  return 0;
case LOCATEDGE_BUTTON:
  if (!bFilteredA || !bFilteredB)
    SendMessage (hWnd, WM_COMMAND, IDM_FILTER, 0L);
  if (!bReconstructedA || !bReconstructedB)
    SendMessage (hWnd, WM_COMMAND, IDM_RECONSTRUCT, 0L);
  EnableWindow(hButton[1], TRUE);
  EnableWindow(hButton[0], FALSE);                                  1470
  break;
case CLEARALL_BUTTON:
  SendMessage (hWnd, WM_COMMAND, IDM_CLEARALL, 0L);
```

```
    return 0;
  case CALCPROFILE_BUTTON:
    SendMessage(hWnd, WM_COMMAND, IDM_2DPROFILE, 0L);
    EnableWindow(hButton[1], FALSE);
    return 0;
  case AVGALL_BUTTON:
    avgRangeLo=1;                                                    1480
    avgRangeHi=ROWA;
    SendMessage(hWnd, WM_COMMAND, IDM_2DAVGPROFILE, 0L);
    return 0;
  case AVGRGN_BUTTON:
    SendMessage(hWnd, WM_COMMAND, IDM_2DAVGPROFILE, 0L);
    return 0;
  case LEFT_BUTTON:
  case RIGHT_BUTTON:
  case TOTAL_BUTTON:
    EnableWindow(hButton[LTRSelect-LEFT_BUTTON+5], TRUE);            1490
    LTRSelect = wParam;
    EnableWindow(hButton[LTRSelect-LEFT_BUTTON+5], FALSE);
    if (bView2DAvgProfile) SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DAVGPROFILE, 0L);
    else if (bView2DProfile) SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
    return 0;
  case SAVE_BUTTON:
    SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0L);
    EnableWindow(hButton[8], FALSE);
    return 0;
  case AUTOSAVE_BUTTON:                                              1500
    if (SendMessage(hButton[9], BM_GETCHECK, 0, 0L)){
      bAutoSave = TRUE;
    }
    else {
      bAutoSave = FALSE;
    }
    return 0;
  }
  break;
  /* scroll bar control */                                           1510
case WM_HSCROLL:
  switch(wParam) {
  case SB_PAGEUP:
  case SB_LINEUP:
    scrlPos--;
    break;
  case SB_THUMBPOSITION:
    scrlPos=LOWORD(lParam);
    break;
  case SB_LINEDOWN:                                                  1520
  case SB_PAGEDOWN:
    scrlPos++;
    break;
  default:
    break;
  }
  if (scrlPos>maxPos) scrlPos=maxPos;
```

```
  else if (scrlPos<minPos) scrlPos=minPos;
  if (scrlPos!=GetScrollPos(hScrl, SB_CTL)) {
    SetScrollPos(hScrl, SB_CTL, scrlPos, TRUE);                          1530
    for (i=scrlPos+1;i<=scrlPos+6; i++){
      sprintf(cBuf, "%4.3f", NormDist[i-1]);
      SetWindowText(hDdatapoint[i-scrlPos-1], cBuf);
    }
    if (bView2DAvgProfile) SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DAVGPROFILE, 0L);
    else if (bView2DProfile) SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
  }
  return(0);
  /* repaint the windows */
case WM_PAINT:                                                           1540
  hInstance = GetWindowWord(hWnd, GWW_HINSTANCE);

  hdc = BeginPaint (hWnd, &ps);

  hLogoDC = CreateCompatibleDC(hdc);
  hOldBmp = SelectObject (hLogoDC, hAFMLOGO);
  BitBlt(hdc, cxwin*0.8, cywin*0.38, cxwin*0.156, cywin*0.2, hLogoDC, 0, 0, SRCCOPY);
  SelectObject(hLogoDC, hOldBmp);
  DeleteDC(hLogoDC);
                                                                         1550
  SetBkMode(hdc, TRANSPARENT);
  TextOut(hdc, cxwin*0.08, cywin*0.33, "Left", 4);
  TextOut(hdc, cxwin*0.86, cywin*0.33, "Right", 5);
  TextOut(hdc, cxwin*0.26, cywin*0.006, "Data Point Values:", 18);
  TextOut(hdc, cxwin*0.258, cywin*0.13, "D", 1);
  TextOut(hdc, cxwin*0.258, cywin*0.19, "T", 1);
  TextOut(hdc, cxwin*0.258, cywin*0.25, "S", 1);
  TextOut(hdc, cxwin*0.023, cywin*0.4, "Sample Attributes:", 18);
  TextOut(hdc, cxwin*0.03, cywin*0.5, "Sharp L", 7);
  TextOut(hdc, cxwin*0.03, cywin*0.56, "Sharp R", 7);                    1560
  TextOut(hdc, cxwin*0.03, cywin*0.62, "Sharp T", 7);
  TextOut(hdc, cxwin*0.03, cywin*0.68, "Std L", 5);
  TextOut(hdc, cxwin*0.03, cywin*0.74, "Std R", 5);
  TextOut(hdc, cxwin*0.03, cywin*0.80, "Std T", 5);
  TextOut(hdc, cxwin*0.03, cywin*0.86, "DataSaved", 9);
  SetBkMode(hdc, OPAQUE);

  hPen = CreatePen(PS_SOLID, 2, RGB(0,128,0));
  hOldPen=SelectObject(hdc, hPen);
  SelectObject(hdc, GetStockObject(NULL_BRUSH));                         1570
  SelectObject(hdc, hOldPen);
  DeleteObject(hPen);
  EndPaint (hWnd, &ps);
  break;
case WM_SIZE:
  cywin=HIWORD(lParam);
  cxwin=LOWORD(lParam);
  MoveWindow(hStaticRectMain, cxwin*0.25, cywin*0.37, cxwin*0.5, cywin*0.55, TRUE);
  MoveWindow(hStaticRectA, cxwin*0.015, cxwin*0.015, cxwin*0.22, cxwin*0.22, TRUE);
  MoveWindow(hStaticRectB, cxwin*0.765, cxwin*0.015, cxwin*0.22, cxwin*0.22, TRUE);
  MoveWindow(hZoomA, cxwin*0.015, cxwin*0.015, cxwin*0.44, cxwin*0.44, TRUE);
```

```
    MoveWindow(hZoomB, cxwin*0.545, cxwin*0.015, cxwin*0.44, cxwin*0.44, TRUE);
    MoveWindow(hDataPtGroup, cxwin*0.25, cywin*0.03, cxwin*0.5, cywin*0.33, TRUE);
    MoveWindow(hAttrbGroup, cxwin*0.015, cywin*0.36, cxwin*0.22, cywin*0.60, TRUE);
    MoveWindow(hScrl, cxwin*0.4, cywin*0.31, cxwin*0.2, cywin*0.048, TRUE);
    for (i=0; i<5; i++)
      MoveWindow(hButton[i], cxwin*0.78, cywin*(0.65+0.06*i), cxwin*0.18, cywin*0.06, TRUE);
    for (i=0; i<7; i++)
      MoveWindow(hAttrb[i], cxwin*0.145, cywin*(0.5+0.06*i), cxwin*0.07, cywin*0.048, TRUE);
    for (i=0; i<6; i++)                                                          1590
      MoveWindow(hDdatapoint[i], cxwin*(0.277+0.078*i),
                 cywin*0.13, cxwin*0.075, cywin*0.052, TRUE);
    for (i=0; i<6; i++)
      MoveWindow(hTdatapoint[i], cxwin*(0.277+0.078*i),
                 cywin*0.19, cxwin*0.075, cywin*0.052, TRUE);
    for (i=0; i<6; i++)
      MoveWindow(hSdatapoint[i], cxwin*(0.277+0.078*i),
                 cywin*0.25, cxwin*0.075, cywin*0.052, TRUE);
    for (i=0; i<3; i++)
      MoveWindow(hButton[i+5], cxwin*(0.35+0.1*i), cywin*0.07, cxwin*0.1, cywin*0.052, TRUE);
    MoveWindow(hButton[8], cxwin*0.4, cywin*0.93, cxwin*0.2, cywin*0.06, TRUE);
    MoveWindow(hButton[9], cxwin*0.6, cywin*0.93, cxwin*0.1, cywin*0.06, TRUE);
    return 0;

    // WM_DESTROY is the same as WM_COMMAND [ IDM_QUIT ]
    // WM_DESTROY calls IDM_FILECLOSE first to clean images and data
    // both from the screen and from the memory.  Then it cleans up
    // the other object like fonts, ...
  case WM_DESTROY:            /* stop application */
    if (bOpenA)                                                                  1610
      SendMessage(hWnd, WM_COMMAND, IDM_FILECLOSE, 1L);
    else if (bOpenB)
      SendMessage(hWnd, WM_COMMAND, IDM_FILECLOSE, 2L);

    DeleteObject(hStaticRectA);
    DeleteObject(hStaticRectB);
    DeleteObject(hStaticRectMain);

    farfree(lPal);
    PostQuitMessage(0); /* this will exit message loop */                        1620
    return(0);
  default:    /* default windows message processing */
    return DefWindowProc (hWnd, wMessage, wParam, lParam);
  }

  return (0L);
}

/*
 *  ChildProcA -- Window procedure for processing messages to and from          1630
 *                the 'A' child window
 */
BOOL bTraceA=FALSE;
long FAR PASCAL ChildProcA (HWND hwnd, WORD wMessage, WORD wParam, LONG lParam)
{
```

```
PAINTSTRUCT  ps;
HDC          hdc;
HPEN         hPen, hOldPen;
static       int     cxwin, cywin, traceY=0;
BITMAP bm;                                                          1640
static BOOL bDragging=FALSE, bBlock=FALSE;
static POINT ptbeg, ptend;
int i;
float xfactor, yfactor;

switch (wMessage) {/* Process windows messages */
case WM_RBUTTONDBLCLK:
  /* code to display larger frame goes here */
  if((bViewRawA || bViewFilteredA || bViewReconstructedA)){
    bZoomA = TRUE;                                                  1650
    ShowWindow(hwnd, SW_HIDE);
    ShowWindow(hZoomA, SW_SHOW);
    InvalidateRect(hZoomA, NULL, FALSE);
  }
  return 0;
case WM_MOUSEMOVE:
  if (bDragging) {
    if (bBlock) OutBlock(hwnd, ptbeg, ptend);
    ptend = MAKEPOINT(lParam);
    OutBlock(hwnd, ptbeg, ptend);                                   1660
    bBlock=TRUE;
  }
  else if (bTraceA) {
    hdc = GetDC(hwnd);
    hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));
    hOldPen = SelectObject(hdc, hPen);
    SetROP2(hdc, R2_XORPEN);
    MoveTo(hdc, 0, traceY);
    LineTo(hdc, cxwin, traceY);
    traceY=HIWORD(lParam);                                          1670

    traceA=(int)(cywin-traceY-1)*ROWA/cywin+1;
    if (traceA>ROWA) traceA=ROWA;
    if (traceA<1) traceA=1;
    traceB=traceA;

    MoveTo(hdc, 0, traceY);
    LineTo(hdc, cxwin, traceY);

    SelectObject(hdc, hOldPen);                                     1680
    DeleteObject(hPen);
    ReleaseDC(hwnd, hdc);

    if (wParam!=1) { //not from window B
                      SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
                    SendMessage(hStaticRectB, WM_MOUSEMOVE, 1, lParam);
                  }
  }
  return 0;
```

```
case WM_LBUTTONDOWN:                                                    1690
  if (wParam==(MK_SHIFT|MK_LBUTTON)){
    if (bBlock) InBlock(hwnd, ptbeg, ptend, cxwin);
    if (!bDragging) {
      ptbeg=MAKEPOINT(lParam);
      bDragging=TRUE;
      bBlock=FALSE;
      SetCapture(hwnd);
    }
  }
  else if (bViewMain) {                                                 1700
    bTraceA=!bTraceA;
    if (bTraceA){
      hdc = GetDC(hwnd);

      hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));
      hOldPen = SelectObject(hdc, hPen);
      SetROP2(hdc, R2_XORPEN);

      MoveTo(hdc, 0, traceY);
      LineTo(hdc, cxwin, traceY);                                       1710

      traceY=HIWORD(lParam);
      traceA=(int)(cywin-traceY-1)*ROWA/cywin+1;
      traceB=traceA;
      MoveTo(hdc, 0, traceY);
      LineTo(hdc, cxwin, traceY);

      SelectObject(hdc, hOldPen);

      DeleteObject(hPen);                                               1720
      ReleaseDC(hwnd, hdc);
      SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
      SendMessage(hStaticRectB, WM_MOUSEMOVE, 1, lParam);
    }
  }
  return 0;
case WM_LBUTTONUP:
  if(bDragging) {
    bDragging=FALSE;
    if (bBlock) OutBlock(hwnd, ptbeg, ptend);                           1730
    ptend=MAKEPOINT(lParam);
    InBlock(hwnd, ptbeg, ptend, cxwin);
    ReleaseCapture();
    if (ptbeg.y>ptend.y){
      avgRangeLo = ptbeg.y<cywin ?  (long)(cywin-ptbeg.y-1)*ROWA/cywin+1 :  1;
      avgRangeHi = ptend.y>0 ?  (long)(cywin-ptend.y-1)*ROWA/cywin+1 :  ROWA;
    }
    else {
      avgRangeHi = ptbeg.y>0 ?  (long)(cywin-ptbeg.y-1)*ROWA/cywin+1 :  ROWA;
      avgRangeLo = ptend.y<cywin ?  (long)(cywin-ptend.y-1)*ROWA/cywin+1 :  1;1740
    }
  }
  return 0;
```

```
  case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);

    if((bViewRawA || bViewFilteredA || bViewReconstructedA)&& !bZoomA ){
      hPal = CreatePalette(lPal);
      hOldPal = SelectPalette(hdcMemA, hPal, FALSE);
      RealizePalette(hdcMemA);                                          1750
      SelectObject(hdcMemA, hBmpNowA);

      //SelectPalette(hdcMemA, hOldPal, FALSE);  /* Couldn't restore hOldPal*/
      hOldPal2=SelectPalette(hdc, hPal, FALSE); /* must be included */
      GetObject(hBmpNowA, sizeof(BITMAP), (LPSTR)&bm);
      //SetStretchBltMode(hdc, STRETCH_ORSCANS);
      SetStretchBltMode(hdc, STRETCH_DELETESCANS);
      StretchBlt(hdc, 0, 0, cxwin, cywin, hdcMemA, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
      SelectPalette(hdcMemA, hOldPal, FALSE);
      SelectPalette(hdc, hOldPal2, FALSE);                              1760
      if (bViewReconstructedA){
        yfactor=(float)cywin/ROWA;
        xfactor=(float)cxwin/COLUMNA;
        hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
        hOldPen=SelectObject(hdc, hPen);
        MoveTo(hdc, *(edgeptA->buffer+1)*xfactor, ((ROWA-1)*yfactor));
        for (i=2; i<=ROWA; i++)
          LineTo(hdc, *(edgeptA->buffer+i)*xfactor,(ROWA-i)*yfactor);
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);                                             1770
      }
      if (bViewMain) {
        hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));
        hOldPen = SelectObject(hdc, hPen);
        SetROP2(hdc, R2_XORPEN);

        MoveTo(hdc, 0, traceY);
        LineTo(hdc, cxwin, traceY);
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);                                             1780
      }
      DeleteObject(hPal);
    }
    if (bBlock) InBlock(hwnd, ptbeg, ptend, cxwin);
    EndPaint(hwnd, &ps);
    return 0;
  case WM_SIZE:
    cywin=HIWORD(lParam);
    cxwin=LOWORD(lParam);
    traceY=cywin-(int)((traceA-1)*cywin/256)-1;                         1790
    return 0;
  default:        /* Default windows message processing */
    return DefWindowProc(hwnd, wMessage, wParam, lParam);
  }
}
/* draw a rectangular frame
 *    ptbeg :  upperleft point
```

```
 *    ptend :  lowerright point
 * the region outside hwnd window will be drawed, too.
 */                                                                    1800
void OutBlock(HWND hwnd, POINT ptbeg, POINT ptend)
{
  HDC   hdc;
  HPEN  hOldPen;
  hdc=CreateDC("DISPLAY", NULL, NULL, NULL);
  ClientToScreen(hwnd, &ptbeg);
  ClientToScreen(hwnd, &ptend);
  hOldPen=SelectObject(hdc, CreatePen(PS_DOT, 1, RGB(0, 0, 0)));
  SelectObject(hdc, GetStockObject(NULL_BRUSH));
  SetROP2(hdc, R2_XORPEN);                                             1810
  Rectangle(hdc, ptbeg.x, ptbeg.y, ptend.x, ptend.y);
  DeleteObject(SelectObject(hdc, hOldPen));
  DeleteDC(hdc);
}
/* draw a rectangular frame
 * the same as OutBlock, but only the region within hwnd window will be drawed.
 */
void InBlock(HWND hwnd, POINT ptbeg, POINT ptend, int cxwin)
{
  HDC   hdc;                                                           1820
  HPEN  hOldPen;

  hdc=GetDC(hwnd);
  hOldPen=SelectObject(hdc, CreatePen(PS_SOLID, 1, RGB(0, 255, 255)));
  SelectObject(hdc, GetStockObject(NULL_BRUSH));
  SetROP2(hdc, R2_XORPEN);
  Rectangle(hdc, 1, ptbeg.y, cxwin-2, ptend.y);
  DeleteObject(SelectObject(hdc, hOldPen));
  ReleaseDC(hwnd,hdc);
}                                                                      1830

/*
 *  ChildZoomA -- Window procedure for processing messages to and from
 *                the zoom A child window
 */
long FAR PASCAL ChildZoomA (HWND hZoomA, WORD wMessage, WORD wParam, LONG lParam)
{
  static       int     cxwin, cywin;
  PAINTSTRUCT  ps;
  HDC          hdc;                                                    1840
  BITMAP               bm;
  HPEN         hOldPen, hPen;
  int          i;
  float        xfactor, yfactor;

  switch (wMessage){ /* Process windows messages */
  case WM_RBUTTONDBLCLK:
    /* code to display larger frame goes here */
    bZoomA = FALSE;
    ShowWindow(hZoomA, SW_HIDE);                                       1850
    ShowWindow(hStaticRectA, SW_SHOW);
```

```
    InvalidateRect(hStaticRectA, NULL, FALSE);
    break;
  case WM_PAINT:
    hdc = BeginPaint(hZoomA, &ps);

    if(bZoomA){
      hPal = CreatePalette(lPal);
      hOldPal = SelectPalette(hdcMemA, hPal, FALSE);
      RealizePalette(hdcMemA);                                          1860
      SelectObject(hdcMemA, hBmpNowA);

      //SelectPalette(hdcMemA, hOldPal, FALSE);  /* Couldn't restore hOldPal*/
      hOldPal2=SelectPalette(hdc, hPal, FALSE); /* must be included */
      GetObject(hBmpNowA, sizeof(BITMAP), (LPSTR)&bm);
      SetStretchBltMode(hdc, STRETCH_DELETESCANS);
      StretchBlt(hdc, 0, 0, cxwin, cywin, hdcMemA, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
      SelectPalette(hdcMemA, hOldPal, FALSE);
      SelectPalette(hdc, hOldPal2, FALSE);
      DeleteObject(hPal);                                               1870
      if (bViewReconstructedA){
        yfactor=(float)cywin/ROWA;
        xfactor=(float)cxwin/COLUMNA;
        hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
        hOldPen=SelectObject(hdc, hPen);
        MoveTo(hdc, *(edgeptA->buffer+1)*xfactor, ((ROWA-1)*yfactor));
        for (i=2; i<=ROWA; i++)
          LineTo(hdc, *(edgeptA->buffer+i)*xfactor,(ROWA-i)*yfactor);
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);                                             1880
      }
    }
    EndPaint(hZoomA, &ps);
    break;
  case WM_SIZE:
    cywin=HIWORD(lParam);
    cxwin=LOWORD(lParam);
    return 0;
  default:        /* Default windows message processing */
    return DefWindowProc(hZoomA, wMessage, wParam, lParam);             1890
  }
  return(0L);
}

/*
 *  ChildProcB -- Window procedure for processing messages to and from
 *                the view B child window
 */
long FAR PASCAL ChildProcB (HWND hwnd, WORD wMessage, WORD wParam, LONG lParam)
{                                                                       1900
  PAINTSTRUCT     ps;
  HDC          hdc;
  HPEN         hOldPen,hPen;
  static       int     cxwin, cywin, traceY=0;
  BITMAP               bm;
```

```
static BOOL bDragging=FALSE, bBlock=FALSE;
static POINT ptbeg, ptend;
int          i;
float        xfactor, yfactor;
                                                                    1910
switch (wMessage){ /* Process windows messages */
case WM_RBUTTONDBLCLK:
  /* code to display larger frame goes here */
  if((bViewRawB || bViewFilteredB || bViewReconstructedB)){
    bZoomB = TRUE;
    ShowWindow(hwnd, SW_HIDE);
    ShowWindow(hZoomB, SW_SHOW);
    InvalidateRect(hZoomB, NULL, FALSE);
  }
  return 0;                                                         1920
case WM_MOUSEMOVE:
  if (bDragging) {
    if (bBlock) OutBlock(hwnd, ptbeg, ptend);
    ptend = MAKEPOINT(lParam);
    OutBlock(hwnd, ptbeg, ptend);
    bBlock=TRUE;
  }
  else if (bTraceA) {
    hdc = GetDC(hwnd);
    hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));                  1930
    hOldPen = SelectObject(hdc, hPen);
    SetROP2(hdc, R2_XORPEN);
    MoveTo(hdc, 0, traceY);
    LineTo(hdc, cxwin, traceY);
    traceY=HIWORD(lParam);

    traceA=(int)(cywin-traceY-1)*ROWB/cywin+1;
    if (traceA>ROWB) traceA=ROWB;
    if (traceA<1) traceA=1;
    traceB=traceA;                                                  1940

    MoveTo(hdc, 0, traceY);
    LineTo(hdc, cxwin, traceY);
    SelectObject(hdc, hOldPen);
    DeleteObject(hPen);
    ReleaseDC(hwnd, hdc);
    if (wParam!=1) { //not from window B
                      SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
                    SendMessage(hStaticRectA, WM_MOUSEMOVE, 1, lParam);
                 }                                                  1950
  }
  return 0;
case WM_LBUTTONDOWN:
  if (wParam==(MK_SHIFT|MK_LBUTTON)){
    if (bBlock) InBlock(hwnd, ptbeg, ptend, cxwin);
    if (!bDragging) {
      ptbeg=MAKEPOINT(lParam);
      bDragging=TRUE;
      bBlock=FALSE;
```

```
      SetCapture(hwnd);                                                    1960
    }
  }
  else if (bViewMain) {
    bTraceA=!bTraceA;
    if (bTraceA){
      hdc = GetDC(hwnd);

      hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));
      hOldPen = SelectObject(hdc, hPen);
      SetROP2(hdc, R2_XORPEN);                                             1970

      MoveTo(hdc, 0, traceY);
      LineTo(hdc, cxwin, traceY);

      traceY=HIWORD(lParam);
      traceA=(int)(cywin-traceY-1)*ROWB/cywin+1;
      traceB=traceA;
      MoveTo(hdc, 0, traceY);
      LineTo(hdc, cxwin, traceY);
                                                                           1980
      SelectObject(hdc, hOldPen);

      DeleteObject(hPen);
      ReleaseDC(hwnd, hdc);
      SendMessage(hWnd, WM_COMMAND, IDM_VIEW2DPROFILE, 0L);
      SendMessage(hStaticRectA, WM_MOUSEMOVE, 1, lParam);
    }
  }
  return 0;
case WM_LBUTTONUP:                                                         1990
  if(bDragging) {
    bDragging=FALSE;
    if (bBlock) OutBlock(hwnd, ptbeg, ptend);
    ptend=MAKEPOINT(lParam);
    InBlock(hwnd, ptbeg, ptend, cxwin);
    ReleaseCapture();
    if (ptbeg.y>ptend.y){
      avgRangeLo = ptbeg.y<cywin ?  (long)(cywin-ptbeg.y-1)*ROWB/cywin+1 :  1;
      avgRangeHi = ptend.y>0 ?  (long)(cywin-ptend.y-1)*ROWB/cywin+1 :  ROWB;
    }                                                                      2000
    else {
      avgRangeHi = ptbeg.y>0 ?  (long)(cywin-ptbeg.y-1)*ROWB/cywin+1 :  ROWB;
      avgRangeLo = ptend.y<cywin ?  (long)(cywin-ptend.y-1)*ROWB/cywin+1 :  1;
    }
  }
  return 0;
case WM_PAINT:
  hdc = BeginPaint(hwnd, &ps);

  if((bViewRawB || bViewFilteredB || bViewReconstructedB)&& !bZoomB){      2010
    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemB, hPal, FALSE);
    RealizePalette(hdcMemB);
```

```
      SelectObject(hdcMemB, hBmpNowB);

      //SelectPalette(hdcMemA, hOldPal, FALSE);  /* Couldn't restore hOldPal*/
      hOldPal2=SelectPalette(hdc, hPal, FALSE); /* must be included */
      GetObject(hBmpNowB, sizeof(BITMAP), (LPSTR)&bm);
      SetStretchBltMode(hdc, STRETCH_DELETESCANS);
      StretchBlt(hdc, 0, 0, cxwin, cywin, hdcMemB, 0, 0, bm.bmWidth, bm.bmHeight,SRCCOPY);
      SelectPalette(hdcMemB, hOldPal, FALSE);
      SelectPalette(hdc, hOldPal2, FALSE);
      if (bViewReconstructedB){
        yfactor=(float)cywin/ROWB;
        xfactor=(float)cxwin/COLUMNB;
        hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
        hOldPen=SelectObject(hdc, hPen);
        MoveTo(hdc, *(edgeptB->buffer+1)*xfactor, ((ROWB-1)*yfactor));
        for (i=2; i<=ROWB; i++)
          LineTo(hdc, *(edgeptB->buffer+i)*xfactor,(ROWB-i)*yfactor);
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);
      }

      if (bViewMain) {
        hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 255));
        hOldPen = SelectObject(hdc, hPen);
        SetROP2(hdc, R2_XORPEN);

        MoveTo(hdc, 0, traceY);
        LineTo(hdc, cxwin, traceY);
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);
      }

      DeleteObject(hPal);
    }
    if (bBlock) InBlock(hwnd, ptbeg, ptend, cxwin);
    EndPaint(hwnd, &ps);
    return 0;
  case WM_SIZE:
    cywin=HIWORD(lParam);
    cxwin=LOWORD(lParam);
    traceY=cywin-(int)((traceA-1)*cywin/256)-1;
    return 0;
  default:        /* Default windows message processing */
    return DefWindowProc(hwnd, wMessage, wParam, lParam);
  }
}

/*
 *  ChildZoomB -- Window procedure for processing messages to and from
 *                the Zoom B child window
 */
long FAR PASCAL ChildZoomB (HWND hZoomB, WORD wMessage, WORD wParam, LONG lParam)
{
  PAINTSTRUCT   ps;
```

```
HDC          hdc;
HPEN         hOldPen,hPen;
static       int     cxwin, cywin;                                          2070
BITMAP               bm;
int          i;
float        xfactor, yfactor;

switch (wMessage){ /* Process windows messages */
case WM_RBUTTONDBLCLK:
  /* code to display original smaller frame goes here */
  bZoomB = FALSE;
  ShowWindow(hZoomB, SW_HIDE);
  ShowWindow(hStaticRectB, SW_SHOW);                                        2080
  InvalidateRect(hStaticRectB, NULL, FALSE);
  return 0;
case WM_PAINT:
  hdc = BeginPaint(hZoomB, &ps);

  if(bZoomB){
    hPal = CreatePalette(lPal);
    hOldPal = SelectPalette(hdcMemB, hPal, FALSE);
    RealizePalette(hdcMemB);
    SelectObject(hdcMemB, hBmpNowB);                                        2090

    //SelectPalette(hdcMemA, hOldPal, FALSE);  /* Couldn't restore hOldPal*/
    hOldPal2=SelectPalette(hdc, hPal, FALSE); /* must be included */
    GetObject(hBmpNowB, sizeof(BITMAP), (LPSTR)&bm);
    SetStretchBltMode(hdc, STRETCH_DELETESCANS);
    StretchBlt(hdc, 0, 0, cxwin, cywin, hdcMemB, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
    SelectPalette(hdcMemB, hOldPal, FALSE);
    SelectPalette(hdc, hOldPal2, FALSE);
    DeleteObject(hPal);
    if (bViewReconstructedB){                                               2100
      yfactor=(float)cywin/ROWB;
      xfactor=(float)cxwin/COLUMNB;
      hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
      hOldPen=SelectObject(hdc, hPen);
      MoveTo(hdc, *(edgeptB->buffer+1)*xfactor, ((ROWB-1)*yfactor));
      for (i=2; i<=ROWB; i++)
        LineTo(hdc, *(edgeptB->buffer+i)*xfactor,(ROWB-i)*yfactor);
      SelectObject(hdc, hOldPen);
      DeleteObject(hPen);
    }                                                                       2110

  }
  EndPaint(hZoomB, &ps);
  return 0;
case WM_SIZE:
  cywin=HIWORD(lParam);
  cxwin=LOWORD(lParam);
  return 0;
default:        /* Default windows message processing */
  return DefWindowProc(hZoomB, wMessage, wParam, lParam);                   2120
}
```

```
}

/*
 *  ChildProcMain -- Window procedure for processing messages to and from
 *                   the 'view 2D' child window
 */
long FAR PASCAL ChildProcMain (HWND hStaticRectMain, WORD wMessage, WORD wParam, LONG lParam)
{
  PAINTSTRUCT  ps;                                                          2130
  HDC          hdc;
  HPEN         hPen, hOldPen;
  int i;
  static       int     cxwin, cywin;
  char         cBuf[256];
  HFONT        hOldFont, hFont;
  float _huge *ptsA, _huge *ptsB;

  switch (wMessage) {/* Process windows messages */
  case WM_PAINT:                                                            2140
    hdc=BeginPaint(hStaticRectMain, &ps);
    if (bView2DProfile || bView2DAvgProfile ||bView3DProfile ){
      bViewMain=TRUE;
      SetBkMode(hdc, TRANSPARENT);
      SetMapMode(hdc, MM_ANISOTROPIC);
      SetWindowExt(hdc, 4000, 5000);
      SetViewportExt(hdc, cxwin, -cywin); //x :  increasing to the right
       SetWindowOrg(hdc, -400, 2250);      // y:  increasing to the up

         hPen = GetStockObject(WHITE_PEN);                                  2150
      hOldPen=SelectObject(hdc, hPen);
      MoveTo(hdc, 0, -2000);
      LineTo(hdc, 3500, -2000);
      LineTo(hdc, 3500, 2000);
      LineTo(hdc, 0, 2000);
      LineTo(hdc, 0, -2000);
      SelectObject(hdc, hOldPen);
      DeleteObject(hPen);

      hPen = CreatePen(PS_DOT,1,RGB(255, 255, 255));                        2160
      hOldPen=SelectObject(hdc, hPen);
      MoveTo(hdc, 1000, -2000);
      LineTo(hdc, 1000, 2000);
      MoveTo(hdc, 2000, -2000);
      LineTo(hdc, 2000, 2000);
      MoveTo(hdc, 3000, -2000);
      LineTo(hdc, 3000, 2000);
      MoveTo(hdc, 4000, -2000);
      LineTo(hdc, 4000, 2000);
      MoveTo(hdc, 0, -1000);                                                2170
      LineTo(hdc, 3500, -1000);
      MoveTo(hdc, 0, 0);
      LineTo(hdc, 3500, 0);
      MoveTo(hdc, 0, 1000);
      LineTo(hdc, 3500, 1000);
```

```
SelectObject(hdc, hOldPen);
DeleteObject(hPen);

/* Create Y-Axis-Title-Font */
SetTextColor(hdc, RGB(255, 255, 255));                                    2180
SetTextAlign(hdc, TA_CENTER|TA_TOP);

hFont = CreateFont (0, 0, -900, 0, FW_BOLD, 0, 0, 0, ANSI_CHARSET,
                    OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                    VARIABLE_PITCH |FF_SWISS, "");
SetTextColor(hdc, RGB(255, 255, 255));
SetTextAlign(hdc, TA_CENTER|TA_TOP);
TextOut(hdc, 1800, -2300, cBuf, wsprintf(cBuf, "X (micron)"));
TextOut(hdc, 0, -2000, cBuf, wsprintf(cBuf, "0"));
TextOut(hdc, 1000, -2000, cBuf, wsprintf(cBuf, "1"));                     2190
TextOut(hdc, 2000, -2000, cBuf, wsprintf(cBuf, "2"));
TextOut(hdc, 3000, -2000, cBuf, wsprintf(cBuf, "3"));
hOldFont=SelectObject(hdc, hFont);
TextOut(hdc, -400, 0, cBuf, wsprintf(cBuf, "Z (micron)"));
TextOut(hdc, -200, 0, cBuf, wsprintf(cBuf, "0"));
TextOut(hdc, -200, 1000, cBuf, wsprintf(cBuf, "1"));
TextOut(hdc, -200, -1000, cBuf, wsprintf(cBuf, "-1"));
SelectObject(hdc, hOldFont);
DeleteObject(hFont);
if (bView2DAvgProfile) {                                                  2200
  hPen=CreatePen(PS_SOLID,1, RGB(255, 0, 0));
  hOldPen=SelectObject(hdc, hPen);
  MoveTo(hdc, 0, 0);
  for (i=2; i<=nPx->column;i++)
    LineTo(hdc, *(nPx->buffer+i)*1000, *(nPAz_bar->buffer+i)*1000 );

  SelectObject(hdc, hOldPen);
  DeleteObject(hPen);
  hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
  hOldPen=SelectObject(hdc, hPen);                                        2210
  MoveTo(hdc, 0, 0);
  for (i=2; i<=nPx->column;i++)
    LineTo(hdc, *(nPx->buffer+i)*1000, *(nPBz_bar->buffer+i)*1000 );
}
else if (bView2DProfile){

  hPen=CreatePen(PS_SOLID,1, RGB(255, 0, 0));
  hOldPen=SelectObject(hdc, hPen);
  MoveTo(hdc, 0, 0);
  ptsA = MatElm(nPAz, traceA, 0); ptsB=MatElm(nPBz, traceB, 0);           2220
  for (i=2; i<=nPx->column;i++)
    LineTo(hdc, *(nPx->buffer+i)*1000, *(ptsA+i)*1000 );

  SelectObject(hdc, hOldPen);
  DeleteObject(hPen);
  hPen=CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
  hOldPen=SelectObject(hdc, hPen);
  MoveTo(hdc, 0, 0);
  for (i=2; i<=nPx->column;i++)
```

```
        LineTo(hdc, *(nPx->buffer+i)*1000, *(ptsB+i)*1000 );                    2230
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);
      }
    }
    else
      bViewMain = FALSE;
    EndPaint(hStaticRectMain, &ps);
    break;
  case WM_SIZE:
    hdc=GetDC(hStaticRectMain);                                                 2240
    cywin=HIWORD(lParam);
    cxwin=LOWORD(lParam);
    SetMapMode(hdc, MM_ISOTROPIC);
    SetWindowExt(hdc, cxwin, cywin);
    SetViewportExt(hdc, cxwin, -cywin);
    SetViewportOrg(hdc, 0, cywin/2);
    ReleaseDC(hStaticRectMain, hdc);
    break;
  default:        /* Default windows message processing */
    return DefWindowProc(hStaticRectMain, wMessage, wParam, lParam);            2250
  }
  return(0L);
}

/*
 *  Information Child Window -- Window procedure for processing messages to
 *                              and from the information window
 */
long FAR PASCAL ChildInfoProc (HWND hInformation, WORD wMessage, WORD wParam, LONG lParam)
{                                                                               2260
  PAINTSTRUCT          ps;
  HDC                  hInstance, hDC;

  static HANDLE        hInfoCancelPB, hInfoOKPB;
  static HANDLE        hCurSampleEF, hSampleNameEF, hDateEF;
  static HANDLE        hCommentsMEF, hXAreaEF, hYAreaEF, hImageSizeEF;
  char  cTempBuf[255]; // edit buffer
    char * pTempBuf;    // pointer to edit buffer

      switch (wMessage) {/* Process windows messages */                          2270
      case WM_CREATE:
       /* get the instance handle */
       hInstance = GetWindowWord (hInformation, GWW_HINSTANCE);
       hCurSampleEF = CreateWindow ("EDIT",NULL,
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
            20, 20, 150, 23, hInformation, CURSAMPLE_EF, hInstance, NULL);
       hSampleNameEF = CreateWindow ("EDIT",NULL,
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
             180, 20, 150, 23, hInformation, SESSION_EF, hInstance, NULL);
       hDateEF = CreateWindow ("EDIT",NULL,                                     2280
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
               20, 70, 150, 23, hInformation, DATE_EF, hInstance, NULL);
       hXAreaEF = CreateWindow ("EDIT",NULL,
```

```
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
                20, 120, 150, 23, hInformation, TIPSHARPNESS_EF, hInstance, NULL);
        hYAreaEF = CreateWindow ("EDIT",NULL,
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
                180, 120, 150, 23, hInformation, TIPRADIUS_EF, hInstance, NULL);
        hImageSizeEF = CreateWindow ("EDIT",NULL,
            WS_CHILD | WS_DISABLED | ES_CENTER | WS_VISIBLE | ES_AUTOHSCROLL | WS_BORDER,
                180, 70, 150, 23, hInformation, IMAGESIZE_EF, hInstance, NULL);
        hCommentsMEF = CreateWindow ("EDIT",NULL,
    WS_CHILD | ES_LEFT | WS_VSCROLL | WS_VISIBLE | ES_AUTOVSCROLL | ES_MULTILINE | WS_BORDER,
                20, 170, 330, 150, hInformation, COMMENTS_MEF, hInstance, NULL);
        hInfoOKPB = CreateWindow ("BUTTON", "&OK",
            WS_CHILD | BS_PUSHBUTTON | WS_VISIBLE,
                290, 330, 60, 30, hInformation, INFO_OK_BUTTON, hInstance, NULL);
        /*InvalidateRect(hStaticRectMain, NULL, TRUE);*/
        break;
      case WM_COMMAND:                                                          2300
        switch(wParam){
        case INFO_OK_BUTTON:
          ShowWindow(hInformation, SW_HIDE);
          break;
        }
        break;
      case WM_PAINT:
        hDC = BeginPaint (hInformation, &ps);
        TextOut(hDC, 20, 4, "Current Sample:", 15);
        TextOut(hDC, 180, 4, "SampleName :", 12);                              2310
        TextOut(hDC, 20, 54, "Date:", 5);
        TextOut(hDC, 20, 104, "XArea:", 6);
        TextOut(hDC, 180, 104, "YArea:", 6);
        TextOut(hDC, 180, 54, "Image Size:", 11);
        TextOut(hDC, 20, 154, "Comments:", 9);
        SetWindowText(hCurSampleEF, szImageFile);
        memset(cTempBuf, '\0', 255);
        strncpy(cTempBuf, szDbKey, 12);
        SetWindowText(hDateEF, cTempBuf);
        memset(cTempBuf, '\0', 255);                                            2320
        SetWindowText(hSampleNameEF, szSampleName);
        sprintf(cTempBuf, "%5.2f", fXArea);
        SetWindowText(hXAreaEF, cTempBuf);
        sprintf(cTempBuf, "%5.2f", fYArea);
        SetWindowText(hYAreaEF, cTempBuf);
        memset(cTempBuf, '\0', 255);
        wsprintf(cTempBuf, "%i X %i", nXPoints, nYPoints);
        SetWindowText(hImageSizeEF, cTempBuf);
        SetWindowText(hCommentsMEF, szComment);
        EndPaint (hInformation, &ps);                                           2330
        ReleaseDC (hInformation, hDC);
        break;
      default:        /* Default windows message processing */
        return DefWindowProc(hInformation, wMessage, wParam, lParam);
      }
  return(0L);
}
```

```
/*
 *  LoadRawData -- Window procedure for loading the raw data file                2340
 */
HGLOBAL LoadRawData (int hFileIn, HGLOBAL hgMemRaw)
{
  int   m, nRecIn = 1;
  register int i;
  int   long    nFileLong, n;
  signed int _huge *hpMemRaw;
  long max=32767;

  nFileLong = (int long)_llseek(hFileIn, 0L, 2);                               2350
  _llseek(hFileIn, 0L, 0);

  m = floor(nFileLong/max);
  n = nFileLong - max*m;
  if(hgMemRaw = GlobalAlloc (GHND, nFileLong+4L)) {
    if(hpMemRaw = (signed int _huge *) GlobalLock (hgMemRaw)){
      for (i=1; i<=m; i++) {
        nRecIn = _lread (hFileIn, hpMemRaw, (((sizeof (signed int ))*max)));
        hpMemRaw += max;
      }                                                                        2360
      nRecIn = _lread(hFileIn, hpMemRaw, (sizeof(signed int))*n );
    }
  }

  GlobalUnlock(hgMemRaw);
  return(hgMemRaw);
}

/* convert the raw data to appropriate scale */
void LoadMatrixF(int row, int column, MatrixF _huge * A,                       2370
                 HGLOBAL hgMemRaw, float *maxA, float *minA)
{
  signed long rc, i;
  signed int _huge *hpMemRaw;
  float tmp;
  float tmpmax, tmpmin;

  if(hpMemRaw = (signed int _huge *) GlobalLock(hgMemRaw)){
    rc = (signed long) row * column;
    tmpmax=tmpmin=*hpMemRaw/fZUnitsPerAngs/10000;                              2380
    for (i=1L; i<=rc; i++){
      tmp = *hpMemRaw/fZUnitsPerAngs/10000;
      ++hpMemRaw;
      if (tmp>tmpmax )  tmpmax=tmp ;
      if (tmp<tmpmin )  tmpmin=tmp;
      *(A->buffer+i) = tmp;
    }
  }
  *maxA = tmpmax;
  *minA=tmpmin;                                                                2390
  GlobalUnlock(hgMemRaw);
```

```
}

/* window procedure for displaying gray scale image on device context hdcMem  */
void DisplayFile(HDC hdcMem, MatrixF _huge *A, float maxA, float minA, int row, int column)
{

  int   x, y, n;
  unsigned long t = 0;
  float ftemp, grid;                                                    2400

  grid = (maxA-minA)/64;
  for( y =(row-1); y >=0; y--){
    for(x=0; x<=(column-1); x++){
      t = t + 1L;
      ftemp = *(A->buffer+t);
      n = floor((ftemp-minA)/grid);

      SetPixel(hdcMem, x, y, PALETTEINDEX(n));
    }                                                                   2410
  }
  return;
}

/* window procedure for SaveAs dialog box */
BOOL FAR PASCAL _export SaveAsDlg(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
  switch(message) {
  case WM_INITDIALOG:
    return TRUE;                                                        2420
  case WM_COMMAND:
    switch(wParam) {
    case IDOK:
      GetDlgItemText(hDlg, ID_FILENAME, filename, 64);
      EndDialog(hDlg, TRUE);
      return TRUE;
    case IDCANCEL:
      EndDialog(hDlg, 0);
      return TRUE;
    }                                                                   2430
    break;
  }
  return FALSE;
}

/* window procedure for Save3D dialog box */
BOOL FAR PASCAL _export Save3DDlg(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
  switch(message) {
  case WM_INITDIALOG:                                                   2440
    return TRUE;
  case WM_COMMAND:
    switch(wParam) {
    case IDOK:
      GetDlgItemText(hDlg, ID_FILENAME3D, filename3D, 64);
```

```
      EndDialog(hDlg, TRUE);
      return TRUE;
    case IDCANCEL:
      EndDialog(hDlg, 0);
      return TRUE;                                                    2450
    }
    break;
  }
  return FALSE;
}

/* window procedure for setup window dialog box */
BOOL FAR PASCAL _export SetupDlg(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
  static WORD tmpfilterSelect, tmpreconSelect;                        2460
  float tmp;
  char cbuf[20];
  FILE *tmpfile;
  int itmp;

  switch(message) {
  case WM_INITDIALOG:
    sprintf(cbuf,"%7.4f", thetaA);
    SetDlgItemText(hDlg, ID_TILTA, cbuf);
    sprintf(cbuf,"%7.4f", thetaB);                                    2470
    SetDlgItemText(hDlg, ID_TILTB, cbuf);
    sprintf(cbuf,"%4.4f", threshold1);
    SetDlgItemText(hDlg, ID_THRESHOLD1, cbuf);
    sprintf(cbuf,"%4.4f", threshold2);
    SetDlgItemText(hDlg, ID_THRESHOLD2, cbuf);
    sprintf(cbuf,"%7.4f", offset1);
    SetDlgItemText(hDlg, ID_OFFSET1, cbuf);
    sprintf(cbuf,"%7.4f", offset2);
    SetDlgItemText(hDlg, ID_OFFSET2, cbuf);
    sprintf(cbuf,"%7.4f", slope1);                                    2480
    SetDlgItemText(hDlg, ID_SLOPE1, cbuf);

    CheckRadioButton(hDlg, ID_MEDIANFILTER, ID_WIENERFILTER, filterSelect);
    tmpfilterSelect=filterSelect;
    CheckRadioButton(hDlg, ID_RECONS1, ID_RECONS2, reconSelect);
    tmpreconSelect=reconSelect;
  case WM_SETFOCUS:
    SetFocus(GetDlgItem(hDlg, filterSelect));
    return FALSE;
  case WM_COMMAND:                                                    2490
    switch(wParam) {
    case IDOK:
      GetDlgItemText(hDlg,ID_TILTA, cbuf, 20);
      thetaA=atof(cbuf);
      GetDlgItemText(hDlg,ID_TILTB, cbuf, 20);
      thetaB=atof(cbuf);

      GetDlgItemText(hDlg,ID_THRESHOLD1, cbuf, 20);
      tmp=atof(cbuf);
```

```
  if (tmp>0&&tmp<1)                                                    2500
    threshold1=tmp;
  GetDlgItemText(hDlg,ID_THRESHOLD2, cbuf, 20);
  tmp=atof(cbuf);
  if (tmp>0&&tmp<1)
    threshold2=tmp;

  GetDlgItemText(hDlg,ID_OFFSET1, cbuf, 20);
  tmp=atof(cbuf);
  if (tmp>0) offset1=tmp;
  GetDlgItemText(hDlg,ID_OFFSET2, cbuf, 20);                           2510
  tmp=atof(cbuf);
  if (tmp>0) offset2=tmp;

  GetDlgItemText(hDlg,ID_SLOPE1, cbuf, 20);
  tmp=atof(cbuf);
  if(tmp<0) slope1=tmp;
  EndDialog(hDlg, TRUE);

  filterSelect=tmpfilterSelect;
  reconSelect=tmpreconSelect;                                          2520
  return TRUE;
case IDCANCEL:
  EndDialog(hDlg, 0);
  return TRUE;
case ID_SETUPSAVE :
  SendMessage(hDlg, WM_COMMAND, IDOK, 0L);
  if ( (tmpfile=fopen("afmparam.def", "w")) == NULL )
    MessageBox(NULL, "Cannot write to afmparam.def", "Warning", MB_OK);
  else {
    SetCursor( LoadCursor (NULL, IDC_WAIT));                           2530
    itmp=filterSelect-ID_MEDIANFILTER+1;
    fprintf(tmpfile, "%d \n", itmp);
    itmp=reconSelect-ID_RECONS1+1;
    fprintf(tmpfile, "%d \n", itmp);

    fprintf(tmpfile, "%f \n", thetaA );
    fprintf(tmpfile, "%f \n", thetaB );
    fprintf(tmpfile, "%f \n", threshold1 );
    fprintf(tmpfile, "%f \n", offset1 );
    fprintf(tmpfile, "%f \n", slope1);                                 2540
    fprintf(tmpfile, "%f \n", threshold2 );
    fprintf(tmpfile, "%f \n", offset2 );
    fclose(tmpfile);
    SetCursor( LoadCursor (NULL, IDC_ARROW));
  }
  return TRUE;
case ID_MEDIANFILTER:
case ID_WIENERFILTER:
  tmpfilterSelect=wParam;
  CheckRadioButton(hDlg, ID_MEDIANFILTER, ID_WIENERFILTER, tmpfilterSelect);2550
  return FALSE;
case ID_RECONS1:
case ID_RECONS2:
```

```
      tmpreconSelect=wParam;
      CheckRadioButton(hDlg, ID_RECONS1, ID_RECONS2, tmpreconSelect);
      return FALSE;
    }
    break;
  }
  return FALSE;                                                          2560
}

/*
 *  AboutDlgProc -- Procedure for displaying the ABOUT Dialog Box for the AFM  program
 */
BOOL FAR PASCAL _export AboutDlgProc (HWND hDlg, UINT message, UINT wParam, LONG lParam)
{
  switch(message)
    {
    case WM_INITDIALOG:                                                  2570
      return TRUE;
    case WM_COMMAND:
      switch(wParam)
        {
        case IDOK:
        case IDCANCEL:
          EndDialog(hDlg, 0);
          return TRUE;
        }
      break;                                                             2580
    }
  return FALSE;
}

/*
 *  SelectA_DlgProc -- Procedure for displaying the Data Selection Dialog Box
 *                     for the AFM  program
 */
BOOL FAR PASCAL _export SelectA_DlgProc (HWND hDlg, UINT message, UINT wParam, LONG lParam)
{                                                                        2590
  static HWND hListbox1, hListbox2, hListbox3;  // handles for list box controls in dialog bc
  static HANDLE hEdit1, hEdit2, hEdit3; // handles for edit fields in dialog box
// cursors for listbox selection and positioning routines
  static int nCurSel1 =0, nOldSel1 = 0, ncurSel2 = 0;
// cursors for listbox selection and positioning routines
  static int nCurSel3 = 0, nOldSel3 = 0;
  DBIResult  rslt;       // Return value from IDAPI functions
  hDBIDb     hDb;        // Handle to the database
  hDBICur    hCur;       // Handle to the table
  CURProps   curProps;    // Properties of the cursor                    2600
  pBYTE    pRecBuf;        // Record buffer
  BOOL     bIsBlank;           // Determine if a field is blank
  hDBIStmt   hStmt;      // Handle to the SQL statement
  CHAR       szQry[]     = { // The text of the SQL statement
                           "SELECT samplename,imagefile,comment \r\n"
                            "FROM \"scantbl.db\" \r\n"
                          };
```

```
switch(message) {
case WM_INITDIALOG:                                                    2610
  // Set default radio button values
  bBoth = TRUE;
  bSelSideA = FALSE;
  bSelSideB = TRUE;
  CheckRadioButton(hDlg, IDC_RADIOBUTTON1, IDC_RADIOBUTTON2, IDC_RADIOBUTTON1);
  CheckRadioButton(hDlg, IDC_RADIOBUTTON3, IDC_RADIOBUTTON4, IDC_RADIOBUTTON4);

  // Get handles to relevant dialog box controls
  hListbox1 = GetDlgItem(hDlg, IDC_LISTBOX1); // list box containing dbkey - date & session
  hListbox2 = GetDlgItem(hDlg, IDC_LISTBOX2); // list box containing imagefile names   2620
  hListbox3 = GetDlgItem(hDlg, IDC_LISTBOX3); // list box containing comments
  hEdit1 =  GetDlgItem(hDlg, IDC_EDIT1); // edit control for selected dbkey - date & session
  hEdit2 = GetDlgItem(hDlg, IDC_EDIT2);  // edit control for side A imagefile name
  hEdit3 = GetDlgItem(hDlg, IDC_EDIT3);  // edit control for side B imagefile name

     // Initialize and connect to IDAPI
  if (InitAndConnect(&hDb) != DBIERR_NONE){
    MessageBox (hDlg, "Unable to connect to database.
       Please quit program and notify programmer.", "Message", MB_OK);
    return TRUE;                                                       2630
  }

  // Setup the directory path
  rslt = DbiSetDirectory(hDb, (pCHAR) szTblDirectory);
  ChkRslt(rslt, "SetDirectory");


  // Prepare the SQL statement
    rslt = DbiQPrepare(hDb, qrylangSQL, szQry, &hStmt);
  if (ChkRslt(rslt, "QPrepare") != DBIERR_NONE) {                      2640
    CloseDbAndExit(&hDb);
    MessageBox (hDlg, "*** Error preparing SQL statement ***", "Message", MB_OK);
    return TRUE:
  }

  //Execute the SQL statement
    rslt = DbiQExec(hStmt, &hCur);
  if (ChkRslt(rslt, "QExec") != DBIERR_NONE) {
    CloseDbAndExit(&hDb);
    MessageBox (hDlg, "*** Error executing SQL statement ***", "Message", MB_OK); 2650
    return TRUE;
  }

  // Check for a valid cursor and set it to the beginning of the result table.
    if (hCur) {
        rslt = DbiSetToBegin(hCur);
        ChkRslt(rslt, "SetToBegin");

        // Determine the size of the record
          rslt = DbiGetCursorProps(hCur, &curProps);                   2660
```

```
    // Allocate memory for the record buffer
      pRecBuf = (pBYTE)malloc(curProps.iRecBufSize);

    if(pRecBuf == NULL){
      MessageBox (hDlg, "Not enough memory for record buffer.
          Please quit program and notify programmer.", "Message", MB_OK);
      DbiCloseDatabase(&hDb);
      DbiExit();
      return TRUE;                                                    2670
    }

    // Get the records from the table and fill the list boxes in the dialog
      while (DbiGetNextRecord(hCur, dbiNOLOCK, pRecBuf, NULL) == DBIERR_NONE){
        // Get the field from the record buffer
          rslt = DbiGetField(hCur, 1, pRecBuf, (pBYTE) szSampleName, &bIsBlank);
        rslt = DbiGetField(hCur, 2, pRecBuf, (pBYTE) szImageFile, &bIsBlank);
        rslt = DbiGetField(hCur, 3, pRecBuf, (pBYTE) szComment, &bIsBlank);
        // Add fields to appropriate list boxes
        SendMessage(hListbox1, LB_ADDSTRING, 0, szSampleName);        2680
        SendMessage(hListbox2, LB_ADDSTRING, 0, szImageFile);
        SendMessage(hListbox3, LB_ADDSTRING, 0, szComment);
      }
    // Set the cursor to the top of the list boxes and fill the edit
      // fields with the these listbox entries
        SendMessage(hListbox1, LB_SETCURSEL, nCurSel1, 0L);
    SendMessage(hListbox2, LB_SETCURSEL, nCurSel2, 0L);
    SendMessage(hListbox3, LB_SETCURSEL, nCurSel3, 0L);
    //lstrcpy(szImageFileA, "");
    //lstrcpy(szImageFileB, "");                                      2690
    if(bSelSideA){    // if selecting for side A
                    SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameA);
                    SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileA);
                    SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
                    SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
    }
    else {      // if selecting for side B
                  SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameB);
                  SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileB);
                  SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);     2700
                  SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
    }

    // Release memory allocated for the record buffer
      free(pRecBuf);

    //Close the cursor to the answer set.
      rslt = DbiCloseCursor(&hCur);
    ChkRslt(rslt, "CloseCursor");
  }                                                                   2710
  else{
    MessageBox (hDlg, "Could not get cursor to the answer table", "Message", MB_OK);
  }

//Release memory allocated for the query.
```

```
    rslt = DbiQFree(&hStmt);
  ChkRslt(rslt, "QryFree");

  //Close the database and exit IDAPI
    CloseDbAndExit(&hDb);                                                    2720
  return TRUE;
case WM_COMMAND:
  switch(wParam) {
  case IDC_LISTBOX1:
    nOldSel1 = nCurSel1;
    nCurSel1 = (int)SendMessage(hListbox1, LB_GETCURSEL, 0, 0L);
    if (nCurSel1 == nOldSel1){
        nCurSel2 = (int)SendMessage(hListbox2, LB_GETCURSEL, 0, 0L);
        nCurSel3 = (int)SendMessage(hListbox3, LB_GETCURSEL, 0, 0L);
        nCurSel1 = nCurSel3;                                                 2730
        SendMessage(hListbox1, LB_SETCURSEL, nCurSel1, 0L);
    }
    else {
        nCurSel2 = nCurSel1;
        nCurSel3 = nCurSel1;
        SendMessage(hListbox2, LB_SETCURSEL, nCurSel2, 0L);
        SendMessage(hListbox3, LB_SETCURSEL, nCurSel3, 0L);
    }
    if(bSelSideA){
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameA);         2740
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileA);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
        SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
    }
    if(bSelSideB){
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameB);
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileB);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);
        SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
    }                                                                        2750
    break;
    /*case IDC_LISTBOX2:
        break;    */
  case IDC_LISTBOX3:
    nOldSel3 = nCurSel3;
    nCurSel3 = (int)SendMessage(hListbox3, LB_GETCURSEL, 0, 0L);
    if (nCurSel3 == nOldSel3){
        nCurSel1 = (int)SendMessage(hListbox1, LB_GETCURSEL, 0, 0L);
        nCurSel2 = (int)SendMessage(hListbox2, LB_GETCURSEL, 0, 0L);
        nCurSel3 = nCurSel1;                                                 2760
        SendMessage(hListbox3, LB_SETCURSEL, nCurSel3, 0L);
    }
    else {
        nCurSel1 = nCurSel3;
        nCurSel2 = nCurSel3;
        SendMessage(hListbox1, LB_SETCURSEL, nCurSel1, 0L);
        SendMessage(hListbox2, LB_SETCURSEL, nCurSel2, 0L);
    }
    if(bSelSideA){
```

```
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameA);
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileA);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
        SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
    }
    if(bSelSideB){
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameB);
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileB);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);
        SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
    }
    break;
case IDC_RADIOBUTTON1:    // Radio button used to set select both sides option
    bBoth = TRUE;
    CheckRadioButton(hDlg, IDC_RADIOBUTTON1, IDC_RADIOBUTTON2, IDC_RADIOBUTTON1);
    break;
case IDC_RADIOBUTTON2:// Radio button used to set the option for selecting only one side
    bBoth = FALSE;
    CheckRadioButton(hDlg, IDC_RADIOBUTTON1, IDC_RADIOBUTTON2, IDC_RADIOBUTTON2);
    if(bSelSideA){
        lstrcpy(szSampleNameB, "");
        lstrcpy(szImageFileB, "");
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);
        SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameA);
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileA);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
        SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
    }
    else {
        lstrcpy(szSampleNameA, "");
        lstrcpy(szImageFileA, "");
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
        SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
        SendMessage(hListbox1, LB_GETTEXT, nCurSel1, szSampleNameB);
        SendMessage(hListbox2, LB_GETTEXT, nCurSel2, szImageFileB);
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);
        SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
    }
    break;
case IDC_RADIOBUTTON3:    // Radio button used to make selection for side A
    bSelSideA = TRUE;
    bSelSideB = FALSE;
    CheckRadioButton(hDlg, IDC_RADIOBUTTON3, IDC_RADIOBUTTON4, IDC_RADIOBUTTON3);
    if(!bBoth){
        lstrcpy(szSampleNameB, "");
        lstrcpy(szImageFileB, "");
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameB);
        SetDlgItemText(hDlg, IDC_EDIT3, szImageFileB);
    }
    break;
case IDC_RADIOBUTTON4:    // Radio button used to make selection for side B
    bSelSideA = FALSE;
    bSelSideB = TRUE;
```

```
    CheckRadioButton(hDlg, IDC_RADIOBUTTON3, IDC_RADIOBUTTON4, IDC_RADIOBUTTON4);
    if(!bBoth){
        lstrcpy(szSampleNameA, "");
        lstrcpy(szImageFileA, "");
        SetDlgItemText(hDlg, IDC_EDIT1, szSampleNameA);
        SetDlgItemText(hDlg, IDC_EDIT2, szImageFileA);
    }                                                                    2830
    break;
   case IDC_OK:
    if(bBoth && (!(lstrcmp(szImageFileA, "")) || !(lstrcmp(szImageFileB, "")))){
        MessageBox(hDlg, "Please make a selection for the other side of the blade.",
                "Warning", MB_ICONHAND | MB_OK);
        break;
    }
    EndDialog(hDlg, 0);
    return TRUE;
   case IDCANCEL:                                                        2840
    lstrcpy(szImageFileA, "*.*");
    lstrcpy(szImageFileB, "*.*");
    bCancelSelDlg = TRUE;
    EndDialog(hDlg, 0);
    return TRUE;
   }
   break;
  }
  return FALSE;
}                                                                        2850

/* BDE FUNCTIONS AND CODE */
//=========================================
// Function:
//        TableOpen(HWND hInformation);
//
// Description:
//        This example shows how to open and close a table.
//=========================================
void TableOpen (HWND hInformation)                                       2860
{
  DBIResult  rslt;       // Return value from IDAPI functions
  hDBIDb     hDb;        // Handle to the database
  hDBICur    hCur;       // Handle to the table
  CURProps   curProps;   // Properties of the cursor
  pBYTE      pRecBuf;    // Record buffer
  BOOL       bIsBlank;   // Determine if a field is blank
  hDBIStmt   hStmt;      // Handle to the SQL statement
  CHAR       szQryWhere1[]={"WHERE (imagefile LIKE '"};
  CHAR       szQryWhere3[]={"%')"};                                      2870
  CHAR       szQryOrder[] = {"ORDER BY imagefile, comment, xpoints, zunitsperangs"};
  CHAR       szQrySelect[] = {"SELECT * \r\n"};
  CHAR       szQryFrom[] = {"FROM \"scantbl.db\" \r\n"};
  CHAR       szQry[200];

  //DISTINCT dbkey, samplename, imagefile, comments, xpoints, ypoints, zunitsperangs
  // Set up query
```

```
memset(szQry, '\0', 200);
lstrcpy(szQry, szQrySelect);
lstrcat(szQry, szQryFrom);                                                    2880
lstrcat(szQry, szQryWhere1);
lstrcat(szQry, szImageFile);
lstrcat(szQry, szQryWhere3);
lstrcat(szQry, szQryOrder);

// Initialize and connect to IDAPI
if (InitAndConnect(&hDb) != DBIERR_NONE)
 {
  MessageBox (hInformation, "Unable to connect to database.
    Please quit program and notify programmer.", "Message", MB_OK);          2890
  return;
 }

// Setup the directory path
rslt = DbiSetDirectory(hDb, (pCHAR) szTblDirectory);
ChkRslt(rslt, "SetDirectory");

rslt = DbiQPrepare(hDb, qrylangSQL, szQry, &hStmt);
if (ChkRslt(rslt, "QPrepare") != DBIERR_NONE)
 {                                                                            2900
  CloseDbAndExit(&hDb);
  MessageBox (hInformation, "*** End of Searching ***", "Message", MB_OK);
  return;
 }

//Screen("     Execute the SQL statement...");
//MessageBox (hInformation, "Execute the SQL statement...", "Message", MB_OK);
rslt = DbiQExec(hStmt, &hCur);
if (ChkRslt(rslt, "QExec") != DBIERR_NONE)
 {                                                                            2910
  CloseDbAndExit(&hDb);
  MessageBox (hInformation, "*** End of Searching ***", "Message", MB_OK);
  return;
 }

// Check for a valid cursor.
 if (hCur)
  {
      rslt = DbiSetToBegin(hCur);
      ChkRslt(rslt, "SetToBegin");                                            2920
      // Determine the size of the record
      rslt = DbiGetCursorProps(hCur, &curProps);

      // Allocate memory for the record buffer
      pRecBuf = (pBYTE)malloc(curProps.iRecBufSize);

      if(pRecBuf == NULL){
       MessageBox (hInformation, "Not enough memory for record buffer.
         Please quit program and notify programmer.", "Message", MB_OK);
       DbiCloseDatabase(&hDb);                                                2930
       DbiExit();
```

```
        return;
        }
        // Get the record from the table
        rslt = DbiGetNextRecord(hCur, dbiNOLOCK, pRecBuf, NULL);
        ChkRslt(rslt, "GetNextRecord");

        // Get the field from the record buffer
          // Note: normally, want to use DbiGetFieldDescs to get the
            // field number of the desired field (ie don't hard code except in examples   2940
              rslt = DbiGetField(hCur, 2, pRecBuf, (pBYTE) szDbKey, &bIsBlank);
              rslt = DbiGetField(hCur, 3, pRecBuf, (pBYTE) szSampleName, &bIsBlank);
              rslt = DbiGetField(hCur, 4, pRecBuf, (pBYTE) szImageFile, &bIsBlank);
              rslt = DbiGetField(hCur, 5, pRecBuf, (pBYTE) szComment, &bIsBlank);
              rslt = DbiGetField(hCur, 10, pRecBuf, &nXPoints, &bIsBlank);
              rslt = DbiGetField(hCur, 11, pRecBuf, &nYPoints, &bIsBlank);
              rslt = DbiGetField(hCur, 12, pRecBuf, &fXArea, &bIsBlank);
              rslt = DbiGetField(hCur, 13, pRecBuf, &fYArea, &bIsBlank);
              rslt = DbiGetField(hCur, 41, pRecBuf, &fZUnitsPerAngs, &bIsBlank);
                                                                                  2950
                                  // Release memory allocated for the record buffer
              free(pRecBuf);
              rslt = DbiCloseCursor(&hCur);
              ChkRslt(rslt, "CloseCursor");
        }
        else
        {
      MessageBox (hInformation, "Could not get cursor to the answer table", "Message", MB_OK);
        }
                                                                                  2960
    //MessageBox (hInformation, "Release memory allocated for the query...", "Message", MB_OK);
        rslt = DbiQFree(&hStmt);
        ChkRslt(rslt, "QryFree");

        CloseDbAndExit(&hDb);
}

// BDE - (C) Copyright 1994 by Borland International
// sniptool.c
                                                                                  2970
#define MAXLEN 50

//============================================================
//  Function:
//          ChkRslt(rslt, pMsg);
//  Input:    rslt    - Return code from IDAPI
//          pMsg    - Null-terminated message
//
//  Return:  IDAPI return code passed in
//                                                                                  2980
//  Description:
//          This will echo an error message if the expected
//          result does not equal the actual result.  The output will be
//          echoed to the screen.
//============================================================
```

```
DBIResult
ChkRslt (DBIResult rslt, pCHAR pMsg)
{
  DBIMSG  dbi_status;
  DBIErrInfo  ErrInfo;                                                    2990

  // Echo only if actual result doesn't equal expected result
        if (rslt != DBIERR_NONE)
        {
                // Get as much error information as possible
                DbiGetErrorInfo(TRUE, &ErrInfo);

                // Make certain information is returned on the correct error
                if (~(ErrInfo.iError == rslt))
                        DbiGetErrorString(rslt, dbi_status);              3000
        }
        return rslt;
}

//=======================================
// Function:
//      InitAndConnect(phDb);
//
// Input:  phDb - Pointer to the database handle
//                                                                         3010
// Return: IDAPI return code after DbiInit() and DbiOpenDatabase()
//
// Description:
//      Initialize IDAPI and connect to a Standard
//      database (non-SQL).
//=======================================
DBIResult InitAndConnect(phDBIDb phDb)
{
 DBIResult   rslt;
                                                                         3020
  // Initialize IDAPI with a NULL environment structure
   rslt = DbiInit(NULL);
 if (ChkRslt(rslt, "Init") == DBIERR_NONE)
   {
    DbiDebugLayerOptions(DEBUGON | OUTPUTTOFILE, "SNIPIT.INF");

   // IDAPI initialized. Now open a Standard database (used to access
   //   Paradox and dBASE tables), by using a NULL database type.
   rslt = DbiOpenDatabase(NULL, NULL, dbiREADWRITE, dbiOPENSHARED, NULL,
                          0, NULL, NULL, phDb);                          3030
   if (ChkRslt(rslt, "OpenDatabase") != DBIERR_NONE)
   {
        rslt = DbiExit();
        ChkRslt(rslt, "Exit");
   }
   rslt = DbiSetPrivateDir(szPrivDirectory);
   if (ChkRslt(rslt, "SetPrivateDir") != DBIERR_NONE)
        {
         CloseDbAndExit(phDb);
```

```
        }                                                              3040
    }
  return rslt;
}

//=============================================
// Function:
//        CloseDbAndExit(phDb);
//
// Input:  phDb - Pointer to the database handle
//                                                                     3050
// Return: IDAPI return code after DbiCloseDatabase() and DbiExit()
//
// Description:
//        Close the supplied database and exit IDAPI.
//=============================================
DBIResult CloseDbAndExit (phDBIDb phDb)
{
 DBIResult   rslt;

 // Close the supplied database                                        3060
   rslt = DbiCloseDatabase(phDb);
 if (ChkRslt(rslt, "CloseDatabase") == DBIERR_NONE)
   {
    DbiDebugLayerOptions(0, NULL);

    rslt = DbiExit();
    ChkRslt(rslt, "Exit");
   }
 return rslt;
}                                                                    3070
```

D.1.4 afmproj5.rc

```
/*** afmproj.rc resource file ***/

#include "afmproj5.h"


MyIcon          ICON            afmproj.ico
AFMLOGO         BITMAP          afm.bmp


AboutBox        DIALOG 80,80,160,80                                  10
                STYLE WS_POPUP | WS_DLGFRAME
                {
                CTEXT "AFM Blade Measurement"          -1, 0,12,160.8
                ICON  "MyIcon"                         -1,8,8,0,0
                CTEXT "AFM Blade Measurement Program" -1,0,36,160.8
                CTEXT "(c) Gillette, Inc.      1994" -1,0,48,160,8
```

```
                DEFPUSHBUTTON "OK"                  IDOK,64,60,32,14,WS_GROUP
                }

SaveAsBox DIALOG 20, 20, 160, 40                                              20
STYLE  WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "Save As"
{
        LTEXT "&Output File Name"    -1, 5, 5, 160, 8
        EDITTEXT                ID_FILENAME, 5, 15, 115, 11, ES_AUTOHSCROLL
        DEFPUSHBUTTON "OK"      IDOK 125, 5, 30, 12, WS_GROUP
        PUSHBUTTON "Cancel"     IDCANCEL 125, 20, 30, 12, WS_GROUP
}

Save3DDlg DIALOG 20, 20, 160, 40                                              30
STYLE  WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "Save 3D"
{
        LTEXT "&Output File Name"    -1, 5, 5, 160, 8
        EDITTEXT                ID_FILENAME3D, 5, 15, 115, 11, ES_AUTOHSCROLL
        DEFPUSHBUTTON "OK"      IDOK 125, 5, 30, 12, WS_GROUP
        PUSHBUTTON "Cancel"     IDCANCEL 125, 20, 30, 12, WS_GROUP
}

SetupDlg DIALOG 30, 30, 200, 160                                              40
STYLE WS_POPUP|WS_DLGFRAME
{
        GROUPBOX "&Filter", -1, 5, 2, 75, 33
        RADIOBUTTON "&Median",  ID_MEDIANFILTER, 10, 10, 45, 12
        RADIOBUTTON "&Wiener",  ID_WIENERFILTER, 10, 20, 45, 12
        GROUPBOX "&Recon_Edge", -1, 5, 47, 75, 38
        RADIOBUTTON "&1st Derivative", ID_RECONS1, 10, 60, 65, 12
        RADIOBUTTON "&2nd Derivative", ID_RECONS2, 10, 70, 65, 12
        GROUPBOX "&Tilt Angle", -1, 5, 95, 75, 53
        LTEXT "Left:",          -1, 10, 110, 30, 11                           50
        EDITTEXT                ID_TILTA 35, 110, 40, 12, ES_AUTOHSCROLL
        LTEXT "Right:",         -1, 10, 125, 30, 11
        EDITTEXT                ID_TILTB 35, 125, 40, 12, ES_AUTOHSCROLL
        DEFPUSHBUTTON "Save", ID_SETUPSAVE, 120, 3, 50, 14, WS_GROUP
        PUSHBUTTON "OK(not Save)" , IDOK, 120, 18,50, 14, WS_GROUP
        PUSHBUTTON "Cancel",  IDCANCEL, 120, 33, 50, 14, WS_GROUP
        GROUPBOX "&First Derivative", -1, 90, 50, 100, 55
        LTEXT "Threshold",      -1, 100, 60, 40, 11
        EDITTEXT                ID_THRESHOLD1 140, 60, 40, 12, ES_AUTOHSCROLL
        LTEXT "Offset",         -1, 100, 75, 40, 11                           60
        EDITTEXT                ID_OFFSET1 140, 75, 40, 12, ES_AUTOHSCROLL
        LTEXT "Slope",          -1, 100, 90, 40, 11
        EDITTEXT                ID_SLOPE1 140, 90, 40, 12, ES_AUTOHSCROLL
        GROUPBOX "&Second Derivative", -1, 90, 108, 100, 45
        LTEXT "Threshold",      -1, 100, 120, 40, 11
        EDITTEXT                ID_THRESHOLD2 140, 120, 40, 12, ES_AUTOHSCROLL
        LTEXT "Offset",         -1, 100, 135, 40, 11
        EDITTEXT                ID_OFFSET2 140, 135, 40, 12, ES_AUTOHSCROLL
}
                                                                              70
```

```
MyMenu MENU
BEGIN
        POPUP "&File"
        BEGIN
                MENUITEM "&Open", IDM_OPENPROJECT
                MENUITEM "&Close", IDM_FILECLOSE
                MENUITEM "Save &Data", IDM_SAVE
                MENUITEM "Save Data &as...", IDM_SAVEAS
                MENUITEM "Save 3D Data...", IDM_SAVE3D
                MENUITEM SEPARATOR                                          80
                MENUITEM "E&xit", IDM_QUIT
        END

        POPUP "&Analysis"
        BEGIN
                MENUITEM "Apply &Filter", IDM_FILTER, GRAYED
                MENUITEM "&Reconstruct Blade Edge", IDM_RECONSTRUCT, GRAYED
                MENUITEM "Calculate Blade &Profile", IDM_2DPROFILE, GRAYED
                MENUITEM "&Average Profile (All)", IDM_2DAVGPROFILE, GRAYED
                MENUITEM "Calculate 3D &Blade Profile", IDM_3DPROFILE, GRAYED90
        END

        POPUP "&Clear Displays"
        BEGIN
                MENUITEM "Clear Frame &A", IDM_CLEARFRAMEA, GRAYED
                MENUITEM "Clear Frame &B", IDM_CLEARFRAMEB, GRAYED
                MENUITEM "Clear &Main Viewing Frame", IDM_CLEARMAINFRAME, GRAYED
                MENUITEM SEPARATOR
                MENUITEM "Clear &All", IDM_CLEARALL, GRAYED
        END                                                                 100

        POPUP "&View"
        BEGIN
                MENUITEM "View &Raw Image A", IDM_VIEWRAWA, GRAYED
                MENUITEM "View Ra&w Image B", IDM_VIEWRAWB, GRAYED
                MENUITEM "View Filtered &Image A", IDM_VIEWFILTEREDA, GRAYED
                MENUITEM "View Filtered I&mage B", IDM_VIEWFILTEREDB, GRAYED
                MENUITEM "View Reconstructed &Edge A", IDM_VIEWRECONSTRUCTA, GRAYED
                MENUITEM "View Reconstructed Edg&e B", IDM_VIEWRECONSTRUCTB, GRAYED
                MENUITEM "&View 2D Blade Profile", IDM_VIEW2DPROFILE, GRAYED110
                MENUITEM "View 3D Blade &Profile", IDM_VIEW3DPROFILE, GRAYED
                MENUITEM "View Average Thickness &Graph", IDM_VIEW2DAVGPROFILE, GRAYED
        END

        MENUITEM "&Setup", IDM_SETUP

        POPUP "&Information"
        BEGIN
                MENUITEM "Side &A", IDM_INFORMATIONA
                MENUITEM "Side &B", IDM_INFORMATIONB                        120
        END

        MENUITEM "A&bout...", IDM_HELPABOUT
```

```
END

STRINGTABLE
BEGIN
        S_PROGRAMCAPTION          "AFM Blade Profile Measurement"
        S_PRINTDRIVERPROBLEM "Could not load Printer Driver - terminating"  130
END
#include "sela_dlg.rc"
```

D.1.5 sela_dlg.rc

```
/*************************************
sel_idlg.rc
produced by Borland Resource Workshop
 *************************************/

#define IDC_LISTBOX1            101
#define IDC_LISTBOX2            102
#define IDC_EDIT1               103
#define IDC_EDIT2               104
#define IDC_RADIOBUTTON1        105                                          10
#define IDC_RADIOBUTTON2        107
#define IDC_GROUPBOX1           108
#define IDC_GROUPBOX2           110
#define IDC_RADIOBUTTON3        111
#define IDC_RADIOBUTTON4        112
#define IDC_LISTBOX3            109
#define IDC_EDIT3               106
#define IDC_OK                  1000

SelectA_DlgBox DIALOG 19, 18, 246, 194                                       20
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Sample Selection"
FONT 8, "MS Sans Serif"
{
 DEFPUSHBUTTON "OK", IDC_OK, 125, 174, 50, 14
 PUSHBUTTON "Cancel", IDCANCEL, 179, 174, 50, 14
 LISTBOX IDC_LISTBOX1, 16, 69, 95, 95, LBS_NOTIFY |
WS_BORDER | LBS_HASSTRINGS | LBS_DISABLENOSCROLL | WS_BORDER | WS_VSCROLL
 LISTBOX IDC_LISTBOX2, 132, 69, 96, 95, LBS_NOTIFY | WS_BORDER | LBS_HASSTRINGS
| LBS_DISABLENOSCROLL | NOT WS_VISIBLE | WS_BORDER | WS_VSCROLL            30
 LISTBOX IDC_LISTBOX3, 132, 69, 96, 95, LBS_NOTIFY
| WS_BORDER | LBS_HASSTRINGS | LBS_DISABLENOSCROLL | WS_BORDER | WS_VSCROLL
 EDITTEXT IDC_EDIT1, 16, 38, 97, 13
 EDITTEXT IDC_EDIT2, 132, 12, 98, 13
 EDITTEXT IDC_EDIT3, 132, 38, 98, 13
 LTEXT "Selected sample name:", -1, 16, 29, 81, 8
 LTEXT "Left image file:", -1, 133, 3, 84, 9
 LTEXT "Right image file:", -1, 133, 29, 84, 8
 LTEXT "Comments:", -1, 133, 59, 74, 8
 LTEXT "Sample name selections:", -1, 16, 59, 95, 8                          40
 CONTROL "&Both Sides", IDC_RADIOBUTTON1, "BUTTON", BS_AUTORADIOBUTTON | WS_GROUP,
        19, 172, 46, 12
```

```
 CONTROL "&One Side", IDC_RADIOBUTTON2, "BUTTON", BS_AUTORADIOBUTTON,
        69, 172, 40, 12
 GROUPBOX "Selection Options", IDC_GROUPBOX1, 16, 162, 95, 26, BS_GROUPBOX
 GROUPBOX "Select", IDC_GROUPBOX2, 16, 3, 96, 23, BS_GROUPBOX
 CONTROL "Left", IDC_RADIOBUTTON3, "BUTTON", BS_AUTORADIOBUTTON | WS_GROUP,
        21, 12, 34, 12
 CONTROL "Right", IDC_RADIOBUTTON4, "BUTTON", BS_AUTORADIOBUTTON, 72, 12, 34, 12
}
```

D.2 matrxlib.ide

D.2.1 matrxlib.def

```
LIBRARY      matrxlib
DESCRIPTION  'matrix operations dll'
EXETYPE      WINDOWS
STUB         'WINSTUB.EXE'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE DISCARDABLE SINGLE
HEAPSIZE     1024
```

D.2.2 matrxlib.h

```
/*********************************************************************
 Header File : matrxlib.h
 Description :
        This is the prototypes of the subroutines and the declarations
        of constant variables for program 'matrxlib.c'

 Written by C. J. Chiu
 Dec. 1994
 *********************************************************************/
#define PI 3.1415962

/* Macro definition ( from 'Windowsx.h' ) */
#define    GetInstanceModule(hInstance) \
               GetModuleHandle((LPCSTR)MAKELP(0, hInstance))

#define    GlobalPtrHandle(lp)        \
             ((HGLOBAL)LOWORD(GlobalHandle(SELECTOROF(lp))))

#define    GlobalLockPtr(lp)          \
           ((BOOL)SELECTOROF(GlobalLock(GlobalPtrHandle(lp))))
#define    GlobalUnlockPtr(lp)        \
           GlobalUnlock(GlobalPtrHandle(lp))
```

```
#define     GlobalAllocPtr(flags, cb)   \
            (GlobalLock(GlobalAlloc((flags), (cb))))
#define     GlobalReAllocPtr(lp, cbNew, flags)  \
  (GlobalUnlockPtr(lp), GlobalLock(GlobalReAlloc(GlobalPtrHandle(lp) , (cbNew), (flags))))
#define     GlobalFreePtr(lp)           \
        (GlobalUnlockPtr(lp), (BOOL)GlobalFree(GlobalPtrHandle(lp)))
                                                                                    30
/* matrix struction definition */
typedef struct {
        int column, row;
        float _huge *buffer;
} MatrixF;

typedef struct {
        int column, row;
        int _huge *buffer;
} MatrixI;                                                                          40

/* find the address of an element M(I, J), M is a matrix */
#define MatElm(M, I, J) \
        ( (M)->buffer + (M)->column*(long)(I-1) + J )

#define NR_END 1

/* create a matrix */
MatrixF _huge * FAR PASCAL new_MatrixF(int, int);
MatrixI _huge * FAR PASCAL new_MatrixI(int, int);                                   50

/* remove a matrix */
int FAR PASCAL old_MatrixF(MatrixF _huge *);
int FAR PASCAL old_MatrixI(MatrixI _huge *);

/*  vectors manipulation */
void FAR PASCAL FVec_Add_FVec(float _huge *, float _huge *, float _huge *, int);
void FAR PASCAL FVec_Sub_FVec(float _huge *, float _huge *, float _huge *, int);
void FAR PASCAL Sca_Add_FVec(float, float _huge *, float _huge *, int);
void FAR PASCAL Max_FVecAB(float _huge *, float _huge *, float _huge *, int);       60
void FAR PASCAL Min_FVecAB(float _huge *, float _huge *, float _huge *, int);
void FAR PASCAL Min_FVecABA(float _huge *, float _huge *, int);
int FAR PASCAL Max_FVec(float _huge *, int);
int FAR PASCAL Min_FVec(float _huge *, int);
int FAR PASCAL Max_IVec(int _huge *, int);
int FAR PASCAL Min_IVec(int _huge *, int);

int FAR PASCAL FindSmaller(float _huge *, int, float);
int FAR PASCAL FindLarger(float _huge *, int, float);
                                                                                    70
void FAR PASCAL fliplr(MatrixF _huge *, MatrixF _huge *);
void FAR PASCAL table(float _huge *, float _huge *, int, float _huge *, float _huge *, int);
void FAR PASCAL sort(int n, float _huge arr[]);
void FAR PASCAL average(MatrixF _huge *, MatrixF _huge *);
void FAR PASCAL std(MatrixF _huge *, MatrixF _huge *);
```

D.2.3 matrxlib.c

```
/*************************************************************
  File : matrxlib.c
  Description : DLL subroutines for matrix manipulatioin

  Written by C. J. Chiu
  Dec. 1994
  *************************************************************/
#include <windows.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "matrxlib.h"

HANDLE ghInstance;

int FAR PASCAL LibMain (HANDLE hInstance, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
{
 HANDLE ghInstance;
 if (wHeapSize > 0)        /* unlock local data segment */
   UnlockData (0);
 ghInstance = hInstance;          /* save instance handle */
 return (1);
}

/*********** dll terminator function ***************/
int FAR PASCAL WEP (int nParameter)
{
 /* any cleanup activities go here */
 return;
}

/*
 * new_MatrixF - allocating memory for a row by column floating point matrix (Win only)
 * Syntax:        MatrixF _huge * new_MatrixF (int row, int column)
 * Parameters:
 *       int row : the number of rows in the matrix
 *       int column : the number of columns in the matrix
 * Return:        A pointer to a new float matrix structure - MatrixF
 */
MatrixF _huge * FAR PASCAL _export  new_MatrixF (int row, int column)
{
 unsigned long bytes;
 MatrixF _huge * A;

 A = (MatrixF FAR *) GlobalAllocPtr(GHND, sizeof(MatrixF));
 if (!A) {
   MessageBox( NULL, "Unable lock pointer to floating point matrix.",
            "Message" , MB_ICONHAND | MB_OK);
   return(NULL);
```

```
 }                                                                          50

 bytes = ((unsigned long)row * column + 4L) * sizeof(float);
 if(A->buffer =(float _huge *)GlobalAllocPtr (GHND, bytes)){
  A->column = column;
  A->row = row;
 }
 else {
  MessageBox( NULL, "Not enough memory available for floating point matrix.",
            "Message" , MB_ICONHAND | MB_OK);
  return(NULL);                                                             60
 }
 return(A);
}

/*
 * new_MatrixI - allocating memory for a row by column integer matrix (Win only)
 * Syntax:          MatrixI * new_MatrixI (int row, int column)
 * Parameters:
 *       int row : the number of rows in the matrix
 *       int column : the number of columns in the matrix                   70
 * Return:          A pointer to a new int matrix structure - MatrixI
 */
MatrixI _huge * FAR PASCAL _export new_MatrixI (int row, int column )
{
 unsigned long bytes;
 MatrixI _huge * A;

 A = (MatrixI FAR *) GlobalAllocPtr(GHND, sizeof(MatrixI));
 if (!A) {
  MessageBox( NULL, "Unable lock pointer to integer matrix.",               80
            "Message" , MB_ICONHAND | MB_OK);
  return(NULL);
 }

 bytes = ((unsigned long)row * column + 4L) * sizeof(int);
 if(A->buffer =(int _huge *)GlobalAllocPtr (GHND, bytes)){
  A->column = column;
  A->row = row;
 }
 else {                                                                     90
  MessageBox( NULL, "Not enough memory available for integer matrix.",
            "Message" , MB_ICONHAND | MB_OK);
  return(NULL);
 }
 return(A);
}

/*
 * old_MatrixF -- free the floating point matrix *A's memory
 * Syntax:          void old_MatrixF (MatrixF _huge *A)                     100
 * Parameters:      *A is a pointer to the matrix to be removed
 * Return:
 */
```

```
int FAR PASCAL _export old_MatrixF(MatrixF _huge *A)
{
 if (A) {
  if (A->buffer) GlobalFreePtr(A->buffer);
  GlobalFreePtr(A);
  return(1);
 }
 else {
  MessageBox( NULL, "Unable to free memory for floating matrix",
            "Message" , MB_ICONHAND | MB_OK);
  return(0);
 }
}

/*
 * old_MatrixI -- Window procedure to free the integer matrix *A's memory
 * Syntax:          old_MatrixI (MatrixI _huge *A)
 * Parameters:    *A is a pointer to the matrix to be removed
 * return :
 */
int FAR PASCAL _export old_MatrixI(MatrixI _huge *A)
{
 if (A) {
  if (A->buffer) GlobalFreePtr(A->buffer);
  GlobalFreePtr(A);
  return(1);
 }
 else {
  MessageBox(NULL, "Unable to free memory for integer matrix",
            "Message", MB_ICONHAND|MB_OK);
  return(0);
 }
}

/*
 * FVec_Add_FVec -- Add two floating vectors A[1..n] and B[1..n]
 * Syntax:          void FVec_Add_FVec (float *A, float *B, float *C, int n)
 * Parameters:      float *A : a pointer to the first vector to be added
 *                  float *B : a pointer to the second vector to be added
 *                  float *c : a pointer to the vector that contains the
 *                                  result of vector (A+B)
 *                  int n : the number of elements in the vector (i.e. the vector length)
 * Return:
 */
void FAR PASCAL _export FVec_Add_FVec(float _huge *A, float _huge *B, float _huge *C, int n)
{
 int _IX;
 for (_IX=1; _IX<= n; _IX++)
  *(++C) = *(++A) + *(++B);
}

/*
 * FVec_Sub_FVec -- Subtract floating vector B[1..n] from vector A[1..n]
 * Syntax             void FVec_Sub_FVec (float *A, float *B, float *C, int n)
```

```
 * Parameters:      float *A : a pointer to the  minuend vector
 *                  float *B : a pointer to the subtrahend vector
 *                  float *c : a pointer to the vector that contains the
 *                             result of vector (A-B)
 *                  int n : the number of elements in the vector (i.e. the vector length)
 * Return:
 */
void FAR PASCAL _export FVec_Sub_FVec(float _huge *A, float _huge *B, float _huge *C, int n)
{
 int _IX;
 for (_IX = 1; _IX<= n; _IX++)
  *(++C) = *(++A) - *(++B);
}

/*
 * Sca_Add_FVec -- add a scalar S to a floating point vector A[1..n]
 *              and put the result in C[1..n]
 * Syntax:          void Sca_Add_FVec (float S, float *A, float *C, int n)
 * Parameters:      float S : the scalar to be added
 *                  float *A : a pointer to the vector addend
 *                  float *C : a pointer to the vector that contain the
 *                             result (sum) of (S+A).
 *                  int n : the number of elements in the vector (i.e. the vector length)
 */
void FAR PASCAL _export Sca_Add_FVec(float S, float _huge *A, float _huge *C, int n)
{
 int _IX;
 for (_IX=1; _IX<=n; _IX++)
  *(++C) = S + *(++A);
}

/*
 * Max_FVecAB -- compare two floating vector A[1..n] and B[1..n] and put the
 *            maximum values in vector C[1..n].
 * Syntax:          void Max_FVecAB(float *A, float *B, float *C, int n)
 * Parameters:
 *                  float *A  : a pointer to the first vector to be compared
 *                  float *B  : a pointer to the second vector to be compared
 *                  float *c  : a pointer to the vector that contains the result
 *                              of the comparison of vectors A & B
 *                  int n : the number of elements in the vectors (i.e. the vector length)
 */
void FAR PASCAL _export Max_FVecAB(float _huge *A, float _huge *B, float _huge *C, int n)
{
 int _IX;
 for (_IX=1; _IX<=n; _IX++)
  *(++C) = *(++A) > *(++B) ? *A : *B;
}

/*
 * Min_FVecAB -- compare two floating vectors A[1..n] and B[1..n] and put the
 *            minimum values in vector C[1..n].
 * Syntax:          void Min_FVecAB(float *A, float *B, float *C, int n)
 * Parameters:    float *A : a pointer to the first vector to be compared
```

```
 *                  float *B : a pointer to the second vector to be compared
 *                  float *c : a pointer to the vector that contains the result
 *                                  of the comparison of vectors A & B
 *                  int n : the number of elements in the vectors (i.e. the vector length)
 */
void FAR PASCAL _export Min_FVecAB(float _huge *A, float _huge *B, float _huge *C, int n)
{
 int _IX;
 for (_IX =1; _IX<=n; _IX++){                                                    220
  *(++C) = *(++A) < *(++B) ? *A : *B;
 }
}

/*
 * Max_FVec -- find the maximum value in the floating vector A[1..n]
 * Syntax:        int Max_FVec(float *A, int n)
 * Parameters:  float * : a pointer to the vector to be searched
 *                  int n : the number of elements in the vector (i.e. the vector length)
 * Return :     the index to the element with maximum value                    230
 */
int FAR PASCAL _export Max_FVec(float _huge *A, int n)
{
 int _IX, tmp_idx=1;
 float tmp_v=*A;

 for (_IX=2; _IX<=n; _IX++)
  if (*++A>tmp_v) {
   tmp_v = *A;
   tmp_idx = _IX;                                                              240
  }

 return(tmp_idx);
}

/*
 * Min_FVec -- find the minimum value in the floating vector A[1..n]
 * Syntax:      int Min_FVec(float *A, int n)
 * Parameters:  float * : a pointer to the vector to be searched
 *                        int n : the number of elements in the vector (i.e. the vector length)  250
 * Return:    the index to the element with minimum value
 */
int FAR PASCAL _export Min_FVec(float _huge *A, int n)
{
 int _IX, tmp_idx=1;
 float tmp_v=*++A;

 for (_IX=2; _IX<=n; _IX++)
  if (*++A<tmp_v) {
   tmp_v = *A;                                                                 260
   tmp_idx = _IX;
  }

 return(tmp_idx);
}
```

```
/*
 * Max_IVec -- find the maximum value in the integer vector A[1..n]
 * Syntax:       int Max_IVec(int *A, int n)
 * Parameters:          int *  is : pointer to the vector to be searched
 *              int n : the number of elements in the vector (i.e. the vector length)
 * Return:   the index to the element with maximum value
 */
int FAR PASCAL _export Max_IVec(int _huge *A, int n)
{
 int _IX, tmp_idx=1;
 int tmp_v=*A;

 for (_IX=2; _IX<=n; _IX++)
  if (*++A>tmp_v) {
   tmp_v = *A;
   tmp_idx = _IX;
  }

 return(tmp_idx);
}

/*
 * Min_IVec -- find the minimun value in integer vector A[1..n]
 * Syntax:                          int Min_IVec(int *A, int n)
 * Parameters:  int * : a pointer to the vector to be searched
 *              int n : the number of elements in the vector (i.e. the vector length)
 * Return:      the index to the element with minimum value
 */
int FAR PASCAL _export Min_IVec(int _huge *A, int n)
{
 int _IX, tmp_idx=1;
 int tmp_v=*++A;

 for (_IX=2; _IX<=n; _IX++)
  if (*++A<tmp_v) {
   tmp_v = *A;
   tmp_idx = _IX;
  }
 return(tmp_idx);
}

/*
 * FindSmaller -- find the first element in the vector A[1..n]
 *             that is smaller than 'value'.
 * Syntax:          int FindSmaller(float *A, int n, float value)
 * Parameters:     float *  is a pointer to the vector to be searched
 *                 int n is the number of elements in the vector (i.e. the vector length)
 *                 float is the value against which to compare the vector elements
 * Return:         the index to the first element found that is smaller than 'value'
 */
int FAR PASCAL _export FindSmaller(float _huge *A, int n, float value)
{
 int IX;
```

```
 for (IX=1; IX<=n; IX++){
  A++;
  if (*(A)<value) return(IX);
 }
 return(0);
}

/*
 * FindLarger -- find the first element in the vector A[1..n]
 *               that is larger than 'value'.
 * Syntax:           int FindLarger(float *A, int n, float value)
 * Parameters:       float *  is a pointer to the vector to be searched
 *                   int n is the number of elements in the vector (i.e. the vector length)
 *                   float is the value against which to compare the vector elements
 * Return:       the index to the first element found that is larger than 'value'
 */
int FAR PASCAL _export FindLarger(float _huge *A, int n, float value)
{
 int IX;
 for (IX=1; IX<=n; IX++)
  if (*(++A)>value) return(IX);
 return(0);
}

/*
 * fliplr -- flip matrix A in the left/right direction and
 *           store the result in matrix B.
 * Syntax:       void fliplr(MatrixF *A, MatrixF *B)
 * Parameters:       MatrixF *A  is a pointer to the float matrix to be fliped
 *                   MatrixF *B is a pointer to the float matrix that will
 *                   contain the results of the flip operation
 * Return:
 */
void FAR PASCAL _export fliplr(MatrixF _huge *A, MatrixF _huge *B)
{
 int row, column;
 int i, j;
 float _huge *ptsA;
 float _huge *ptsB;

 row = A->row;
 column = A->column;

 for (i=1; i<=row; i++) {
  ptsA = MatElm(A, i, 1);   // macro
   ptsB = MatElm(B, i+1, 1)-1;    //macro
       for (j=1; j<=column; ptsA++, ptsB--, j++) *ptsB = *ptsA;
 }
}

/*
 * average -- average the elements in every column of matrix A ( m x n )
 * Syntax:       void average(MatrixF *A, MatrixF *Abar)
 * Parameters:   MatrixF *A       - a pointer to the matrix to be averaged
```

```
 *                MatrixF *Abar    - a pointer to the vector containing
 *                the results of the averaging of the columns in matrix A
 */
void FAR PASCAL _export average(MatrixF _huge *A, MatrixF _huge *Abar)
{
 register int i;
 float _huge *ptsA;                                                    380
 MatrixF _huge *sumA;
 int rowA, columnA;

 rowA = A->row;
 columnA = A->column;

 sumA = new_MatrixF(1, columnA);
 for (i=1; i<=rowA; i++){
  ptsA = MatElm(A, i, 1)-1;
  FVec_Add_FVec(ptsA, sumA->buffer, sumA->buffer, columnA);            390
 }

 for (i=1; i<=columnA; i++)  *(Abar->buffer+i) = *(sumA->buffer+i)/rowA;

 old_MatrixF(sumA);
 return;

}

/*                                                                     400
 * table -- Table look-up procedure.
 *          looking up nX where  X[1..n] should be ascending)
 * Syntax:       void table(float *X, float *Y, int size, float *nX, float *nY, int nsize)
 * Parameters:   float *X : a pointer to x vector in table
 *               float *Y : a pointer to y vector in table
 *               int len : length of vectors *X
 *               float *nX : a pointer to vector to look up
 *               float *nY : a pointer to vectors which contains the look up results
 *               int nlen : length of vectors
 * Return:                                                             410
 */
void FAR PASCAL _export table(float _huge *X, float _huge *Y, int size, float _huge *nX,
                              float _huge *nY, int nsize)
{
 int i, IDX;

 for (i=1; i<=nsize; i++) {
  IDX = FindLarger(X, size, *(nX+i));
  if (IDX==0) IDX = size;
  *(nY+i) = *(Y+IDX) + (*(nX+i)-*(X+IDX))/(*(X+IDX)-*(X+IDX-1))       420
   * (*(Y+IDX)-*(Y+IDX-1));
 }
 return;
}

/*
 * sort -- sort an array arr[1..n] into ascending numerical order,
```

```
 *                                   by straight insertion method.
 *  Syntax:                          void  sort(int n, float arr[])
 *  Parameters:              int n is the length of the array to be sorted                    430
 *                           float arr[] is the name of the array to be sorted
 */
void FAR PASCAL _export sort(int n, float _huge arr[])
{
 int i, j;
 float a;

 for (j=2; j<=n; j++) {
  a = arr[j];
  i = j-1;                                                                                  440
  while (i>0 && arr[i]>a) {
   arr[i+1] = arr[i];
   i--;
  }
  arr[i+1] = a;
 }
}

/*
 *  std -- calculate the standard deviation in every column                                 450
 *                           of matrix A (m x n) and store the result in the row vector
 *                           stdA(1 x n)
 *  Syntax: void std(MatrixF *A, MatrixF *stdA)
 *  Return:
 */
void FAR PASCAL _export std(MatrixF _huge *A, MatrixF _huge *stdA)
{
 int row, column;
 int i, j;
 float _huge *ptsA;                                                                         460
 float _huge *ptsstdA, _huge *ptstmp, _huge *ptssum2;
 MatrixF _huge *Abar, _huge *sum2. _huge *tmp;

 row = A->row;
 column = A->column;

 Abar = new_MatrixF(1, column);
 sum2 = new_MatrixF(1, column);
 tmp = new_MatrixF(1, column);
                                                                                            470
 average(A, Abar);

 for (i=1; i<=row; i++) {
  ptsA=MatElm(A, row, 1)-1;
  ptssum2 = sum2->buffer;
  ptstmp = tmp->buffer;
  FVec_Sub_FVec(ptsA, Abar->buffer, tmp->buffer, column);
  for (j=1; j<=column; j++){
   ptssum2++;  ptstmp++;
   *(ptssum2)= *(ptssum2) + (*(ptstmp))*(*ptstmp);                                          480
  }
```

```
  }

  ptsstdA = stdA->buffer;
  ptssum2 = sum2->buffer;
  for (j=1; j<=column; j++){
    ptsstdA++; ptssum2++;
    *ptsstdA = sqrt(*ptssum2/(row-1));
  }
  old_MatrixF(Abar);                                                    490
  old_MatrixF(sum2);
  old_MatrixF(tmp);
  return;
}
```

D.3 afmlib.ide

D.3.1 afmlib.def

```
LIBRARY       afmlib
DESCRIPTION   'afm operations dll'
EXETYPE       WINDOWS
STUB          'WINSTUB.EXE'
CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE DISCARDABLE SINGLE
HEAPSIZE      1024
IMPORTS       NEW_MATRIXF = MATRXLIB.NEW_MATRIXF
              OLD_MATRIXF = MATRXLIB.OLD_MATRIXF
              NEW_MATRIXI = MATRXLIB.NEW_MATRIXI                        10
              OLD_MATRIXI = MATRXLIB.OLD_MATRIXI
              MAX_IVEC = MATRXLIB.MAX_IVEC
              MIN_IVEC = MATRXLIB.MIN_IVEC
              TABLE = MATRXLIB.TABLE
              SORT = MATRXLIB.SORT
              MIN_FVECAB = MATRXLIB.MIN_FVECAB
              SCA_ADD_FVEC = MATRXLIB.SCA_ADD_FVEC
              FINDSMALLER = MATRXLIB.FINDSMALLER
```

D.3.2 afmlib.h

```
/*************************************************************
        Header File : afmlib.h
        Description : header files for afmlib.c

        Written by C.J. Chiu
        Dec. 1994
 *************************************************************/
void FAR PASCAL tip_prof(float *tip_data, float XUNIT, int tip_pts);
void FAR PASCAL wiener2( MatrixF _huge *A, MatrixF _huge *C, int nhoodrow, int nhoodcol);
void FAR PASCAL medfilt2(MatrixF _huge *A, MatrixF _huge *AA, int m, int n);        10
void FAR PASCAL rec_edg1( MatrixF _huge *A, MatrixF _huge *C, MatrixI _huge *edgept,
```

```
                        float XUNIT, float threshRatio, float offset, float slope1);
void FAR PASCAL rec_edg2( MatrixF _huge *A, MatrixF _huge *C, MatrixI _huge *edgept,
                        float XUNIT, float threshRatio, float offset );
void FAR PASCAL combine2D(MatrixF _huge *AA, MatrixF _huge *BB,
                        MatrixI _huge *edgeptA, MatrixI _huge *edgeptB,
                        float thetaA, float thetaB, float XUNIT, float NORMGRID,
                        MatrixF _huge **nPx, MatrixF _huge **nPAz, MatrixF _huge **nPBz);
void FAR PASCAL combine3D( MatrixF _huge *AA, MatrixF _huge *BB, MatrixF _huge *nPx,
                        MatrixF _huge *nPAz,                                          20
                        MatrixF _huge *nPBz, MatrixI _huge *edgeptA, MatrixI _huge *edgeptB,
                        float thetaA, float thetaB, MatrixF _huge *XX, MatrixF _huge *YY,
                        MatrixF _huge *ZZA, MatrixF _huge *ZZB, float XUNIT, float YUNIT );
```

D.3.3 afmlib.c

```
/*********************************************************************
  File : afmlib.c
  Description : DLL subroutine for afm profiling related functions

  Written by C.J. Chiu
  March, 1995
  *********************************************************************/
#include <windows.h>
#include <math.h>
#include <stdio.h>                                                              10
#include <stdlib.h>
#include "matrxlib.h"
#include "afmlib.h"

HANDLE ghInstance;

int FAR PASCAL LibMain (HANDLE hInstance, WORD wDataSeg, WORD wHeapSize, LPSTR lpszCmdLine)
{
  HANDLE ghInstance;
  if (wHeapSize > 0)        /* unlock local data segment */                     20
    UnlockData (0);
  ghInstance = hInstance:           /* save instance handle */
  return (1);
}

/*********** dll terminator function ***************/
int FAR PASCAL WEP (int nParameter)
{
  /* any cleanup activities go here */
  return;                                                                       30
}

/*
 * tip_prof : establish the tip profile
 *    XUNIT : the distance of the adjacent points in the X direction
 *    tip_pts : total points
 */
void FAR PASCAL _export tip_prof(float *tip_data, float XUNIT, int tip_pts)
```

```
{
 int i;                                                                    40
 float R=0.020; /* micron */
 float m;

 m=0.1763*3;
 for (i=-(tip_pts-1)/2; i<=(tip_pts-1)/2; i++)
  *(++tip_data) = sqrt(R*R/m/m + i*i*XUNIT*XUNIT)/m - R/m/m;
}

/*
 *       rec_edg1 : use the envelop image method to reconstruct the profile   50
 *                  then use the first derivative based method to find the edge.
 *                          The blade should be on the lefthand side.
 *                          A : input matrix (MxN)
 *                          C : output matrix (MxN)
 *                          edgept : the index vector of the identified edge. (Mx1)
 *                          threshRatio : ending point of searching
 *                          offset : ending point - (offset/XUNIT) = starting point
 *                          slope1 : slope threshold
 */
void FAR PASCAL _export rec_edg1(MatrixF _huge *A, MatrixF _huge *C, MatrixI _huge *edgept,
                                 float XUNIT, float threshRatio, float offset, float slope1)
{
 MatrixF _huge *tip_data=NULL;
 MatrixF _huge *ptsB=NULL;
 int tip_pts, ctpt;
 int j,k,row,column;
 long i;
 float _huge *ptsA, _huge *ptsC;
 int addpts;  /* the points per row to do the deconvolution */
 int startpt;  /* starting point for deconvolution */                       70
 int endpt;       /* ending point for deconvolution */
 int flag;
 float threshold, slope, SLOPE;

 SLOPE = slope1*2*XUNIT;

 row = A->row;
 column = A->column;

 addpts = floor(offset/XUNIT);                                             80
 /* load tip profile */
 tip_pts = ((int)(0.5/XUNIT))*2+1;
 ctpt=(int)(0.5/XUNIT)+1;
 tip_data = new_MatrixF(1,tip_pts);
 ptsB = new_MatrixF(1, tip_pts);
 tip_prof(tip_data->buffer, XUNIT, tip_pts);

 for (i=1L; i<=(long)column*row; i++) *(C->buffer+i) = *(A->buffer+i);

 for (i=1; i<= row; i++) {                                                 90
  ptsA = MatElm(A, i, 1)-1;
  ptsC = MatElm(C, i, 1)-1;
```

```
  /* find the area for reconstruction*/
  threshold = *(ptsA+column) + threshRatio*( *(ptsA+1) - *(ptsA+column) );
  endpt = FindSmaller(ptsA, column, threshold);
  startpt = endpt-addpts > 1 ? endpt-addpts : 1;
  /* envelop image analysis */
  if (startpt<ctpt) {
    for (j=startpt; j<ctpt;j++) {
        Sca_Add_FVec(*(ptsA+j), tip_data->buffer+ctpt-j, ptsB->buffer+ctpt-j, ctpt-1+j);
        Min_FVecAB(ptsC, ptsB->buffer+ctpt-j, ptsC, ctpt-1+j);
    }
    startpt=ctpt;
  }
  if (column-endpt+1<ctpt){
    for (j=column-ctpt+2;j<=endpt;j++){
        Sca_Add_FVec(*(ptsA+j), tip_data->buffer, ptsB->buffer, ctpt+column-j);
        Min_FVecAB(ptsC+j-ctpt, ptsB->buffer, ptsC+j-ctpt, ctpt+column-j);
    }
    endpt=column-ctpt+1;
  }
  for (j=startpt; j<=endpt; j++)  {
    Sca_Add_FVec(*(ptsA+j), tip_data->buffer, ptsB->buffer,  tip_pts);
    Min_FVecAB(ptsC+j-ctpt, ptsB->buffer, ptsC+j-ctpt, tip_pts);
  }

  /* finding edge point */
  flag = 0;
  for (k=startpt; k<=endpt; k++) {
    slope = *(ptsC+k+2) - *(ptsC+k);
    if (slope <= SLOPE ) flag = flag + 1;
    else flag = 0;
    if (flag==3) break;
  }
  *(edgept->buffer+i) = k-2;
 }
 old_MatrixF(tip_data);
 old_MatrixF(ptsB);
}

/*
 *      rec_edg2 : the same as rec_edg1, but use the second derivative based method
 *                 to find the edge.
 */
void FAR PASCAL _export rec_edg2(MatrixF _huge *A, MatrixF _huge *C, MatrixI _huge *edgept,
                                 float XUNIT, float threshRatio, float offset)
{
 MatrixF _huge *tip_data=NULL;
 MatrixF _huge *ptsB=NULL;
 int tip_pts, ctpt;
 int j,k,row,column;
 long i;
 float _huge *ptsA, _huge *ptsC;
 int addpts;  /* the points per row to do the deconvolution */
 int startpt:  /* starting point for deconvolution */
 int endpt;       /* ending point for deconvolution */
```

```
int flag, l, m;
float threshold, slope, SLOPE, SLOPE1, dd, tmp;

SLOPE = -1*3*XUNIT;                                                    150
SLOPE1=-1.2*3*XUNIT;

row = A->row;
column = A->column;

addpts = floor(offset/XUNIT);

tip_pts =((int)(0.5/XUNIT))*2+1;
ctpt = (int)(0.5/XUNIT)+1;
tip_data = new_MatrixF(1,tip_pts);                                     160
ptsB = new_MatrixF(1, tip_pts);
tip_prof(tip_data->buffer, XUNIT, tip_pts);

for (i=1L; i<=(long)column*row; i++) *(C->buffer+i) = *(A->buffer+i);

for (i=1; i<= row; i++) {
  ptsA = MatElm(A, i, 1)-1;
  ptsC = MatElm(C, i, 1)-1;
  threshold = *(ptsA+column) + threshRatio*( *(ptsA+1) - *(ptsA+column) );
  endpt = FindSmaller(ptsA, column, threshold);                        170
  startpt = endpt-addpts > 1 ? endpt-addpts : 1;

  if (startpt<ctpt) {
    for (j=startpt; j<ctpt;j++) {
        Sca_Add_FVec(*(ptsA+j), tip_data->buffer+ctpt-j, ptsB->buffer+ctpt-j, ctpt-1+j);
        Min_FVecAB(ptsC, ptsB->buffer+ctpt-j, ptsC, ctpt-1+j);
    }
    startpt=ctpt;
  }
  if (column-endpt+1<ctpt){                                            180
    for (j=column-ctpt+2;j<=endpt;j++){
        Sca_Add_FVec(*(ptsA+j), tip_data->buffer, ptsB->buffer, ctpt+column-j);
        Min_FVecAB(ptsC+j-ctpt, ptsB->buffer, ptsC+j-ctpt, ctpt+column-j);
    }
    endpt=column-ctpt+1;
  }
  for (j=startpt; j<=endpt; j++)  {
    Sca_Add_FVec(*(ptsA+j), tip_data->buffer, ptsB->buffer,  tip_pts);
    Min_FVecAB(ptsC+j-ctpt, ptsB->buffer, ptsC+j-ctpt, tip_pts);
  }                                                                    190

  /* finding edge point */
  flag = 0; l=0;  dd=0, m=startpt;
  for (k=startpt; k<=endpt; k++) {
    slope = *(ptsC+k+3) - *(ptsC+k);
    if (slope <= SLOPE ) flag = 1;
    if (flag==1) {
        if (slope <SLOPE1 ) break;
        tmp=(*(ptsC+k+10)-*(ptsC+k+6)) - (*(ptsC+k+4)-*(ptsC+k));
        if (tmp<dd) {                                                  200
```

```
            m=startpt+l+5;
            dd=tmp;
          }
      }
      l++;
    }
    *(edgept->buffer+i) = m;
  }
  old_MatrixF(tip_data);
  old_MatrixF(ptsB);                                                    210
}

/*
 * combine2D : combin the 2 sides images.
 *        AA : left side image for combination (MxN)
 *        BB : right side image (MxN)
 *        edgeptA : the edgept index vector for AA (Mx1)
 *        edgeptB : the edgept index vector for BB (Mx1)
 *      thetaA, thetaB : the stage's tilt angles (degrees)
 *        NORMGRID : the interpolation distance (micron)                  220
 *        nPx : output X data (1xK),
 *        nPAz : output Z data of AA (MxK)
 *        nPBz : output Z data of BB (MxK)
 *  Note : the memory and the size of nPx, nPAz, nPBz are assigned in the function.
 */
void FAR PASCAL _export combine2D(MatrixF _huge *AA, MatrixF _huge *BB,
                                  MatrixI _huge *edgeptA, MatrixI _huge *edgeptB,
                                  float thetaA, float thetaB, float XUNIT, float NORMGRID,
                          MatrixF _huge **nPx, MatrixF _huge **nPAz, MatrixF _huge **nPBz)
{                                                                       230
  register int i, j, k;
  int pts, tmpx, rowA, columnA, rowB, columnB, minrowA, maxrowB;
  double sA, cA, sB, cB;
  float mindist, tmpz;
  MatrixF _huge *PAx, _huge *PAz, _huge *PBx, _huge *PBz;
  float delPAx, delPAz, delPBx, delPBz;
  float _huge *ptsA, _huge *ptsB, _huge *ptsnPAz, _huge *ptsnPBz;

  sA = sin(thetaA*PI/180);
  cA = cos(thetaA*PI/180);                                              240
  sB = sin(thetaB*PI/180);
  cB = cos(thetaB*PI/180);

  rowA = AA->row; columnA = AA->column;
  rowB = BB->row; columnB = BB->column;
  /* take the smallest distance of the traces as the final distance for
     displaying 2D profile */
  minrowA = Min_IVec(edgeptA->buffer, edgeptA->row);
  maxrowB = Max_IVec(edgeptB->buffer, edgeptB->row);
  mindist = min( (*(edgeptA->buffer+minrowA)-1),                        250
                 (columnA-*(edgeptB->buffer+maxrowB)) ) *XUNIT;
  pts = floor(mindist/NORMGRID);

  *nPx = new_MatrixF(1, pts);
```

```
  *nPAz = new_MatrixF(rowA, pts);
  *nPBz = new_MatrixF(rowB, pts);

  PAx = new_MatrixF(1, columnA);
  PAz = new_MatrixF(1, columnA);
  PBx = new_MatrixF(1, columnB);                                          260
  PBz = new_MatrixF(1, columnB);

  for (i=1; i<=pts; i++) *((*nPx)->buffer+i) = NORMGRID*(i-1);
  /* coordinate transformation */
  for (i=1; i<=AA->row; i++) {
    ptsA = MatElm(AA, i, 1)-1;
    ptsnPAz = MatElm(*nPAz, i, 1)-1;
    tmpx = *(edgeptA->buffer+i);
    tmpz = *(ptsA+tmpx);
    k=tmpx;                                                               270
    for (j=1; j<=tmpx; k--,j++) {
      delPAx = (-tmpx+j)*XUNIT;
      delPAz = *(ptsA+j) - tmpz;
      *(PAx->buffer+k) = -cA*delPAx - sA*delPAz;
      *(PAz->buffer+k) = sA*delPAx - cA*delPAz;
    }
    /* interpolation to let each trace line have the same spacing in the X direction */
    table( PAx->buffer, PAz->buffer, tmpx,
           (*nPx)->buffer, ptsnPAz, pts );
                                                                          280
    ptsB = MatElm(BB, i, 1)-1;
    ptsnPBz = MatElm(*nPBz, i, 1)-1;
    tmpx = *(edgeptB->buffer+i);
    tmpz = *(ptsB+tmpx);
    for (j=tmpx; j<=columnB; j++) {
      delPBx = (-tmpx+j)*XUNIT;
      delPBz = *(ptsB+j) - tmpz;
      *(PBx->buffer+j) = cB*delPBx - sB*delPBz;
      *(PBz->buffer+j) = sB*delPBx + cB*delPBz;
    }                                                                     290
    table( PBx->buffer+tmpx-1, PBz->buffer+tmpx-1, columnB-tmpx+1,
           (*nPx)->buffer, ptsnPBz, pts);

  }
  /* clear memory */
  old_MatrixF(PAx);
  old_MatrixF(PAz);
  old_MatrixF(PBx);
  old_MatrixF(PBz);
                                                                          300
  return;
}

/*
 *  combine3D : generate the necessary information to display the 3D picture
 *        AA, BB, nPx, nPAz, nPBz, edgeptA, edgeptB : the same as combine2D
 *      thetaA, thetaB : the stage tilt angles
 *        XX : the 3D matrix of the x element
```

```
 *       YY : the 3D matrix of the y element
 *       ZZA : the 3D matrix of the z element (size A)                    310
 *       ZZB : the 3D matrix of the z element (size B)
 */
void FAR PASCAL _export combine3D(MatrixF _huge *AA, MatrixF _huge *BB,
                  MatrixF _huge *nPx, MatrixF _huge *nPAz, MatrixF _huge *nPBz,
                  MatrixI _huge *edgeptA, MatrixI _huge *edgeptB,
                  float thetaA, float thetaB, MatrixF _huge *XX, MatrixF _huge *YY,
                  MatrixF _huge *ZZA, MatrixF _huge *ZZB, float XUNIT, float YUNIT )
{
 float deledgeAx, deledgeAz, deledgeBx, deledgeBz;
 double sA, cA, sB, cB;                                                   320
 float nedgeAx, nedgeAz, nedgeBx, nedgeBz, nedgex, nedgez;
 register int i;
 int edgeA1x, edgeB1x, edgeAnx, edgeBnx, column;
 float edgeA1z, edgeB1z, edgeAnz, edgeBnz, *ptsA, *ptsB;

 sA = sin(thetaA*PI/180);
 cA = cos(thetaA*PI/180);
 sB = sin(thetaB*PI/180);
 cB = cos(thetaB*PI/180);
 column = nPx->column;                                                    330

 edgeA1x = *(edgeptA->buffer+1);
 edgeB1x = *(edgeptB->buffer+1);
 ptsA = MatElm(AA, 1, 1) - 1;
 ptsB = MatElm(BB, 1, 1) - 1;
 edgeA1z =  *(ptsA + edgeA1x);
 edgeB1z =  *(ptsB + edgeB1x);

 for (i=1; i<=nPAz->row; i++){
  ptsA = MatElm(AA, i, 1) - 1;                                            340
  ptsB = MatElm(BB, i, 1) - 1;

  edgeAnx = *(edgeptA->buffer+i);
  edgeBnx = *(edgeptB->buffer+i);
  edgeAnz = *(ptsA + edgeAnx);
  edgeBnz = *(ptsB + edgeBnx);
  deledgeAx = (edgeAnx - edgeA1x)*XUNIT;
  deledgeBx = (edgeBnx - edgeB1x)*XUNIT;
  deledgeAz = edgeAnz - edgeA1z;
  deledgeBz = edgeBnz - edgeB1z;                                          350

  nedgeAx = -cA*deledgeAx - sA*deledgeAz;
  nedgeAz = sA*deledgeAx - cA*deledgeAz;
  nedgeBx = cB*deledgeBx - sB*deledgeBz;
  nedgeBz = sB*deledgeBx + cB*deledgeBz;

  nedgex = (nedgeAx + nedgeBx)/2;
  nedgez = (nedgeAz + nedgeBz)/2;

  Sca_Add_FVec(nedgex, nPx->buffer, MatElm(XX, i, 1)-1, column);          360
  Sca_Add_FVec((i-1)*YUNIT, MatElm(YY, i, 1)-1, MatElm(YY, i, 1)-1, column);
  Sca_Add_FVec(nedgez, nPAz->buffer+(i-1)*nPx->column, MatElm(ZZA, i, 1)-1, column);
```

```
    Sca_Add_FVec(nedgez, nPBz->buffer+(i-1)*nPx->column, MatElm(ZZB, i, 1)-1, column);
  }
}

/*
 * medfilt2 : 2D median filter
 *        A : input data (MxN)
 *        AA : output data (MxN)                                          370
 *        m x n : the size of the window for filtering
 */
void FAR PASCAL _export medfilt2(MatrixF _huge *A, MatrixF _huge *AA, int m, int n)
{
  register int  j, l;
  int row, column, cm, cn, frow, fcol, subtotal, ctblk;
  unsigned long total, i, k;
  float  _huge *block;
  float _huge *tmp;
  float  _huge *ptsA;                                                     380
  static HANDLE hblock, htmp;

  row = A->row;
  column = A->column;
  total = (unsigned long)row * column;
  subtotal = (m * n);
  ctblk = (subtotal/2 + 1);

  hblock = GlobalAlloc(GMEM_MOVEABLE, ((subtotal+1L) * sizeof(float)));
                                                                          390
  htmp = GlobalAlloc(GMEM_MOVEABLE, ((total+1L) * sizeof(float)));

  //block = (float _huge *)malloc( (subtotal+1)*sizeof(float) );
  //tmp = (float _huge *)malloc( (total + 1) * sizeof(float) );

  if((block = (float _huge *)GlobalLock(hblock)) && (tmp = (float _huge *)GlobalLock(htmp))){
    cm = m/2 + 1;
    cn = n/2 + 1;
    frow = row - cm + 1;
    fcol = column - cn + 1;                                               400

    //*(AA->buffer) = 0L;
      for (i=1L; i<=total; i++) *(AA->buffer+i) = *(A->buffer+i);
      //*(AA->buffer + i) = *(A->buffer + i);

          for (i=cm; i<=frow; i++) {
          for (j=cn; j<=fcol; j++) {
          tmp = block;
          for (k=i-cm+1; k<=i+cm-1; k++) {
          ptsA = MatElm(A, k, j-cn+1) - 1;                                410
          for (l=1; l<=n; l++){
          //      tmp = tmp + 1L;
          //ptsA = ptsA + 1L;
          *(++tmp) = *(++ptsA);
          }
          }
```

```
            sort(subtotal, (float _huge *)block);
            ptsA = MatElm(AA, i, j);
            //      block = block + (long)ctblk;
            *ptsA = *(block + (long)ctblk);                                    420
            }
      }
    }
    GlobalUnlock(hblock);
    GlobalUnlock(htmp);
    GlobalFree(htmp);
    GlobalFree(hblock);
}

/* wiener2() : 2D adaptive wiener filter.                                     430
 *       A :  the original MxN matrix before filtering
 *       C :  the filtered MxN matrix
 *       nhoodrow, nhoodcol : the size of neighborhoods used to estimate
 *                            the local image mean and standard deviation
 */
void FAR PASCAL _export wiener2(MatrixF _huge *A, MatrixF _huge *C, int nhoodrow, int nhoodcol)
{
  int  j, k, l;
  int arow, acol, subtotal, padrow, padcol;
  unsigned long total, i;                                                      440
  float _huge *ptsA=NULL, _huge *ptsB=NULL, _huge *ptsC=NULL, _huge *ptsmean=NULL,
  float  _huge *ptsvar=NULL;
  float tmp, minnA, maxnA, scale, del;
  float summean, sumvar, noise;
  MatrixF _huge *B=NULL, _huge *mean=NULL, _huge *var=NULL;

  arow = A->row;
  acol = A->column;
  total = (unsigned long)arow * acol;
  subtotal = nhoodrow * nhoodcol;                                              450

  padrow = (nhoodrow-1)/2;
  padcol = (nhoodcol-1)/2;

  B = new_MatrixF(arow+nhoodrow-1, acol+nhoodcol-1);
  mean = new_MatrixF(arow, acol);
  var = new_MatrixF(arow, acol);

  ptsA = A->buffer + 1L;
  minnA = *ptsA;                                                               460
  maxnA = *ptsA;
  for (i=2L; i<=total; i++) {
    ptsA = ptsA+1L;
    if (*ptsA>maxnA) maxnA = *ptsA;
    else if (*ptsA<minnA) minnA = *ptsA;
  }

  /* intensity matrix */
  scale = (maxnA - minnA);
  ptsA = A->buffer + 1L;                                                       470
```

```
for (i=1+padrow; i<=arow+padrow; i++){
  ptsB = MatElm(B, i, 1+padcol);
  for (j=1+padcol; j<=acol+padcol; j++) {
    *ptsB = (*ptsA - minnA) / scale;
    ++ptsA;
    ++ptsB;
  }
}

ptsmean = mean->buffer+1L;                                          480
ptsvar = var->buffer+1L;
for (i=1+padrow; i<=arow+padrow; i++){
  for (j=1+padcol; j<=acol+padcol; j++) {
    summean = 0;
    sumvar = 0;
    for (k=i-padrow; k<=i+padrow; k++) {
        ptsB = MatElm(B, k, j-padcol);
        for (l=j-padcol; l<=j+padcol; l++) {
          summean += *ptsB;
          sumvar += (*ptsB) * (*ptsB);                              490
          ++ptsB;
        }
    }
    tmp = summean /(float)subtotal;
    *ptsmean = tmp;
    *ptsvar = sumvar/(float)subtotal - tmp * tmp;
    ++ptsmean;
    ++ptsvar;
  }
}                                                                   500

/* estimate noise */
tmp = 0;
ptsvar = var->buffer+1L;
for (i = 1L; i<=total; i++){
  tmp = tmp + *ptsvar;
  ++ptsvar;
}
noise = tmp/(float)total;
                                                                    510
/*  pixel-wise filter the image */
ptsC = C->buffer+1L;
ptsmean = mean->buffer+1L;
ptsvar = var->buffer+1L;
for (i = 1L; i<=arow; i++){
  ptsB = MatElm(B, i+padrow, 1+padcol);
  for (j=1; j<=acol; j++) {
    del = *ptsvar - noise;
    if (del<0) {
        del =0;                                                     520
        *ptsvar = noise;
    }
    *ptsC = ((*ptsmean) +  (del/(*ptsvar)) * (*ptsB - *ptsmean)) * scale + minnA;
    ++ptsB;
```

```
    ++ptsC;
    ++ptsmean; ++ptsvar;

   }
  }
  /* clear memory */                                          530
  old_MatrixF(B);
  old_MatrixF(mean);
  old_MatrixF(var);
}
```

D.4 filesel.ide

D.4.1 filesel2.def

```
LIBRARY       filesel2
DESCRIPTION   'file selection dll'
EXETYPE       WINDOWS
STUB          'WINSTUB.EXE'
CODE          PRELOAD MOVEABLE DISCARDABLE
DATA          PRELOAD MOVEABLE DISCARDABLE SINGLE
HEAPSIZE      1024
```

D.4.2 filesel2.h

```
/* filesel2.h header file for filesel.c */

#define DLI_EDIT                    101
#define DLI_DIRECT                  102
#define DLI_OK                      103
#define DLI_CANCEL                  104
#define DLI_DIRLIST                 105
#define DLI_FILELIST          106

#define FILENAMELONG          13                                10
#define PATHNAMELONG          128
#define FILEOPENCAPTIONLONG 32

typedef struct tagFILEOPENDATA
{
        char            szPath [PATHNAMELONG];
        char      szFile [FILENAMELONG];
        char            szCaption [FILEOPENCAPTIONLONG];
}FILEOPENDATA, FILESAVEASDATA;
                                                                  20

BOOL   FAR PASCAL CallFileDlg (HWND hWnd, HANDLE hFileOpenData);
BOOL   FAR PASCAL FileOpenProc (HWND hDlg, WORD wMessage, WORD wParam,
       LONG lParam);
BOOL   FAR PASCAL CallFileSaveAsDlg (HWND hWnd, HANDLE hFileSaveAsData);
BOOL   FAR PASCAL FileSaveAsProc (HWND hDlg, WORD wMessage, WORD wParam,
```

```
        LONG lParam);
void    FAR PASCAL CombinePathFile (LPSTR szFileName, LPSTR szDir, LPSTR szFile);
void    FAR PASCAL SplitPathFile (LPSTR lpSource, LPSTR lpPath, LPSTR lpFile);
```

D.4.3 filesel2.c

```
#include <windows.h>
#include <string.h>      /* needed for strtok() function */
#include <direct.h>     /* needed for getcwd() function */
#include "filesel2.h"


HANDLE ghInstance;      /* static data /*

                              /* dll initiator function */
 int FAR PASCAL LibMain (HANDLE hInstance, WORD wDataSeg,
                         WORD wHeapSize, LPSTR lpszCmdLine)
{
 if (wHeapSize > 0)       /* unlock local data segment */
   UnlockData (0);
 ghInstance = hInstance;          /* save instance handle */
 return (1);
}

/*
 * CallFileDlg() -- This function calls the dialog box procedure, passing the
 *                  names of the starting subdirectory, search path (like "*.*"),
 *                  and the dialog box caption.
 *  Prototype:     BOOL FAR PASCAL _export CallFileDlg (HWND, HANDLE)
 *  Parameters:    HWND hWnd, represents the handle of window of the calling program.
 *                 HANDLE hFileOpenData, represents the handle of the
 *  Return:        A pointer to a FILEOPENDATA structure containing the selected path and file names.
 *
 * Note:           The calling program should check the bNotCancel variable in the
 *                 FILEOPENDATA to make sure that it is TRUE before proceeding.
 *                 A FALSE value implies that the user did not select a file.
 */
BOOL FAR PASCAL _export CallFileDlg (HWND hWnd, HANDLE hFileOpenData)
{
 FARPROC                    lpfnDlgProc;
 BOOL                   bRetStatus;
 FILEOPENDATA FAR *  lpFOD;

 lpFOD = (FILEOPENDATA FAR *) GlobalLock (hFileOpenData);
// lock the mem for FILEOPENDATA struct
   if (lpFOD)
    {
        lpfnDlgProc = MakeProcInstance (FileOpenProc, ghInstance);
        /* call dialog box function*/
        bRetStatus = DialogBoxParam (ghInstance, "FileOpenDlg",
```

```
                                    hWnd, lpfnDlgProc, (DWORD) (LPSTR) lpFOD);
        FreeProcInstance ( lpfnDlgProc);
        GlobalUnlock (hFileOpenData);
        return (bRetStatus);
    }
 return (FALSE);                                                          50
}


/*
 *  FileOpenProc() -  This function runs the FileDlg dialog box,
 *                    defined in filesel.rc. It should be called using the
 *                    DialogBoxParam() function, passing a pointer to a
 *                    FILEOPENDATA structure containing the starting
 *                    caption, path name and file search strings
 *                    (like "c://", and "*.*").                           60
 *  Call:             FileOpenProc (hDlg, wMessage, wParam, lParam)
 *  Parameters:       HWND hDlg - represents the handle of the dialog box
 *                    WORD wMessage -
 *                    WORD wParam   -
 *                    LONG lParam   -
 *  Return:           TRUE if a file was selected,  FALSE if not
 *  Note:           No attempt is made to open the file within this procedure.
 */
BOOL FAR PASCAL _export FileOpenProc (HWND hDlg, WORD wMessage, WORD wParam, LONG
                                    lParam)                              70
{

 static FILEOPENDATA  FAR * lpFOD;
 static char FAR          szPathName[PATHNAMELONG], szFileName [FILENAMELONG].
 static char FAR      szTmpFileName [FILENAMELONG];
 static char FAR          szPathFile [PATHNAMELONG];    /* working buffer */
 static BOOL              bMadeSel;
 int nCurSel, nFileIndx;
 int nListBoxCount;
                                                                         80
 int InStr (LPSTR lpszString, LPSTR lpszCheck);

 switch (wMessage) {
 case WM_INITDIALOG:
  lpFOD = (FILEOPENDATA FAR *) lParam;
  SetWindowText (hDlg, lpFOD->szCaption);       /* pointer to data */
  /* initialize working buffer */
  CombinePathFile (szPathFile, lpFOD->szPath, lpFOD->szFile);
  /* fill both list boxes */
  DlgDirList (hDlg, szPathFile, DLI_FILELIST, DLI_EDIT, 0);              90
  DlgDirList (hDlg, szPathFile, DLI_DIRLIST, DLI_DIRECT, 0xC010);
  SetDlgItemText (hDlg, DLI_EDIT, lpFOD->szFile);
  // test code
    SendDlgItemMessage(hDlg, DLI_FILELIST, LB_SETCURSEL, 0, 0L);
  DlgDirSelect (hDlg, szPathFile, DLI_FILELIST);
  SplitPathFile (szPathFile, szPathName, szFileName);  /* display selected file */
  SendDlgItemMessage(hDlg, DLI_FILELIST, LB_GETTEXT, 0, szFileName);
  SetDlgItemText(hDlg, DLI_EDIT, szFileName);
```

```
  bMadeSel = TRUE;  /* keep track of if file was selected */
  return (FALSE);                                                    100
 case WM_COMMAND:
  switch (wParam){
  case DLI_OK:                    /* OK button */
    GetDlgItemText (hDlg, DLI_EDIT, lpFOD->szFile,FILENAMELONG);
    GetDlgItemText (hDlg, DLI_DIRECT, lpFOD->szPath, PATHNAMELONG);
    EndDialog (hDlg, bMadeSel);
    break;
  case DLI_CANCEL:    /* CANCEL button */
    EndDialog (hDlg, FALSE);
    break;                                                           110
  case DLI_EDIT:         /* the EDIT control */
    if (HIWORD (lParam) == EN_CHANGE){
        nListBoxCount = SendDlgItemMessage(hDlg, DLI_FILELIST, LB_GETCOUNT, 0, 0L);
        if(nListBoxCount > 1 && nListBoxCount != LB_ERR){
          GetDlgItemText (hDlg, DLI_EDIT, szTmpFileName,FILENAMELONG);
          nFileIndx = SendDlgItemMessage(hDlg, DLI_FILELIST, LB_FINDSTRING, -1,
                                         szTmpFileName);
          if(nFileIndx != LB_ERR && szTmpFileName != szFileName){
            SendDlgItemMessage(hDlg, DLI_FILELIST, LB_SETCURSEL, nFileIndx, 0L);
            SendDlgItemMessage(hDlg, DLI_DIRECT, LB_SETCURSEL, nFileIndx, 0L);   120
            GetDlgItemText(hDlg, DLI_FILELIST, szFileName, FILENAMELONG);
            SetDlgItemText(hDlg, DLI_EDIT, szFileName);
          }
        }
        bMadeSel = TRUE;    /* user entered file */
        /*GetNextDlgTabItem(hDlg, hCtl, FALSE  */
    }
    break;
  case DLI_FILELIST:    /* the File LISTBOX */
    switch (HIWORD (lParam){                                         130
    case LBN_SELCHANGE: /* picked a file */
        /* get user's selection */
        DlgDirSelect (hDlg, szPathFile, DLI_FILELIST);
        SplitPathFile (szPathFile, szPathName, szFileName);   /* display selected file */
        SetDlgItemText (hDlg, DLI_EDIT, szFileName);
        bMadeSel = TRUE;
        break;
    case LBN_DBLCLK: /* double-clicked file */
        GetDlgItemText (hDlg, DLI_EDIT, lpFOD->szFile,
                        FILENAMELONG);                               140
        GetDlgItemText (hDlg, DLI_DIRECT, lpFOD->szPath, PATHNAMELONG);
        EndDialog (hDlg, bMadeSel);
        break;
    }
    break;
  case DLI_DIRLIST:                /* directory list box */
    switch (HIWORD (lParam)){
      case LBN_SELCHANGE:   /* picked a dir name */
        /* get user's selection */
          DlgDirSelect (hDlg, szPathFile, DLI_DIRLIST);              150
          /* refresh dir list */
          DlgDirList (hDlg, szPathFile, DLI_DIRLIST, DLI_DIRECT, 0Xc010);
```

```
        /* if directory name is just \..*/
        if ((lnStr (szPathFile, "..") != -1) || lstrlen (szPathFile) == 0)
        lstrcpy (szPathFile, "*.*");
        /* refresh file list */
        DlgDirList (hDlg, szPathFile, DLI_FILELIST, DLI_EDIT, 0);
        SplitPathFile (szPathFile, szPathName, szFileName);
        /* show new file spec */
        SetDlgItemText (hDlg, DLI_EDIT, szFileName);                    160
        bMadeSel = FALSE;  /* reset file selection */
        break;
     }
        break;
    }
    return (TRUE);
  }
  return (FALSE);
}
                                                                        170
/*
 * CallFileSaveAsDlg() -   This function calls the dialog box procedure, passing
 *                           the names of the starting subdirectory, search
 *                         path (like "*.*"), and the dialog box caption.
 * Prototype:  BOOL FAR PASCAL _export CallFileSaveAsDlg (HWND, HANDLE)
 * Parameters: HWND hWnd, represents the handle of window of the calling program.
 *               HANDLE hFileSaveAsData, represents the handle of the
 * Return: A pointer to a FILEOPENDATA structure containing the selected path
 *       and file names.
 * Note:        The calling program should check the bNotCancel variable in the   180
 *              FILEOPENDATA to make sure that it is TRUE before proceeding.
 *              A FALSE value implies that the user did not select a file.
 */
BOOL FAR PASCAL _export CallFileSaveAsDlg (HWND hWnd, HANDLE hFileSaveAsData)
  {
    FARPROC                 lpfnDlgProc;
    BOOL                    bRetStatus;
    FILESAVEASDATA FAR *    lpFSAD;

    lpFSAD = (FILESAVEASDATA FAR *) GlobalLock (hFileSaveAsData);       190
    if (lpFSAD){
        lpfnDlgProc = MakeProcInstance (FileSaveAsProc, ghInstance);
        /* call dialog box function*/
        bRetStatus = DialogBoxParam (ghInstance, "FileSaveAsDlg",
                                     hWnd, lpfnDlgProc, (DWORD) (LPSTR) lpFSAD);
        FreeProcInstance ( lpfnDlgProc);
        GlobalUnlock (hFileSaveAsData);
        return (bRetStatus);
    }
    return (FALSE);                                                     200
  }
/*
 * FileSaveAsProc() - This function runs the FileDlg dialog box.
 *                      defined in filesel.rc. It should be called using the
 *                      DialogBoxParam() function. passing a pointer to a
 *                      FILEOPENDATA structure containing the starting
```

```
 *            caption, path name and file search strings (like "c://", and "*.*").
 * Prototype:    BOOL FAR PASCAL _export FileSaveAsProc (HWND, WORD, WORD, LONG)
 * Parameters:          HWND hDlg    - represents the handle of the dialog box
 *                      WORD wMessage -                                          210
 *                    WORD wParam   -
 *                    LONG lParam   -
 * Return:  TRUE if a file was selected,  FALSE if not
 * Note: A message asking the user to confirm the overwrite of an existing
 *            file is issued (the user may continue or cancel).
 */
BOOL FAR PASCAL _export FileSaveAsProc (HWND hDlg, WORD wMessage, WORD wParam, LONG
                                        lParam)
{
 static FILESAVEASDATA      FAR * lpFSAD ;                                       220
 static char FAR            szPathName[PATHNAMELONG], szFileName [FILENAMELONG];
 static char FAR            szPathFile [PATHNAMELONG];    /* working buffer */
 static BOOL                bMadeSel;

 int InStr (LPSTR lpszString, LPSTR lpszCheck);

 switch (wMessage){
  case WM_INITDIALOG:
   lpFSAD = (FILESAVEASDATA FAR *) lParam;
   SetWindowText (hDlg, lpFSAD->szCaption);    /* pointer to data */           230
   /* initialize working buffer */
   CombinePathFile (szPathFile, lpFSAD->szPath, lpFSAD->szFile);
   /* fill both list boxes */
   DlgDirList (hDlg, szPathFile, DLI_FILELIST, DLI_EDIT, 0);
   DlgDirList (hDlg, szPathFile, DLI_DIRLIST, DLI_DIRECT, 0xC010);
   SetDlgItemText (hDlg, DLI_EDIT, lpFSAD->szFile);
        bMadeSel = FALSE;      /* keep track of if file was selected */
   return (FALSE);
  case WM_COMMAND:
   switch (wParam){                                                            240
   case DLI_OK:                /* OK button */
        GetDlgItemText (hDlg, DLI_EDIT, lpFSAD->szFile, FILENAMELONG);
        GetDlgItemText (hDlg, DLI_DIRECT, lpFSAD->szPath, PATHNAMELONG);
        EndDialog (hDlg, bMadeSel);
        break;
   case DLI_CANCEL:  /* CANCEL button */
        EndDialog (hDlg, FALSE);
        break;
   case DLI_EDIT:         /* the EDIT control */
        if (HIWORD (lParam) == EN_CHANGE)                                      250
          bMadeSel = TRUE;   /* user entered file */
        break;
   case DLI_FILELIST:          /* the File LISTBOX */
        switch (HIWORD (lParam)){
        case LBN_SELCHANGE: /* picked a file */
          /* get user's selection */
          DlgDirSelect (hDlg, szPathFile, DLI_FILELIST);
    SplitPathFile (szPathFile, szPathName, szFileName); /* display selected file */
          SetDlgItemText (hDlg, DLI_EDIT, szFileName);
          bMadeSel = TRUE;                                                     260
```

```
            break;
        case LBN_DBLCLK: /* double-clicked file */
            GetDlgItemText (hDlg, DLI_EDIT, lpFSAD->szFile,FILENAMELONG);
            GetDlgItemText (hDlg, DLI_DIRECT, lpFSAD->szPath,PATHNAMELONG);
            EndDialog (hDlg, bMadeSel);
            break;
          }
          break;
        case DLI_DIRLIST:              /* directory list box */
          switch (HIWORD (lParam)){                                          270
          case LBN_SELCHANGE:  /* picked a dir name */
            /* get user's selection */
            DlgDirSelect (hDlg, szPathFile, DLI_DIRLIST);
            /* refresh dir list */
            DlgDirList (hDlg, szPathFile, DLI_DIRLIST,
                        DLI_DIRECT, 0Xc010);
            /* if directory name is just \.. */
            if ((InStr (szPathFile, "..") != -1) || lstrlen (szPathFile) == 0)
              lstrcpy (szPathFile, "*.*");
            /* refresh file list */                                          280
            DlgDirList (hDlg, szPathFile, DLI_FILELIST,DLI_EDIT, 0);
            SplitPathFile (szPathFile, szPathName,szFileName);
            /* show new file spec */
            SetDlgItemText (hDlg, DLI_EDIT, szFileName);
            bMadeSel = FALSE;  /* reset file selection */
            break;
          }
          break;
      }
      return (TRUE);                                                         290
    }
  return (FALSE);
}
/*
 * SplitPathFile() -       This function separates the directory and file name in
 *                  lpSource into two strings
 * Prototype:              void FAR PASCAL _export SplitPathFile (LPSTR,LPSTR,LPSTR)
 * Parameters:             LPSTR lpSource  - pointer to the combined directory/path
 *                         LPSTR lpPath    - pointer to output string for
 *                         LPSTR lpFile    - pointer to output string for      300
 */
void FAR PASCAL _export SplitPathFile (LPSTR lpSource, LPSTR lpPath,  LPSTR lpFile)
{
  LPSTR lpChar;

  lpChar = lpSource + (long) lstrlen (lpSource); /* point to end */
  /* move backwards through string looking for :  or \ chars */
  while (*lpChar != ':' && *lpChar != '\\' && lpChar > lpSource)
    lpChar--;
                                                                             310
  if (*lpChar != ':' && *lpChar != '\\')  /* none found? */
    {
      lstrcpy (lpFile, lpChar);  /* copy file name from end */
      *lpPath = 0;               /* did not find subdirectory */
```

```
    return;
   }
  else /* found : or \, so can copy file and path names */
   {
    lstrcpy (lpFile, lpChar + 1);   /* copy file name from end */
    *(lpChar + 1) = 0;              /* end path before file starts */
    lstrcpy (lpPath, lpSource);
   }
}
/*
 * CombinePathFile() - This function combine the szDir and szFile
 *                     strings to make a complete dir/file name
 * Prototype:          void FAR PASCAL _export CombinePathFile (LPSTR,LPSTR,LPSTR)
 * Parameters:         LPSTR lpSource  - pointer to the combined directory/path
 *                     LPSTR szDir     - pointer to path name string
 *                     LPSTR lpFile    - pointer to file name string
 */
void FAR PASCAL _export CombinePathFile (LPSTR szFileName, LPSTR szDir, LPSTR szFile)
{
 lstrcpy (szFileName, szDir);
 if (szFileName [lstrlen (szFileName)- 1] != '\\')
  lstrcat (szFileName, "\\");
 lstrcat (szFileName, szFile);
}
/* see if lpszCheck is in lpszString, return match pos, else ret -1 */
/* note: this is an internal function to the dll - not exported */
int InStr (LPSTR lpszString, LPSTR lpszCheck)
{
 LPSTR          lpCheck, lpString;
 int            nMatch, nPos;

 nPos = 0;
 do{
  lpCheck = lpszCheck;
  lpString = lpszString;
  nMatch = 0;
  do {
   if (*lpCheck == *lpString)
       nMatch++;
   else
       break;
  }while (*lpCheck++ && *lpString++);
  if(nMatch == lstrlen(lpszCheck))
   return(nPos);
  else
   nPos++;
 }while (*lpszString++);
 return (-1);
}
```

Bibliography

[1] G. Binnig and D. Smith, Rev. Sci. Instrum. Vol. 57, p.1688, 1986

[2] J. Jahanmir, B. G. Haggar, and J. B. Hayes "The Scanning Probe Microscope", Journal of Scanning Microscopy, Vol. 6, No.3, 1992, p.625-660

[3] Rafael C. Gonzalez and Richard E. Woods "Digital Image Processing", Addison Wesley

[4] Joseph E. Griffith and David A. Grigg "Dimensional Metrology with Scanning Probe Microscopes", J. Appl. Phys. Vol. 74(9), Nov 1993.

[5] J. David Keller and Fransiska S. Franke "Envelop Reconstruction of Probe Microscope Images", Journal of Surface Science, Vol.41(294) p.409-419, May 1993.

[6] J. David Keller "Reconstruction of STM and AFM images distorted by finite-size tips", Surf. Sci. Vol. 253, p.353-364

[7] Tarzen Kwok "Design and Implementation of a High Precision Profilometer", Master of Science Thesis, M.I.T, Department of Mechanical Engineering, Feb 1995.

[8] Lee, J. S. "Digital Image Enhancement and Noise Filterinig by Use of Local Statistics", IEEE Trans. Patt. Ana. Mach. Int., Vol. PAMI-2, March 1980, p.165-168

[9] K.L. Lee, D.W. Abraham, F. Secord, and L. Landstein "Submicron Si trench profiling with an electron-beam fabricated atomic force microscope tip", J. Vac. Sci. Technol. B, Vol. 9, p.3562-3568.

[10] Jae S. Lim "Two-Dimensional Signal and Image Processing", Prentice-Hall, 1990.

[11] P. Niedermann and Fischer "Imaging of granular high-Tc thin films using a scanning tunneling microscope with large scan range.", J. Microsc. Vol. 152, p.93-101

[12] William H. Press, Saul A. Teukolskyk, William T. Vetterling, and Brian P. Flannery "Numerical Recipes in C", Cambridge University Press, 1992

[13] "How to Buy an SPM", Park Scientific Instruments.

[14] Tamar Peli and David Malah "A Study of Edge Detection Algorithms", Computer Graphics and Image Processing. Vol. 20, 1982, p.1-21

[15] G. Reiss, F. Schneider, J. Vancea, and H. Hoffmann "Scanning tunneling microscopy on rough surfaces: deconvolution of constant current images.", Appl. Phys. Lett. 57, 867-869

[16] Schwarz U. D., Haefke H., Reimann P., and Guntherodt H.-J. "Tip Artefacts in Scanning Force Microscopy", J. Microsc. 173, 183-197

[17] Alexander H. Slocum "Precision Machine Design", Prentice-Hall, 1992

[18] Kamal Youcef-Toumi, Tarzen Kwok, and Cheng-Jung Chiu "Laboratory for Manufacturing and Productivity Report", M.I.T., May 1995