AN AUTOMATED IMAGING DATA ACQUISITION AND ANALYSIS SYSTEM FOR THE SCANNING LASER ACOUSTIC MICROSCOPE

BY

DAVID D. NICOZISIN

B.S., University of Illinois, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1989

Urbana, Illinois

DEDICATION

This thesis is dedicated to my parents, Rev. Fr. George and Sylvia Nicozisin, who taught me long ago that through faith and perseverance, I could succeed at whatever I did. Thank you for your love and support. God bless you both.

ACKNOWLEDGMENTS

I would like to thank my adviser, Professor William D. O'Brien, Jr., for all of his encouragement and assistance during this project, but especially for his undying optimism during some of the more lean months.

In addition, I would like to thank Diane Agemura and Felice Chu for their help in testing the new system as well as providing suggestions along the way. I would also like to acknowledge Ilmar Hein, Wanda Elliot, Nadine Smith, Rodney Mickelson, and the other people who provided assistance and advice. Thanks everybody.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

It is a well-known fact that computers are an integral part of today's scientific research. Whether they are used for acquiring data, processing data, or running simulations, computers have given scientists the ability to perform tasks that were never before possible or at least prohibitively time-consuming. As time has gone on, those tasks have become more involved and more complicated; and, consequently, scientists have desired enhanced performance from computer systems. Computer designers have answered those desires with supercomputers or specialized data-processing hardware for specific applications. However, the computer designers realized that what they were doing, as with all engineering disciplines, was a practice of trade-offs--in this case, balancing enhanced speed and performance with current technology and cost. It is unfortunate that the concept of trade-offs has become so cliché-ridden, because no truer concept exists in engineering.

Something similar to this occurred at the Bioacoustic Research Laboratory at the University of Illinois, Urbana, Illinois, although perhaps on a smaller scale. Located in the Laboratory is a Scanning Laser Acoustic Microscope (SLAM), a 100 MHz ultrasonic microscope. The SLAM has been used in ongoing research in the areas of biology and metallurgy. The SLAM can be used to measure such things as the attenuation of sound and the speed of sound in

a specimen; and through image and data analyses, the SLAM operator can, for example, characterize biological tissue specimens such as the content of a particular chemical or, in the case of metallurgy, characterize metal fatigue. The purpose of this thesis, however, is not to discuss in depth any of the studies involving the SLAM, but to describe the mechanisms that go into the data analysis on which those studies rely.

Through initial studies in the use of ultrasound to characterize specimens, it was discovered that much data processing went into performing that task. However, the data processing can be very time-consuming, and often time is a constraint for certain specimens. For example, certain fresh tissue samples must be analyzed quickly to obtain desired results. Also, to characterize certain stress fractures in metal correctly, the metal must be examined shortly after the fracture is induced. Consequently, a data acquisition and analysis system using a computer was designed and built using the currently available resources and technology.

However, it was discovered later that the system did not meet all requirements and could be enhanced with new resources; and a new system was designed and built. This was the most recent system, but it had its drawbacks. A faster, easier-to-use system that would allow the operator to concentrate more on specimen preparation and interpreting results and be less concerned with the actual data processing was desired. However, it was desired that the user should be able to interact with the system more easily should it become necessary. In an effort to enhance performance

yet again and produce an automated, "user friendly" system, a new imaging data acquisition and data analysis system was proposed. This is the subject of this thesis.

CHAPTER 2

THEORY AND BACKGROUND INFORMATION

2.1. Scanning Laser Acoustic Microscope

The scanning laser acoustic microscope (SLAM) (Sonomicroscope 100, Sonoscan Inc., Bensenville, Illinois) is the signal-generating basis for which the new imaging data acquisition and analysis system was specifically designed. The SLAM is a 100 MHz ultrasonic microscope used in examining various types of specimens. A detailed description of SLAM operation is available from the references [1-6]; however, a summary of its operation and functions will be presented here.

A block diagram of the SLAM is shown in Figure 1. The specimen is placed on the microscope stage and covered with a special coverslip. The coverslip is a piece of clear 1/4 inch plastic with a gold film on one side that is thin enough to make the coverslip semireflective. The gold-film side is placed on the specimen. A piezoelectric transducer located in the stage is driven with a 100 MHz signal, and sound is produced that propagates up through the stage. The stage itself can either be of fused silica or have a trough to hold a small volume of water. In either case, the fused-silica or water acts as a low-loss coupling medium between the transducer and the specimen. The water stage gives a more uniform sound field but is somewhat more fragile. Once the sound propagates through the coupling medium, it propagates through the specimen region and is attenuated and refracted to some degree,

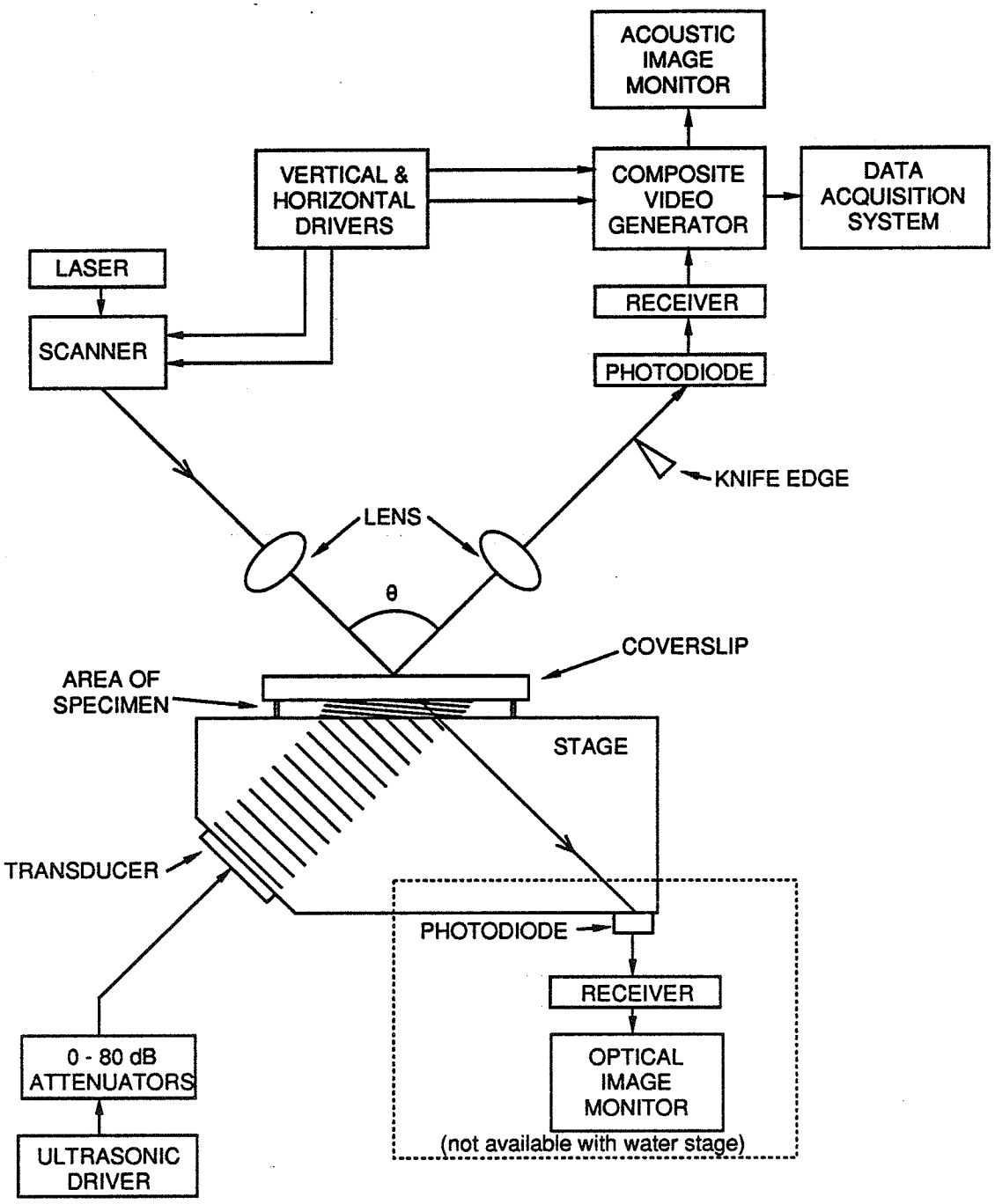

Figure 1. SLAM block diagram.

depending on the specimen. Upon reaching the interface between the specimen and coverslip, the sound produces a distortion, or dynamic ripple, in the gold-film surface which oscillates at the acoustic frequency and has an amplitude proportional to the acoustic pressure reaching the interface. From above, a focused laser beam, scanning in synchronization with a video signal generator, covers an area of approximately 3 mm horizontally by 2 mm vertically. The angle of reflected light is dependent on the localized amplitude of the distortion. A photodiode is used to convert the reflected laser light into a usable electrical signal. Placed in the path of the reflected light is a knife edge which blocks a portion of the light, dependent on the reflected angle. Consequently, the resulting signal from the photodiode is proportional to the localized acoustic pressure variations due to the specimen inhomogeneity. The photodiode signal is sent through a receiver to the aforementioned video signal generator where it is converted into a two-dimensional image and displayed on a standard video monitor.

The SLAM can display three different types of images. The first is the normal acoustic amplitude image generated by the photodiode. It represents acoustic pressure amplitude and hence can be used to assess the attenuation of sound in a specimen. On the display, bright areas correspond to lower attenuation and darker areas correspond to higher attenuation. However, due to some system nonlinearities, the overall acoustic illumination is

not uniform; but this problem can be overcome and will be dealt with in a later section.

The importance of the acoustic image is that it can be used to calculate the ultrasonic attenuation coefficient. The attenuation coefficient represents the normalized decrease in ultrasonic energy after the ultrasound is passed through a specimen. This is one of the means by which a specimen can be characterized, since certain changes in the make-up of a specimen will change its attenuation coefficient.

The second type of image that the SLAM can generate is an acoustic interference image. This image is generated by electronically mixing the output of the photodiode with a 100 MHz reference signal. Doing this produces an image consisting of approximately 39 light and 39 dark alternating vertical bands (fringes). The bands represent equal phase contours of the ultrasonic wave after it passes through the specimen. If a homogeneous medium such as a saline solution (normally used with tissue specimens as a coupling medium) is placed between the stage and coverslip, then the vertical interference lines should be straight and equally spaced. If a specimen is placed in the saline solution, then the interference lines will shift at the interface between specimen and saline. The importance of this "fringe shift" is that the magnitude and direction of the shift are dependent on the localized speed of sound in the specimen relative to the speed of sound in the saline solution (or whatever is used as reference). The interference lines will shift to the right if the speed of

sound in the specimen is greater than that of the reference, and to the left if the speed of sound is less than that of the reference.

The speed of sound is another important way to characterize a specimen, and by placing an unknown sample with a known reference, the speed of sound in the unknown can be determined. Since the phase shift at any location within the image represents the localized change of speed, the speed data can vary significantly depending on the specimen. Obtaining many speed values for a given specimen and performing statistical analysis can yield the heterogeneity index (HI) which gives an indication of the acoustic heterogeneity of a specimen and is useful in characterizing the specimen.

The third type of image is the optical image. It represents an image like that of a light microscope. It is produced by a separate photodiode located within the stage. Since the coverslip is semireflective, some laser light can pass through the specimen and reach the photodiode. Only the fused-silica stage has this photodiode, so an optical image can only be generated while using that stage. The optical image is useful for positioning a specimen correctly within the viewing area. However, an experienced operator can usually position the specimen correctly using the other two types of images as well.

2.2. Available Signals for Analysis

Since this is a data acquisition as well as a data analysis system, some sort of signal is required as an input to be

digitized. The SLAM has many electrical signals that allow it to operate; however, only a few contain information useful to the operator. The first signal is the output of the receiver which converts the signal from the photodiode in the path of the laser light reflected off gold film. This can be seen in Figure 1. (This should not be confused with the photodiode located inside the fused-silica stage which is used for generating optical images.) This signal relates to the received light energy which relates to the received acoustic energy at the specimen coverslip interface. The receiver output is the basic, unmodified, one-dimensional signal before being converted to a video signal. The laser first scans from left to right, and the time frame during which the signal returns to the left is known as the backtrace. During this time, no acoustic energy is received, and the receiver output represents maximum possible attenuation (zero energy). This is important because that portion of the signal can be used as a reference in calculating attenuation. The receiver output can be used in conjunction with synchronization signals that the SLAM also generates.

The receiver output is used to generate a video signal that drives the video monitors. This is the second useful signal generated by the SLAM. To understand how this signal is generated, one must first understand what a video signal is. More detailed information is available [7,8], but a brief description is given here.

The SLAM video output is known as a black and white composite video signal. An example of a composite video signal is shown in Figure 2. It meets the requirements of such a signal as specified by the Federal Communications Commission. Basic black and white, two-dimensional television is reconstructed from a one-dimensional signal because it represents the motion of a modulated bright spot traversing the viewing area from left to right and from top to bottom in a series of straight parallel lines known as rasters. For most of North and South America and including the United States, the specification calls for 525 raster lines to make up one full video frame.

To give the impression of continuity of motion, the frame must be updated. Because of a biological phenomenon known as persistence of vision, the human eye is unaware of discontinuity of motion at frame frequencies greater than 15 Hz. However, the eye can detect flicker, the changing light intensity as frames are switched, at frequencies higher than that depending upon the frequency and brightness of the source. It was found that frame

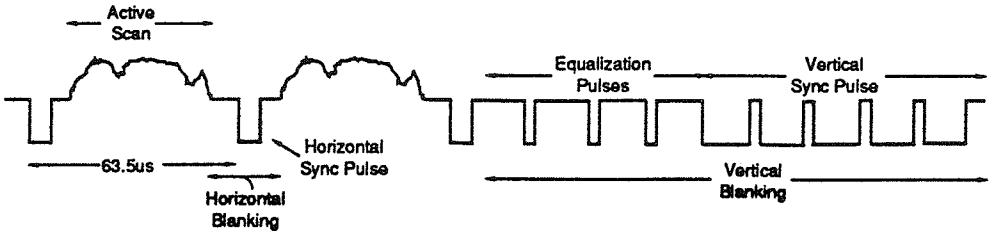

Figure 2. Example of composite video signal (not to scale).

frequencies of 50 Hz or greater were sufficient to overcome this problem. However, updating 525 lines at that rate requires more bandwidth; and since television was meant to be modulated and broadcast over communication channels, this was unacceptable. The solution to this problem was to divide a complete video frame into two fields: the odd raster lines and the even raster lines. This is called interlaced scanning, where every other line is scanned on a given pass. Interlaced scanning is shown in Figure 3. Since there are an odd number of lines and because the raster lines actually have a slight downward slope, each field is made up of 262.5 lines, where only half of line 525 is covered during the odd field, and the other half is covered during the beginning of the even field. Under the current standard, each field is updated at a frequency of 60 Hz (for reasons having to do with the standard commercial power being 60 Hz). This gives an effective full frame

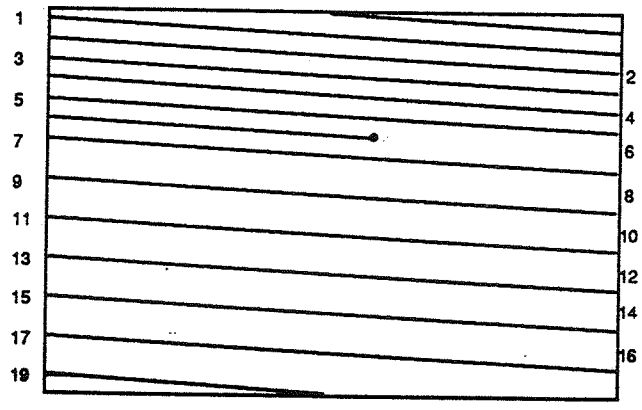

Figure 3. Interlaced scanning.

frequency of 30 Hz, which is fast enough to preserve continuity, while updating every other line at 60 Hz provides enough average brightness to eliminate flicker.

The scanning is done by controlling two drivers, that on a television monitor, control the direction of an electron beam before it reaches the screen. There is a horizontal driver and a vertical driver, both of which generate ramp functions that cause the beam to traverse linearly across and down the screen, respectively. In a composite video signal, the information for controlling these drivers is contained with the actual picture information. The information is coded as horizontal and vertical synchronization pulses. The horizontal pulses have a frequency of 15.75 kHz (60 Hz multiplied by 262.5 raster lines), whereas the vertical pulses have a frequency of 60 Hz (one for each field). This means that each raster line is scanned in approximately 63.49 μs. Approximately 11 μs of that time is used for the sync pulse and horizontal blanking. The horizontal blanking is necessary because it effectively turns off the display as the beam moves back to the left portion of the screen at the end of a line. This is called retrace. The rest of the portion of the 63.5 μs is the active scan region; and the displayed information is represented by a voltage level--the lowest representing black, the highest representing white, and in between representing some "grey level." The levels used for blanking and the pulse waveform are voltages that are less than that of black, making them "blacker than black."

There is also a vertical blanking period that occurs when the beam retraces from the bottom of the screen to the top. Because of some pulse equalization to maintain stability, and because it takes physically longer to perform a vertical retrace, horizontal scanning continues during the vertical blanking period. This effectively prevents some of the 525 raster lines from being displayed; and because the actual time can vary, the number of eliminated rasters can vary from 20 to 45, depending on the system.

The overall composite video signal is limited to 4.5 MHz, again, to preserve bandwidth. The signal level is 1.0 volt peak-to-peak, and polarized such that the pulse signal level is the most negative value and the brightest white is the most positive value. The actual dc level of the signal is not important, because the signal is usually ac-coupled to an input.

The SLAM forms a composite video signal by using the horizontal and vertical drivers that control the laser scanner to drive a video signal generator as well. Since the receiver output is a voltage level signal and is already synchronized with the horizontal and vertical pulses, it can be added to those synchronization signals. The receiver output is ac-coupled to the video signal generator, internally dc-restored and scaled to an appropriate amplitude and added to scaled vertical and horizontal pulses to form a composite signal that is output to the video monitor. Those synchronization signals are also available as separate horizontal and vertical pulses or as one composite synchronization signal (no active scan information).

2.3. Two-Dimensional Images

Since this thesis deals with the concept of imaging, it is important to fully understand the principles involved. An image is merely a representation of something real. In the case of the SLAM functions, the receiver output and composite video signal both represent, in the form of a one-dimensional electrical signal, either the attenuation of acoustic energy or constant phase contours of that attenuation (in the interference mode). Through the proper scanning technique, the one-dimensional signal can then be decoded into the form of a two-dimensional image.

In all the data acquisition and data analysis systems built for the SLAM, analyses are performed by some sort of computer. In order for the computer to process information, that information must be in digital form. Consequently, the one-dimensional electrical signal is digitized. When converting an analog signal such as the receiver output or composite video signal to digital representation, via an analog-to-digital converter (ADC), sampling frequencies are an important consideration. Even after meeting requirements such as the Nyquist criterion, which requires that the sampling rate must be at least twice the maximum frequency of the signal to be sampled, other things must be considered when the signal represents a full two-dimensional image.

In a two-dimensional image system such as that used in the SLAM, the pertinent information in the output signals is contained in blocks that will become raster lines. When that portion of the signal is sampled by an ADC, each sample that is taken in the area

that would be displayed on a monitor is called a pixel, which is an abbreviation for picture element. The number of bits used to represent the pixel gives the dynamic range--the numerical distance between the smallest and the largest representable values. For an 8-bit conversion, there are $2^8 = 256$ possible discrete grey levels, ranging from 0 (black) to 255 (white). Also, the more bits, the smaller the difference between two adjacent grey level values.

The ability to resolve between two adjacent point sources is called the resolution of an imaging system. The closer the two point sources are, the higher the resolution has to be to resolve them. This relates to the pixels because higher resolution requires more pixels, implying a higher sampling rate. Consequently, increasing resolution is limited only by the current sampling technology. The total number of pixels in an image is given by the number of pixels per line multiplied by the total number of lines. Although technically incorrect, this is sometimes called the resolution of a system.

Although the sampling rate seems to represent two different concepts--maximum resolution in an imaging system, and twice the maximum possible frequency of a one-dimensional signal--the concepts are actually related. This is especially true in the case of a SLAM imaging system, since the image is generated from a one-dimensional signal. For example, if it is intended to use the entire 4.5 MHz bandwidth of a video signal, then it is known that the data acquisition portion of an imaging system must sample the signal at the Nyquist rate of 9 MHz or faster to prevent aliasing.

However, the most important relationship is that between temporal and spatial frequencies. This is due to the fact that a physical area is scanned within a certain time period. An example of this relationship is the SLAM in the interference mode. In this mode approximately 39 equally spaced vertical lines are generated. Since the horizontal scanning distance is approximately 3 mm, this implies that the distance, or period, between lines is around 77 μm. Consequently, this implies a spatial frequency of 1/(77 μm), or approximately 13000 m^{-1}. Spatial frequency can be determined from the pixel information as well. For example, if a system has a "resolution" of 390 x 200 (390 pixels per line, 200 lines), then the period of each interference line will be 10 pixels, with a spatial frequency of 1/(10 pixels). Because each pixel represents the average intensity of a physical area of size 7.7 μm by 10 μm (3 mm/390 by 2 mm/200), the physical spacing will be 7.7 μm multiplied by 10, or 77 μm as before.

However, since the sampled video signal is also a time-domain signal, the interference lines can be represented by a temporal period and frequency as well. For example, if the active (displayed) scan time of a raster line is 52.5 μs, then the period of the interference lines is approximately 1.3 μs. This implies a temporal frequency of 770 kHz. In this imaging system, each pixel would represent 0.13 μs in the horizontal direction. Although the actual time spacing between two adjacent pixels on different lines is 63.5 μs (one full raster line length), it has no spectral meaning, since this is a one-dimensional signal.

The importance of this relationship is that if a spectral analysis is to be performed on an image, a different basis can be used. However, the spectral analysis is limited to a one-dimensional analysis of an individual raster line. If analysis of a particular column or if a two-dimensional spectrum is desired, only the space domain can be used.

CHAPTER 3

PREVIOUS AUTOMATION SYSTEMS AND ALGORITHMS

3.1. Consideration of Previous Systems

It is important to take a close look at the previous automation systems used with SLAM, particularly because certain algorithms were developed to perform the image analysis [9-13]. Therefore, it was unnecessary to develop new algorithms for the new system, since the original operations were proven to provide valid results. Consequently, the analysis operations in the new system are largely enhanced versions of the previous system. The previous system will be examined here along with the methods for performing attenuation coefficient and speed of sound measurements.

3.2. Previous System Hardware and Software

The previous image acquisition and analysis system [14] consisted of two parts. The acquisition portion of the system was designed specifically for the SLAM and built mostly from discrete parts and prefabricated building blocks such as a TRW analog-to-digital converter board. The ADC was an 8-bit unit, with a 30 MHz sampling rate, and sampled the receiver output of the SLAM. Since the receiver output carries no timing information, synchronization was made thrcugh use of the separate horizontal and vertical pulse signals. After being sampled, the data were sent to the system's internal main memory. Because of memory limitations, the image resolution was 768 x 255 pixels. The 768 pixel specification is effectively half of what the ADC is capable of performing at 30

MHz. Also, the 255 lines represent one field, not a complete frame, and several of those lines occur during the backtrace and contain no data. Overall system control was carried out by a Zilog Z-80 8-bit microprocessor, which was the system-to-user-interface via a terminal.

The system was also capable of performing high-speed multiplications and additions, although most analysis was done by sending image data to the laboratory's Digital Equipment Corporation VAX 11/730 mainframe computer. The transfer of data was performed using a 9600 BAUD serial connection. One full image consisted of 198,208 bytes, and took approximately 186 seconds to transfer uninterrupted.

The software of the data acquisition system processor was mainly embedded as an operating system into ROM. However, a BASIC monitor was provided to allow users to write simple programs to manipulate data in the system. Most analysis was performed on the VAX, where the code was written exclusively in FORTRAN.

3.3. Attenuation Coefficient Measurement

The attenuation coefficient represents the normalized attenuation of acoustic energy after passing through the specimen. To calculate the attenuation coefficient using the SLAM, the insertion loss (IL) method is used. Using the acoustic image, the insertion loss technique calculates the difference in amplitudes of the received signal with and without a specimen of known thickness in the sound path. For tissue specimens, this can be performed by placing the specimen with normal saline, since saline

is a very low-loss material. The saline then becomes an unblocked sound path reference. The IL is calculated several times using several thicknesses, and then plotted versus thickness. The attenuation coefficient is determined as the slope of the best-fit line calculated using the linear least squares method.

In developing this technique on the SLAM, several problems had to be addressed [13]. They included a low signal-to-noise ratio (S/N), nonuniform illumination of the acoustic images, and a nonconstant raster line reference signal that is dependent upon the overall energy of the signal. To compensate for the low S/N ratio, signal averaging was used. To overcome the nonuniformity of illumination, a small subarea of the image was used that was approximately uniform. To account for the changing reference level, the beginning of each raster line (the reference) was sampled, averaged, and subtracted from the main signal level.

The actual analysis was performed by the Z-80 microprocessor system on an area 96 pixels horizontally by 32 pixels vertically located near the center of the image area. The SLAM was adjusted to make that region as bright and uniform as possible. Since the imaging area was fixed, the specimen had to be mobile to allow for the taking of several measurements in both the specimen and the reference region. (Note that the reference region--the area used as the unblocked sound path reference--is different from the reference signal, which is the portion of the receiver output that occurs during the retrace and represents full attenuation.) For

tissue specimens, the mobility was provided by placing the specimen on a thin sheet of plastic coupled to the SLAM stage with water.

To find a value representing the acoustic illumination of the image area, an image was acquired, including the reference signal. The reference signal for each raster line used in the subarea was averaged using 10 samples and then subtracted from the pixel values on that raster line. All the pixels in the subarea were averaged. This procedure was repeated 8 times, and the overall average was taken. This value was converted to decibels, and then displayed on the terminal screen approximately every 5 seconds. Equation (1) is a mathematical description of this process.

$$V = 10 \log \left\{ \frac{1}{n} \sum_{1}^{n} \left[\frac{1}{3072} \sum_{i=y}^{y+31} \sum_{j=x}^{x+95} \left(v_{ij} - r_i \right) \right] \right\} \qquad (1)$$

where V = the average of the image area averages in decibels,

v_{ij}= the value of the pixel at position (i,j),

r_i =the average reference signal level for the i^{th} raster line,

n = the number of times the subarea is averaged,

i = the image row index,

j = the image column index,

x = the starting column number of the subarea, and

y = the starting row number of the subarea.

To measure insertion loss, the specimen was first placed on the plastic sheet surrounded by saline. Next, the sheet was positioned so that the saline was in the subarea, and the SLAM was adjusted to maximize brightness and uniformity in that area.

Several values of V (from Equation (1)) were taken in saline by moving the sheet, and the values were recorded by the operator in a notebook. Similarly, several values of V were taken in the specimen region and recorded. Since the saline was assumed uniform, the saline V values were averaged to form V_r. If the values recorded for the specimen are V_s, then the insertion loss values for a particular thickness are given by

$$IL = V_r - V_s. \tag{2}$$

These insertion loss values are then used to calculate the attenuation coefficient as described previously.

3.4. Speed of Sound Measurement

The ultrasonic speed of sound in a specimen can be determined using the SLAM interference image. The vertical lines, or fringes, represent equal phase wavefronts of the sound after passing through the specimen. The specimen is placed on the stage with a homogeneous reference medium with a known speed of sound such that the image is separated into three horizontal regions. The top region is the reference medium, followed by the specimen region, and the bottom is also the reference medium. This causes the interference lines to shift upon entering the specimen region, and shift back upon reentering the bottom reference region. The specimen must be prepared properly so that there is not an abrupt shift in the interference lines. The speed of sound, c_x, in the specimen can be determined from the normalized fringe shift, N, using the following equation derived in [10]:

$$c_x = \frac{c_0}{\sin \theta_0} \sin \left\{ \tan^{-1} \left[\frac{1}{(1/\tan \theta_0) - (N\lambda_0/T\sin \theta_0)} \right] \right\} \quad (3)$$

where c_0 = the speed of sound in the reference medium,

λ_0 = the wavelength of sound in the reference medium,

θ_0 = the angle of sound, from the normal, in the reference medium, and

T = the thickness of the specimen.

Only N must be determined; the other quantities are known.

The normalized fringe shift, N, refers to the amount of lateral fringe shift relative to the fringe spacing. A number of ways have been used to determine N. The first method required measuring the fringe shift and fringe spacing by hand directly off the monitor or from a photograph [10]. This method relied heavily on operator experience and ability, was time-consuming, and resulted in only a few speed values per specimen. Subsequently, a computerized acquisition system was built that used a correlator to track each of the interference lines, and allowed computer determination of fringe shift [9,11,15]. Using this method, 30 to 39 speed values could be obtained per specimen which allowed statistical analysis to be performed more effectively. This tracking system worked well with homogeneous specimens which have smooth interference lines; however, on heterogeneous specimens it often failed.

To analyze heterogeneous specimens, a technique called the spatial frequency domain technique (SFDT) was developed [11]. It was later modified [14] to improve results. A brief description

of the SFDT is provided here as well as the modifications that were made.

If one were to plot the pixel values for one raster line of an interference image versus their corresponding column index, the resulting waveform would approximate a sine wave. This is because the interference lines are alternating light and dark bands. Next, if one were to plot one raster line from the reference medium and one from the specimen region on the same plot, it would be apparent that the specimen sine wave, although of the same frequency, has shifted in phase relative to the reference region. This is the key to the spatial frequency domain technique. If the frequency of the sine waves is ξ_0, then the phase shift $\Delta\phi(-\xi_0)$ is given by

$$\Delta\phi(-\xi_0) = 2\pi y_0/\lambda_y, \tag{4}$$

where y_0 is the amount of horizontal shift, and λ_y is the fringe spacing. Since y_0/λ_y is the definition of the normalized fringe shift N, N is given by

$$N = \Delta\phi(-\xi_0)/2\pi. \tag{5}$$

To find the phase shift, the Fourier transform of each raster line must be taken to find its spatial frequency spectrum. Next, the phases for each line are found by picking the phase at the frequency where the spectrum is a maximum (ξ_0), and taking the negative of that value. This is because the sampled data are real, and from the Fourier transform theory, the magnitude is symmetrical about the imaginary axis while the phase is symmetrical about the origin. Also from the transform theory, it is known that there is a 2π ambiguity problem in determining the phase, since the phase

wraps around. To correct this, a phase unwrapping algorithm can be employed. An average phase for the reference region is calculated, and the average is subtracted from each specimen phase value to yield the phase shift $\Delta\phi(-\xi_0)$.

Since the data are discrete, the Discrete Fourier Transform (DFT) is used. The DFT is given by

$$X_k = \sum_{l=0}^{N-1} x_l \, e^{-j\frac{2\pi}{N}kl} \qquad k = 0, 1, 2, \ldots N-1, \tag{6}$$

where X_k is the discrete frequency domain variable with index k, x_l is the discrete space or time domain (depending on the basis used) variable with index l, and N is the number of points used in calculating the transform. The larger N is, the smaller the frequency spacing, or the more "frequency bins" available. The original SFDT used a 32-point Fast Fourier Transform (FFT) (which is a computationally fast DFT algorithm, but requires that N must be a power of 2). It assumed a constant fringe spacing, approximately 84.5 μm (1.5 μs in the time domain), and hence a constant spatial frequency of approximately 11.8 km^{-1} (temporal frequency of around 660 kHz). Consequently, when the phase for each raster line was being calculated, it was always at the same discrete frequency. (Note that this frequency is not exactly the desired frequency ξ_0, because some leakage does occur when using the DFT. In the SFDT and its derivatives, the nearest neighbor method is used, and no interpolation is performed.)

The algorithm was later modified, because it was found that for heterogeneous specimens, the actual spacing of the interference lines can vary significantly. Consequently, the algorithm would look for the maximum spectral component within a certain band of frequencies that were known to be the limits of the variations. Also, because only a certain number of frequencies bins were to be checked, a normal DFT was calculated for those frequencies alone. The actual DFT size, N, was expanded to decrease the frequency spacing. However, since speed variation can be very localized within a specimen, the actual number of data points used in the calculation had to be limited to a window length, W, shorter than N. The remaining N - W data points were zero-padded.

The previous system employed a 768-point DFT. It assumed temporal frequency limits on the interference lines to be 600 kHz and 915 kHz. These values correspond to 18 frequency components that were calculated and checked. A data window length of 20 pixels was used since this corresponded to the average period of the interference lines in that system. The actual calculations were performed on the VAX, and the necessary complex exponentials for calculating the desired frequency components were precalculated and stored to reduce computation time. Input for the program included the actual digitized data, and a file the user could modify which specified the lines to be used as reference and the starting column of the analysis. After the data were read by the program, the DFT of the specified frequency components was calculated. The frequency with maximum spectral magnitude was

found, and its corresponding phase stored. The phase data were unwrapped. Because of system nonlinearities, the interference lines tend to have a slight slope as they traverse to the bottom of the screen. Consequently, instead of finding an average reference phase value, the algorithm took the reference phase values and calculated a slope and intercept using the linear least squares method. Finally, the phase shifts were found for each raster line by subtracting the corresponding reference phase (calculated by using the raster line index as a range value) from the current phase. The calculated results were stored in a file, and could be plotted on paper to see a cross-sectional view of the speed data versus raster line. This modified SFDT reduced the overall time needed to produce speed data, and since it worked vertically, it calculated one speed value for each available raster line, in this case, 255 values.

As a means of characterizing how heterogeneous a specimen is, a heterogeneity index (HI) can be computed from the speed data [14]. The standard deviation σ is a method of computing HI, since it is a statistical measure of values about a mean. However, finding the standard deviation of the speed values alone is not enough. The SLAM system may contribute to this variation through various sources, including electrical noise, a nonuniform sound field, and errors caused by finite word length in the data acquisition and computation portions of the system. Consequently, a homogeneous medium such as saline, which should have the same speed value throughout, will have some variation due to system

contributions. To correct for this contribution when calculating the HI, the variation due to the SLAM system is subtracted to yield variations due to specimen heterogeneity. As a measure of SLAM system variations, the standard deviation of the reference medium is used since it is a homogeneous substance. The standard deviation of the reference is subtracted from the standard deviation of the specimen. Equation (7) is a mathematical description of calculating the HI.

$$HI = (\sigma_s - \sigma_r) \times 100 \qquad (7)$$

where σ_s is the standard deviation of the specimen, and σ_r is the standard deviation of the reference. For the previous acquisition and analysis system, a FORTRAN program run on the VAX was used to calculate the HI after the user specified the desired regions to be used in the analysis.

CHAPTER 4

CURRENT SYSTEM

The previous system worked well, but it had its shortcomings. Consequently, a new system was proposed which is the primary subject of this thesis. The algorithms such as those to compute the attenuation coefficient and speed data were left essentially intact because they were effective in the previous system. However, some problems were revealed with the algorithms and had to be corrected. Other problems resulting from incompatibility had to be dealt with in other ways. What follows is an overall description of the new system, hardware and software, and the decisions that went into the new design.

4.1. Hardware

The first decision to be made was the type of computing machine on which the analysis necessary for specimen characterization should be performed. Previous systems had used both a minicomputer and a mainframe, which provided reasonably fast computation times. However, the bottleneck in those systems was the transfer of data from the data acquisition system to the computer, which, for example, was performed previously using a 9600 BAUD serial connection. In addition, the mainframe was a time-shared system, and performance was considerably slower during peak usage hours. Final analysis of the speed and attenuation coefficient data was performed after completing a SLAM session and was done off-line. As a result, it was decided to have a dedicated

processing machine for SLAM analysis algorithms. At that time, there was an IBM Personal Computer AT (AT) available as a resource. Although actual computation time in the AT was slower than for the mainframe, it was decided that the shorter transfer time would shorten overall analysis time, and the AT was selected for the new system.

The AT uses an 8 MHz 80286 microprocessor for its central processing unit (CPU). In addition, it also has an 80287 math coprocessor to enhance the speed of floating point operations. The AT has a 640 kbyte main memory and a 30 Mbyte winchester hard drive for storing programs and data. Also, an ArchiveXL 5540 40 Mbyte tape back-up system was obtained. This device occupies a half-height disk drive space in the AT and uses palm-size tape cartridges. The tape back-up was obtained primarily as permanent storage for image files which are too large and accumulate to quickly to be kept on the hard drive for long periods of time. The AT also has an Enhanced Graphics Adapter (EGA) with a monochrome monitor which allow for some graphics capabilities. More detailed information on the IBM PC/AT can be found in the references [16].

Once the computing portion of the system had been decided upon, the next decision to be made concerned the data acquisition section of the new system. The previous system had been designed and built specifically for the SLAM. Its primary limitations were the speed and size of its memory. The memory was selected at the time because it was what was available for reasonable cost.

Technology has improved since that time, and memory integrated circuits are more dense (larger capacity) and faster at a lower cost. But technology has also improved at the building block level of system design as well. Devices called video frame grabbers have been built that plug directly into the expansion slots of IBM PC-type computers, and digitize and store video images. Using the frame grabber would save much of the time required to design, build, and debug a new hardware system. Consequently, it was decided that this would be the approach of the new system; the frame grabber and the AT would provide a self-contained image data acquisition and analysis system.

The frame grabber selected was the Data Translation DT2851 video frame grabber. The DT2851 fits into one standard extended expansion slot in the AT and is powered directly by the AT. Communications between the AT and the frame grabber are made through the standard IBM PC I/O registers. The DT2851 requires a standard composite video signal as an input. The DT2851 has an 8-bit 10 MHz ADC which is one-third of the sampling rate of the previous system (although the effective rate of that system was 15 MHz). However, because of the spacing of the interference lines and other points of interest, it was decided that the higher sampling rate was unnecessary. The DT2851 also acquires one full frame instead of just one field. As a result, one full image is 512 pixels by 480 lines. One full frame requires 256 kbytes of memory for storage, and the DT2851 has memory for two such frames (buffers) consisting of dual-ported random access memory (RAM).

The buffer memory resides on the AT bus in the logical address space of the AT's extended memory. This allows both the frame grabber and the computer to access the memory if desired.

The DT2851 can output the contents of either of its buffers to a standard analog video monitor with red-green-blue (RGB) inputs. Although the input is a digitized black and white signal, the frame grabber can output a color image. This is performed by using output look-up tables which use the grey level data stored in the buffers to drive 3 digital-to-analog converters (DAC), one each for red, green and blue. In this way, "false color" outputs can be generated to enhance an image. The frame grabber can either use external synchronization stripped off the incoming composite video signal, or it can use its own internal synchronization for displaying images when an external signal is not available or when frame acquisition is complete. An important function of the DT2851 is the pass-through mode where an image is digitized and then reconverted to an analog image in real-time. This effectively allows the image monitor to be used as a normal video monitor where the user is unaware of the digitization process. The DT2851 has many other features such as a full screen cross-hair, and input look-up tables which allow preprocessing changes to incoming data, and the ability to mask (prevent alteration of) specified bits in the pixel bytes. Together, they can allow, for example, two 4-bit images to be combined and displayed in one 8-bit buffer. More in-depth descriptions of the features and specifications of the DT2851 can be found in its technical manual [17].

To increase the speed of some imaging functions, a Data Translation DT2858 frame processor board was purchased. The DT2858 also plugs into an extended expansion slot of the AT and communicates through the standard I/O registers. However, it connects to the DT2851 through a pair of 10 MHz asynchronous parallel I/O ports. These allow high-speed transfer of images between the two devices without using the AT's bus. Various logical and arithmetic functions can be performed on images transferred to the frame processor, such as adding frames or calculating the logical "AND" of two frames. In addition, these functions can be used to produce more powerful functions which the DT2858 performs. These include frame averaging (up to 256 frames can be averaged), convolutions (to perform filtering), and histograms. A histogram is a representation of the distribution of pixel values in an image. It is computed by incrementing a counter for each grey level value every time that level is encountered in an image. The DT2858 performs these functions much faster in hardware than the AT could by running a program. The DT2858 memory is 256 k by 16-bit words. The longer word length prevents overflow when adding frames together. More detailed information on the DT2858 and its features is also available [18].

4.2. Software

Once the hardware decisions had been made and the devices obtained, it was necessary to start implementing the algorithms necessary for SLAM specimen characterization. The computer language "C" was selected for writing the software because it

represents the best compromise between the capabilities of a high-level language and the necessary hardware interaction. Next, the Microsoft C Optimizing Compiler was employed because it produces efficient machine code for the IBM PC and compatible machines. To simplify communications and data transfers with the frame grabber and frame processor, Data Translation's DT-IRIS software package was obtained. DT-IRIS is a library of subroutines written specifically for accessing the Data Translation hardware. The subroutines are called from C programs as functions, the equivalents of subroutines in the C language. The DT-IRIS routines access internal functions of the DT2851 and DT2858, and simplify performing, for example, convolutions, frame averaging, setting look-up tables, and the transfer of pixel values in the frame buffers to and from user-defined arrays. Information on the programming tools can be found in the references [19-21].

In the sections that follow, the new programs created for data analysis will be described. Although the basic algorithms remain the same as in the previous system, several differences exist and will be examined here. The actual C coding for these programs can be found in the appendices; however, necessary inputs required from the user and subsequent outputs are described.

4.2.1. Attenuation coefficient measurement

The process to calculate the attenuation coefficient of a specimen is very similar to the previous system. The insertion loss (IL) method is still used. The actual data for calculating IL is generated by a program called "atten." Atten produces the

V values from Equation (1) and stores them in a file. The data are in a format that can be used by an existing C program called "regress," which calculates both IL and the attenuation coefficient.

The program atten calculates V values using Equation (1) with one important difference. The value r_i is no longer available. Recall that r_i is a reference value for a raster line found by averaging several samples of the SLAM's receiver signal during the retrace. Since the DT2851 samples a video signal, that portion of the receiver signal has been replaced by the horizontal sync pulse and the horizontal blanking signal. Consequently, it is not available to the user.

This situation presents a problem, because the reference level of the receiver output varies, depending upon the overall energy of the signal. Since the composite video signal is created from the receiver output, the problem carries over into the video signal as well. Without a reference, reproducible values of V cannot be obtained. Since it was desired to retain the new frame grabber imaging system, solutions to the problem were proposed. Software compensation to the problem was desired so that no external hardware would be necessary. In this way, the problem would be invisible to the user. Several software solutions were tried using difference references and imaging areas, but none of these were successful. A hardware solution was created that effectively created a new composite video signal by adding the receiver output with a composite-sync-only signal. This was attempted because it

was found that the changing reference problem in the receiver output was due to the coupling circuit between the receiver and the video signal generator. When disconnected from the video signal generator, the receiver reference would stay at a constant level. However, it was then discovered that the overall receiver output drifted on an unpredictable dc-offset voltage that could not be removed without causing the unstable reference signal to recur. This problem may be due to the age of the SLAM circuitry, and since it was desired not to modify that circuitry, a compromise solution was proposed.

The actual solution does not involve any hardware changes, but requires certain actions to be taken by the user. The solution is similar to that used in Equation (1) in that a reference level is subtracted from each pixel value in a raster line. The difference is that the user must now make the reference available. The reference level in the receiver output represents maximum attenuation of the acoustic signal as retrace occurs. The solution to the problem is simply to carry that maximum attenuation into the viewing area where the data can be digitized and used.

There are a few ways to carry the maximum attenuation into the displayed image area, all of which effectively "black out" the left-most portion of the display. It is important that the left part is used, since the scanning is done left to right and the actual voltage level of the reference (both the real receiver output reference and the expanded artificial reference) depends on the previous raster line energy. Once the reference area is

extended into the imaging area, r_i is calculated by averaging the first ten pixel values in each row, and subtracted as shown in Equation (1).

One way to black out the left-most portion of the screen is done fairly easily when the water stage is installed. The physical design of the water stage allows the stage to be moved so that the transducer and the metal area that surrounds it can be shifted into the image area. Once the metal area moves into the image area, no sound is transmitted or received in that portion--maximum attenuation. The design of the fused-silica stage does not allow this method. Another way to fully attenuate the signal is to block the path of the laser light emitted onto or reflected from that portion of the image. This can be done by removing a portion of the gold film from the coverslip or by placing a piece of thick plastic in the path of the laser where the left part of the viewing area is scanned.

One other phenomenon was observed while the reference problem was being corrected. When the user wishes to change the overall signal level of a SLAM image, he can use two different controls. One is the receiver gain which controls the signal level leaving the receiver in Figure 1, and the other is the video gain which controls the signal level of the active scan portion of the video signal that feeds the frame grabber. The phenomenon discovered was that changing the video gain has an interesting effect.

To understand the effect, one first has to understand that to use the insertion loss technique to calculate the attenuation

coefficient, the SLAM is assumed to be a linear system. For example, if one were to plot the V values from Equation (1) versus known inserted attenuation, then the resulting plot should be a straight line with some slope m. Apparently, changing the video gain has the effect of changing the slope; increasing the gain increases the slope, decreasing the gain decreases the slope. The actual slope is unimportant, because the attenuation coefficient itself is calculated from the slope of insertion loss values for several thicknesses. Consequently, if the SLAM output is linear, the ratio between the insertion loss values for two thicknesses will remain constant, independent of the slope of the V values versus inserted attenuation. The actual value of that slope only changes the y-intercept on the attenuation coefficient plot. However, the slope must remain constant throughout the measurement of all the insertion loss values for all the thicknesses, or the attenuation coefficient measurement will be invalid.

To take data using the atten program, the specimen is prepared the same way it has been prepared in the past. The SLAM user then selects the acoustic image mode and runs the program by typing "atten" at the prompt on the AT. The image monitor will then display the same image (in real time) as the SLAM monitor, with two exceptions. The image monitor also displays an outlined box (overlaid on the acoustic image) that represents the subarea to be used for calculating V. This is available to assist the operator in making that region as bright and uniform as possible. In addition, an overlaid line drawn from top to bottom on the left

side of the screen represents the minimum amount of blacked-out reference area required to calculate V correctly. The next step is to perform a histogram. This allows as much as possible of the dynamic range of the frame grabber to be used. The user is provided with both a graphic representation of pixel value distribution as well as the absolute minimum, absolute maximum, average, and most prevalent pixel values. The user adjusts the receiver gain such that dynamic range is maximized without saturating (level too high) the input. The video gain can only be changed initially, when calculating the V values for the first thickness. It should remain at that setting until all the thicknesses for a particular specimen have been analyzed.

Once these procedures have been completed, the user is asked for the specimen thickness and a file name to store the data. If an existing file is specified, atten will append the new data to the old file. This is done to allow data for several thicknesses to be stored in one file which can be read in by the aforementioned regress program. Atten can now calculate the V values for storage. The minimum-reference line and subarea imaging box remain on the image display to assist the user. The user has four options while in the data-taking mode. To calculate V_r, a V value in the reference region (not to be confused with maximum attenuation reference), the user moves the specimen such that the desired reference region falls within the imaging box and presses "r" on the computer keyboard. This causes 8 frames to be acquired and averaged (to improve the signal-to-noise ratio). The value V_r is

then calculated and stored in the specified file. In addition, V_r is printed on the computer monitor. Similarly, to calculate V_s, the V value for the specimen, the specimen is moved into the imaging box and the user presses the "s" key. This stores V_s in the specified file and prints it on the screen. The third option is to terminate the program which is done by pressing the "enter" key. Finally, if any other key is pressed, V is calculated and displayed on the screen, but it is not stored in the output file. This is useful for displaying V values in questionable areas of the specimen or reference regions before storing them in the output file. In addition, each time a V value is calculated and displayed, the current absolute maximum and absolute minimum pixel values are displayed to keep the user aware of possible saturation problems which may occur after data calculations have begun. Under such conditions, the data may not be valid, and the user should begin again. The user may take as many reference and specimen data points as desired and terminate the program normally. When data acquisition for all thicknesses has been completed, the user should run the regress program and specify the corresponding file to calculate the insertion loss values and the attenuation coefficient for that specimen.

4.2.2. Speed of sound measurement

Calculation of the speed of sound in a specimen is done in a manner similar to the previous system. The specimen must be prepared in the same manner as before. The spatial frequency domain technique is still used to generate the speed values. The

analysis still works from top to bottom, and a linear least squares algorithm is still used to find a reference phase from the selected reference regions. Some features of the speed algorithm have changed, however. Because the DT2851 samples a full video frame, 480 speed values are available in one pass as opposed to 255. Also, the user is provided with many more options that affect how and where the analysis is performed.

The main modification to the SFDT algorithm was in regard to the DFT calculation. Although the number of pixels per line of the current system is somewhat less (33%) than that of the previous system, the current 512 pixels is more than enough to determine the frequency of the interference lines. A 512-point DFT was selected in favor of the 768-point DFT used previously. Using a slightly wider frequency band than before, 21 discrete frequencies are checked for maximum magnitude after calculating the DFT.

One other important modification was made. It was discovered in the course of debugging the new speed program that when an image with known interference line frequency was analyzed, the program would pick the incorrect frequency. Figure 4 shows this problem in more detail. In the figure, a portion of the DFT of three waveforms is plotted. Each waveform is a sine wave with a period of 16 pixels. Using a 512-point DFT, the spectral magnitude should peak at frequency bin 32 with no leakage. For a complex sine wave $e^{j\phi k}$ (= $\cos(\phi k) + j\sin(\phi k)$) where ϕ is the frequency of the sine wave, the DFT in Figure 4 peaks at 32 as expected. According to the Fourier transform theory [22], if a sine wave is sampled for

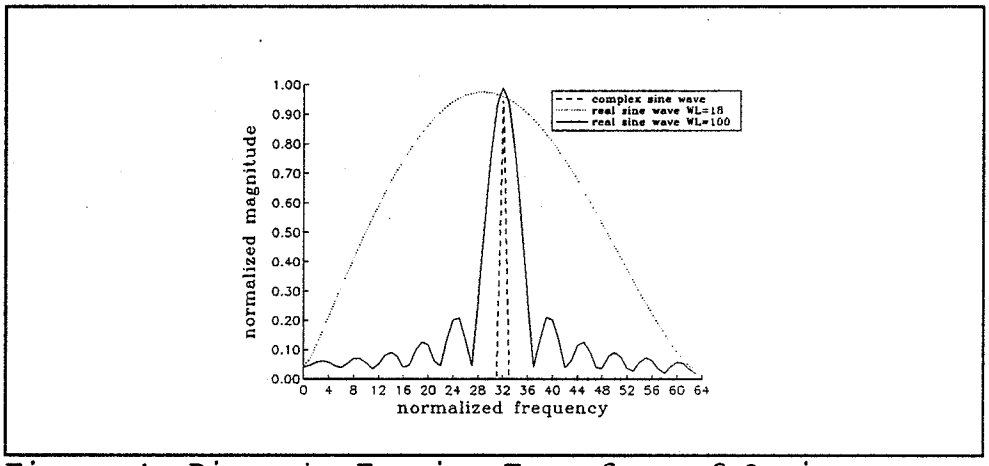

Figure 4. Discrete Fourier Transform of 3 sine waves.

a given window length and zero-padded for the remainder of the DFT length, the resulting spectrum should resemble a sinc function, where sinc(x) = sin(x)/x. The width of the main lobe of the sinc function depends on the window length of the sampled sine wave, with a wide lobe corresponding to a short window length. This is shown in Figure 4 as the DFT of a real sine wave (sin(ϕk)) with a window length of 100. However, the third plot in Figure 4 is the DFT of a real sine wave with a window length of 18. In this case, the resulting DFT is a sinc waveform with a very wide main lobe that clearly does not peak at 32 as expected (it peaks at 28). This does not seem to follow the theory which predicts a wide main lobe that still peaks at 32. This DFT skewing problem stems from the fact that the sampled waveform is real, whereas the theory predicts for complex sine waves. Consequently, the DFT will peak at a frequency at or near the expected frequency, depending upon the phase of the input.

A possible explanation for the skewing problem is that the all-real frequency components required to produce the truncated sine wave are greater in magnitude than the expected frequency. Since the amount of truncation is dependent upon the phase, the frequency component magnitudes will vary. However, apparently the phases of the frequency components are close in value to the phase of the desired frequency. This was found to be true in the previous system as well as in the new system, because acceptable values for speed were being calculated. Since the phase value of the selected incorrect frequency is near in value to the actual phase on a given raster line, the SFDT would produce speed results that are near the expected values. In the previous system, the error due to this deviation apparently became part of the overall system error.

Since the SLAM data acquisition system can produce only real data, the DFT skewing problem cannot be entirely corrected. It was desired to allow the skewing problem to contribute as little error as possible; consequently, a compromise solution was proposed. The skewing problem is minimized by increasing the window length. The longer the window length, the closer the DFT spectrum approaches the actual spectrum for a complex waveform. This can be seen in Figure 4, where the DFT for a real sine wave with 100 samples clearly approaches that of a complex sine wave. The current system uses a 26-point window length. This still produces occasional error, but no more than is due to DFT leakage (i.e., the current system will choose a frequency within one bin

of the actual frequency). Since, as previously discovered, the phases of the components track the phase of the actual maximum frequency, the occasional deviation due to incorrect frequency selection is not readily apparent. The only disadvantage due to increased window length is the increased computation time.

Aside from those changes, the new algorithm still precalculates and stores the complex exponentials from Equation (6). It calculates the DFT for the appropriate frequencies, finds the frequency with maximum spectral magnitude and its phase. The next steps are to unwrap that phase data and convert the phase data to speed of sound values using Equations (5) and (3).

Before running the speed calculation program, certain steps must be taken. The specimen must be prepared such that the transition between the specimen and reference region is as smooth as possible. This will allow the SFDT algorithm to track the phase values correctly. The size and placement of the specimen are limited such that some portion of the reference region must be within the image area. When specimen preparation is complete, the SLAM is placed in interference mode and analysis may begin.

To perform the speed of sound analysis, the user runs a program called "speed." Speed allows the user to change, during run-time, various factors that affect the analysis. However, many of these required inputs have default values.

Once the speed program is running, the user has three possible images on which the analysis can be performed. The first possible image is the one currently stored in the frame grabber's buffer

memory. Ordinarily, this image is the last image to have been analyzed. By selecting this image, the user can repeat the analysis, if desired. The second image that can be used is one that is stored as a file. The user can specify a stored file to analyze an image that was acquired at some previous time. The third, which is also the default image, is to acquire a new image from the SLAM. Before acquiring a new image, the user is asked if a histogram is desired. Using the histogram, the dynamic range can be maximized. Since the SFDT uses the frequency and phase of the interference lines, the actual distribution of the pixel values is not as important as is the case of the atten program. It is more important to prevent saturating the input. When the user is ready to acquire an image, the number of frames to average is requested. The default value is 8 frames.

After the desired image is displayed on the image monitor, the user must select various parameters for calculating the speed of sound using the SFDT and Equation (3). The first user input is the thickness of the current specimen as described in Equation (3). Because this number varies, it has no default value. The speed of sound in the reference medium is also required. Since most of the analysis being performed on the SLAM is for tissue specimens in a saline reference medium, the default value is 1520 m/s, the speed of sound in saline at 30°C. Another input requires the type of stage being used when the image was acquired. The type of stage affects the value of θ_0, the angle of the sound waves through the reference medium. The user also may specify a "skip value." The

skip value directs the analysis to be performed on every raster line defined by skip value x i (where i = 0, 1, 2, . . .). Since picking a number larger than one can reduce overall analysis computation time, this is useful for performing a quick analysis on a questionable specimen. The default skip value is 1, which directs all 480 raster lines to be analyzed.

Two important user-definable items can have a tremendous effect on the program output. The first item is the vertical region of the image where the speed will be calculated. This region can be varied, for example, if a specimen has known regions where the speed is expected to vary significantly. In addition, a region can be selected if it has a reference-to-specimen transition that is smoother than other areas. Using the on-screen cross-hair, the user selects the column that represents the starting point. During the analysis, 26 pixel values are read from the image buffer for each raster line using this column for the first value. The default column makes the analysis region the approximate center of the image. The other important item is that the user may select the areas of the image that are to be used as a reference region. One or two such regions may be specified within the previously selected vertical band. Two regions are useful if the specimen lies in the vertical center of the image, with the reference medium both above and below. If only one region is specified, the other is automatically disabled. The user selects a reference region by using the cross-hair to specify a top row and bottom row for that region. The default is two

regions, one at the top of the image and one at the bottom, which approximates the regions used on the previous system. The default values of the starting column and the reference regions are overlaid on the image monitor prior to selection.

There are two other required user inputs prior to analysis. The first is a file name which specifies where the results of the analysis will be stored. The other input is a character string with which the user can describe or name the specimen.

Once the inputs have been entered by the operator, speed calculations begin automatically. When the analysis is complete, the speed data and other pertinent information are stored in the specified file. Next, a plot of speed versus raster line is displayed on the image monitor (the plot uses the second of the two buffers in the frame grabber; the interference image remains stored in the buffer). This gives immediate results to determine whether or not the analysis was successful. After this, the user can store the interference image in a specified file if desired. To aid in the calculation of the heterogeneity index, the user can perform statistical analysis at this time. If this option is selected, the operator uses the cross-hair on the speed versus raster line plot to specify the regions where the mean and standard deviation of the speed data are to be calculated. The results of the statistical analysis are displayed on the computer monitor and stored in a specified file. Upon completion of this speed analysis session, the user can either quit or may continue with either the current interference image or a new one.

Aside from the main "atten" and "speed" programs, there are other programs available to assist the user in image and data manipulation and analysis. For example, there is a program called "display" which can perform three different functions, depending upon what is typed on the command line. If, for example, the operator types "display" alone, the frame grabber is put in pass-through mode. If "display off" is typed, the display is turned off. If "display <file name>" is typed, the image stored at <file name> is displayed on the image monitor. Typing "grab <file name>" at the command line causes 8 frames to be acquired and averaged and stored at <file name>. Using "plot," a plot of speed versus raster line can be produced from a speed output data file entered on the command line. The default is to plot the speed data on the image monitor. However, by typing an additional character on the command line after the file name (for example, "plot <file name> x," where "x" is any character), the speed data will be plotted on the computer monitor. In this mode, a hard-copy printout of the plot can be produced. A program called "stats" allows for post-processing of the speed program output data file specified on the command line and calculates the mean and standard deviation of user-selected regions. This can be performed if the user does not wish to do it while the speed program is in use. The last program, called "changew," allows the user to change the location and size of the box used to calculate the V values in the atten program.

CHAPTER 5

SYSTEM VERIFICATION

Although the algorithms for calculating the attenuation coefficient and the speed of sound in a specimen are essentially taken from the previous system, it was important to verify the results in the new system. A verification test designed specifically for this purpose was used on the previous system and was subsequently applied to the new system as well. Details of this test can be found in the references [23]. However, the tests for both the attenuation coefficient and the speed of sound involve using a standard whose attenuation coefficient or speed of sound value was calculated by some independent method.

Before verifying that the new system could produce acceptable attenuation coefficients, it was important to establish the linearity and accuracy of the IL values calculated from the V values produced by the atten program. This was done by employing precision electrical attenuators that were placed between the ultrasonic driver and the transducer, as in Figure 1. There is an impedance mismatch between the ultrasonic driver and the attenuators that causes the output power of the driver to change when the attenuators are switched into the system. To minimize the mismatch problem, a large amount of baseline attenuation (10 dB) was added. To compensate for the decrease in signal, the receiver gain was increased. This tends to increase the signal-to-noise ratio, but frame averaging reduces the effect of the noise.

To evaluate the linearity, a thin layer (< 10 μm) of saline was placed on the stage under a coverslip. The V values for the saline were calculated as a function of inserted electrical attenuation. Setting V_r in Equation (2) equal to V_0, the average of the V values calculated for 0 dB (relative) inserted electrical attenuation, insertion loss values were calculated as a function of the inserted attenuation. Insertion loss for a given video gain setting is plotted versus inserted electrical attenuation in Figure 5. Recall that in Chapter 4, the video gain has an effect on the overall slope of this line. (The insertion loss is essentially calculated by subtracting a constant value from the V values--the slope is unaffected.) The overall shape of the plot is somewhat parabolic; however, in a given operating range, assume the plot can

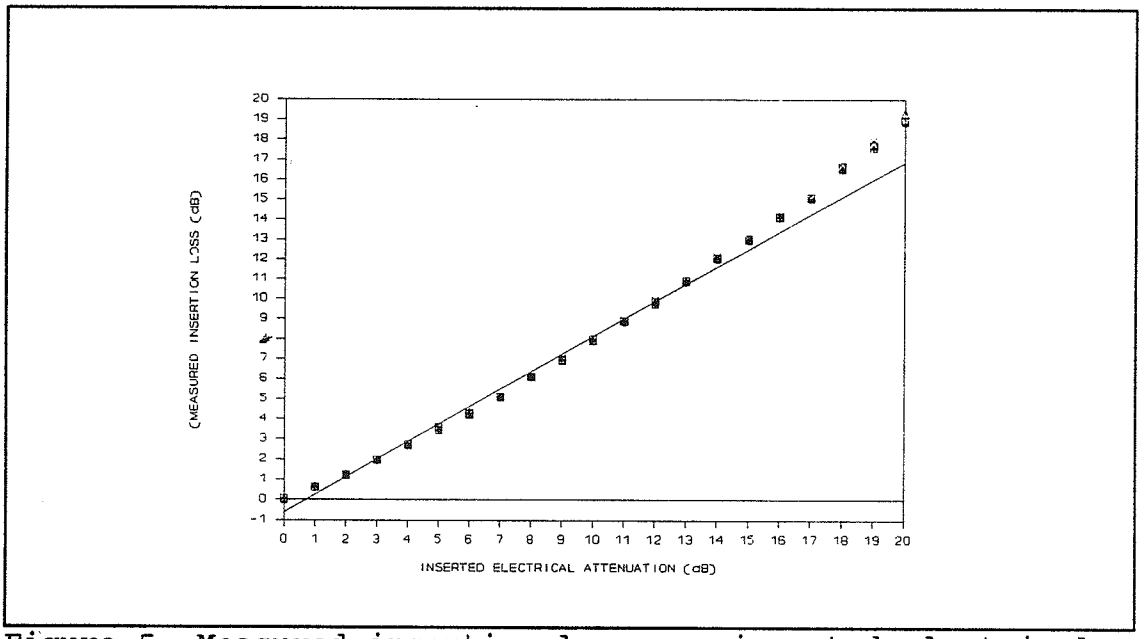

Figure 5. Measured insertion loss vs. inserted electrical attenuation.

be considered linear. The data were analyzed using linear least squares regression. The best-fit line chosen was one encompassing the values from 0 dB to 15 dB. This gives a slope of approximately 0.87 and a correlation coefficient of 0.994. The error associated with the slope was 0.008. Consequently, the assumption of linearity in a given range is valid. Above 20 dB, the pixel distribution is reduced enough to prevent the algorithm from extrapolating accurate results, and the plot will reach some constant level. Performing the regression on all the data reveals the slight nonlinearity, but the analysis for most specimen occurs within smaller, approximately linear portions of the operating environment.

To determine the accuracy of the insertion loss values, the V values for a thin layer of saline were measured with varying amounts of inserted electrical attenuation. Since saline is a homogeneous medium and the subimage area used by the atten program was unchanged, variations in the measured values is assumed to be the result of electrical noise, variations in laser and transducer power, and other unknowns in the overall system. Twenty values were recorded for each of four levels of attenuation at a given video gain setting. The mean and standard deviations were computed for each level. The accuracy of V, calculated as two standard deviations of the mean, was ±0.35 dB. Since IL is calculated by subtracting one V value from another, the accuracy in IL is calculated according to

$$\Delta IL = \sqrt{((\Delta V_s)^2 + (\Delta V_r)^2)}. \qquad (8)$$

The Δ refers to the accuracy associated with a quantity. With an accuracy of ±0.35 dB for both ΔV_r and ΔV_s yields an accuracy for IL of ±0.5 dB.

Once the linearity and accuracy of the insertion loss values were verified, it is important to evaluate the accuracy and precision of the attenuation coefficient measurement. A 10% bovine serum albumin (BSA) solution (10.0 gm BSA per 100 ml distilled H_2O) was used as a standard. The absorption coefficient for BSA has been determined independently with an overall uncertainty of ±5 percent. Since scattering is negligible in homogeneous liquids, the attenuation coefficient was assumed to be equal to the absorption coefficient. The temperature of the BSA solution while on the stage with the laser and ultrasound on has been determined to be 30°C. The absorption coefficient of 10% BSA at 30°C and 100 MHz is 4.18 dB/mm with an overall uncertainty of ±0.21 dB/mm.

The attenuation coefficient for BSA was determined on the SLAM by calculating the insertion loss values for various thicknesses of BSA solution. To control the thickness, a metal spacer (a washer) was used to contain the solution. To calculate IL from Equation (2), a V_r is needed. The average of several V values taken for a very thin layer of BSA (< 10 μm) are used for V_r. Then, six to nine V values were calculated for each four thicknesses of the BSA solution. The four thicknesses used were 0.356, 0.710, 0.920, and 1.080 mm. The accuracy of the spacer measurement was ±2.5 μm, and since this contributed less than 0.3 percent uncertainty to the error of the attenuation coefficient,

the thickness uncertainty was ignored. This procedure was repeated, and the V values were input into the regression program. The program generated two attenuation coefficient values of 4.27 dB/mm and 3.97 dB/mm, and the 95 percent confidence intervals for those two values were 3.55 to 4.99 dB/mm (±16 percent) and 3.34 to 4.61 dB/mm (±16 percent), respectively. Using the known value of 4.18 dB/mm, the errors in the calculated values were -2.2 percent and 5.0 percent respectively. With a ±5 percent overall uncertainty for the literature value, and using the higher error value of 5 percent, the accuracy uncertainty becomes ±10 percent (the sum of ±5 and ±5 percent). Since the 95 percent confidence interval was ±16 percent, the precision is therefore ±16 percent.

The accuracy and precision of the spatial frequency domain technique used to calculate the speed of sound values were also tested. Dow Corning 710 oil was used as a standard. It has a published speed of sound value of 1340 m/s at 30°C. Normal saline was used as a reference medium and has a reported speed of sound value of 1520 m/s also at 30°C. The speed values were calculated for several thicknesses of 710 oil by using metal washer spacers as in the attenuation coefficient determination. A small drop of 710 oil was placed in the middle of the spacer, and saline was added that surrounded the oil. Since the saline and the oil do not mix, a smooth transition region exists between the two separate substances, meeting the requirements of the SFDT algorithm.

The results of this test were inconclusive. The speed of sound was calculated for several thicknesses, and the results had

means in the range 1265 m/s to 1290 m/s with standard deviations ranging from 4.0 to 8.0. Since the results were consistent but not accurate (relative to the assumed 1340 m/s), Equation (3) and the manual method described in [10] were used to check the values. This method, which measures the fringe shift directly and does not use the SFDT, is not as precise as SFDT. However, it can determine values within a given range with reasonable accuracy. Applying the manual method, speed values in the range of 1270 m/s were calculated, agreeing with the speed program output.

The reason for the discrepancy is unknown. In addition, the speed program has been used to calculate the speed values of certain biological specimens whose characteristics were known in advance. In those cases, the output of the speed program was approximately the expected results. Actual verification of the speed analysis will be determined at a later date, with either a new Dow Corning 710 oil sample, or with some other standard.

CHAPTER 6

DISCUSSION

6.1. Comparison of the New System to the Previous System

Since the current system was conceived and built as an improvement over the previous system, it is important to compare the two systems. Most importantly, the new system should maintain the ability to produce acceptable results for speed of sound and attenuation coefficient measurements. The data used to calculate the attenuation coefficient have been shown to have errors that are approximately equivalent to the previous system, and is therefore acceptable.

Actual verification of the speed of sound measurement is currently unavailable, although speed values calculated using the SFDT agreed with values derived using a different method. Consequently, it has been assumed that the speed program is functioning correctly. Theoretically, it should perform at least as well as the previous system, since the technique is essentially the same. Factors that affect the error are a shorter DFT length (less frequency leakage) than the previous system and the corrected skewing problem in the current system. An actual comparison cannot be determined until the new system is retested with another standard.

In terms of other factors, however, the new system does perform better than the previous system. The data generated are more easily manipulated. All data produced are stored in

accessible files on the AT or backed up on tape (which can be easily restored). Linear regression used to calculate the attenuation coefficient from the insertion loss values can be repeated if desired. Speed data can be plotted and printed when required by the user. The previous system allowed only a large plot printed on a line printer attached to the VAX. The new system allows quick plots on the computer or image monitor, and smaller, more manageable printed plots. Also, the images are easily stored and manipulated to reproduce speed data or to display images for presentation.

Visibly displaying images is a major advantage over the previous system, which merely stored the image data and performed analysis by manipulating image data files. Using the image monitor allows the user to see where the image is analyzed. When using the atten program, the user is given a visible measurement region to ensure that correct data are being taken. This reduces questionable results produced from having air bubbles or other disturbances in the measurement region. The previous system relied heavily on operator experience to do this; however, the user was still essentially "blind." In performing speed of sound measurements, the user can instruct the algorithm where to perform the analysis. This allows much more flexibility for specimens with specific multiple regions or for specimens that are not positioned in an easily analyzed manner. The previous system had the capability to change the reference regions and starting column used in the analysis. However, these values were stored in a file, and

could not be changed during run-time, and of course, the user was unable to see exactly where the analysis was being performed.

Having a self-contained system is also an advantage over the previous system. The user is provided with an operating environment with which many people are already familiar (IBM PC and the Disk Operating System (DOS)), or that they can easily learn. It is not required to send data to a different location as was done previously. As a result, the data acquisition and analysis system merely becomes a part of the overall SLAM system.

Another important advantage over the previous system is the time required to analyze the data. The amount of time for calculating the V values using the atten program is approximately the same as the previous system (though the set-up time may be reduced). However, because the V values are stored in a file instead of written by hand as was done previously, the user can input that data file into the regress program and get immediate results upon completion of data acquisition. Previously, calculation of insertion loss and attenuation coefficient was usually performed after the SLAM session was complete. The time required to perform the speed analysis has also been reduced. After specimen preparation was completed, the previous system required as much as 7 minutes during peak VAX usage to produce 255 speed values. If the user wanted to see the results, he was required to go to another room to get a printout of the speed data. The current system requires approximately 1 minute to respond to the required inputs and approximately 2 minutes to perform the

analysis. After the 3 minutes, the user is provided with a visual result of the 480 speed values calculated, at which time he may either statistically analyze the results or repeat the procedure if those results were unsatisfactory.

Perhaps the only disadvantage of the new system is the compromise solution made for the atten program. Recall that the user is required to provide a blacked-out reference area on the left side of the image to produce valid results. Using the water stage, this is not as much of an inconvenience. However, when using the fused silica stage, greater care must be taken. With operator experience, the requirement merely becomes part of the overall specimen preparation process, but again, the solution is at best a compromise.

6.2. Future Work

Although the new system is an improvement over the previous system, certain things should be done to make the new system even better. Many of these ideas were part of the original proposal for a new automated system, but development problems and time constraints caused those of lesser importance to be put aside for the time being. However, in the future, the separate image data acquisition and analysis programs should be combined in one menu-driven program to make almost all the data manipulation invisible to the user. Also, the system cannot presently produce a hard copy of an image. Such an ability would be useful for presentations and reports. This problem may be rectified shortly when the AT becomes part of the laboratory computer network. Certain other computers

in the laboratory with frame-grabbing capabilities have the necessary printer drivers as well. In addition, future work should make more use of the abilities of the DT2851 frame grabber. The abilities to manipulate images and to produce enhanced false-color images have not been fully utilized at present.

An important consideration to future work should also be the floating reference problem described in Chapter 3. The solution used in the current system for the atten program is a compromise, and it would be desirable to correct the problem in a manner invisible to the user. Unfortunately, to fully correct it, the SLAM circuitry may have to be modified or replaced. However, some other solution may be proposed.

CHAPTER 7

CONCLUSIONS

A new imaging data acquisition and analysis system was proposed to enhance the features of the Scanning Laser Acoustic Microscope. After considering different options and the advantages and disadvantages of each, an operating environment was selected. The current system employs an IBM PC/AT with an installed Data Translation video frame grabber to acquire and process data. This system differs significantly from the system it replaces. However, the methods used to characterize specimen characterization, attenuation coefficient and speed of sound, remain similar to those used in the previous system.

Aside from merely generating data, the current system also provides an environment with which the user can actively interact. Through image displays and other visual information, the user can better understand and affect the various analyses that the system is capable of performing. Once data acquisition has been completed and the user adds the required inputs, the system performs the necessary analysis and makes the output readily available to the user. The capabilities of the system have not yet been fully tapped, and certain other features need to be utilized more extensively.

The system is capable of producing valid data, as was its predecessor. Problems with floating signal levels, video gain setting, and idiosyncracies of the Fourier transform initially

produced unexpected results. Certain steps had to be taken that involved compromises to get the system to function correctly. Some of those steps proved to be satisfactory. However, in the future, some other solutions may prove to be necessary to produce a genuinely automated system. Also in the future, the speed program must be validated before results can be assumed accurate, and published. It is hopeful, however, that the current system will prove to be an effective resource in SLAM research.

REFERENCES

[1] Whitman, R. L. and A. Korpel, "Probing of Acoustic Surface Perturbations by Coherent Light," Applied Optics, vol. 8,pp. 1567-1576, 1969.

[2] Korpel, A. and P. Desmares, "Rapid Sampling of Acoustic Holograms by Laser-Scanning Techniques," Journal of the Acoustical Society of America, vol. 45, pp. 881-884, 1969.

[3] Kessler, L. W., P. R. Palermo, and A. Korpel, "Practical High Resolution Acoustic Microscopy," Acoustic Holography, G. Wade, editor, vol. 4, Plenum Press, New York, pp. 51-71, 1972.

[4] Kessler, L. W., "Review in Progress and Applications of Acoustic Microscopy," Journal of the Acoustical Society of America, vol. 55, 909-918, 1974.

[5] Kessler, L. W., "Introduction to Acoustic Imaging Systems," Acoustic Imaging, G. Wade, editor, Plenum Press, New York, pp. 43-63, 1976.

[6] Kessler, L. W., "Imaging with Dynamic-Ripple Diffraction," Acoustic Imaging, G. Wade, editor, Plenum Press, New York, pp. 229-239, 1976.

[7] Shure, A., Basic Television, John F. Rider Publisher, Inc., New York, 1958.

[8] Zworykin, V. K. and G. A. Morton, Television, John Wiley and Sons, Inc., New York, 1954.

[9] Embree, P. M., S. G. Foster, G. Bright, and W. D. O'Brien, Jr., "Ultrasonic Velocity Spatial Distribution Analysis of Biological Materials with the Scanning Laser Acoustic Microscope," Acoustical Imaging, M. Kaveh, R. K. Mueller, and J. F. Greenleaf, editors, vol. 13, Plenum Press, New York, pp. 203-216, 1984.

[10] Goss, S. A. and W. D. O'Brien, Jr., "Direct Ultrasonic Velocity Measurements of Mammalian Collagen Threads," Journal of the Acoustical Society of America, vol. 65, pp. 507-511, 1979.

[11] Embree, P. M., K. M. U. Tervola, S. G. Foster, and W. D. O'Brien, Jr., "Spatial Distribution of the Speed of Sound in Biological Materials with the Scanning Laser Acoustic Microscope," IEEE Transactions on Sonics and Ultrasonics, vol. SU-32, pp. 341-350, 1985.

[12] Tervola, K. M. U. and W. D. O'Brien, Jr., "Spatial Frequency Domain Technique: An Approach to Analyze the Scanning Laser Acoustic Microscope Interferogram Image," IEEE Transactions on Sonics and Ultrasonics, vol. SU-32, pp. 544-554, 1985.

[13] Tervola, K. M. U., S. G. Foster, and W. D. O'Brien, Jr., "Attenuation Coefficient Measurement Technique at 100 MHz with the Scanning Laser Acoustic Microscope," IEEE Transactions on Sonics and Ultrasonics, vol. SU-32, pp. 259-265, 1985.

[14] Steiger, D. L., "Ultrasonic Assessment of Skin and Wounds with the Scanning Laser Acoustic Microscope," M.S. thesis, University of Illinois, Urbana, IL, 1986.

[15] Foster, S. G., "An Image Digitizing System for the Scanning Laser Acoustic Microscope," M.S. thesis, University of Illinois, Urbana, IL, 1981.

[16] Technical Reference to the Personal Computer AT, International Business Machines Corp., Boca Raton, FL, 1985.

[17] User Manual for the DT2851 High Resolution Frame Grabber, Data Translation, Inc., Marlborough, MA, 1988.

[18] User Manual for the DT2858 Auxiliary Frame Processor, Data Translation, Inc., Marlborough, MA, 1988.

[19] Kernighan, B. W. and D. M. Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, NJ, 1978.

[20] Microsoft C 5.1 Optimizing Compiler User's Guide, Microsoft Corporation, Redmond, WA, 1987.

[21] User Manual for the DT-IRIS Subroutine Library, Data Translation, Inc., Marlborough, MA, 1988.

[22] Oppenheim, A. V., and R. W. Schafer, Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[23] Steiger, D. L., W. D. O'Brien, Jr., J. E. Olerud, M. A. Riederer-Henderson, and G. F. Odland, "Measurement Uncertainty Assessment of the Scanning Laser Acoustic Microscope and Application to Canine Skin and Wound," IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, vol. 35, pp. 741-748, 1988.

APPENDIX A ATTEN PROGRAM LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <conio.h>
#include <math.h>
#include <stdio.h>
#include <malloc.h>

/*****************************************************************************/
/*                                                                           */
/*  "atten" calculates values for finding insertion loss and attenuation     */
/*          coefficient.  These values are stored in a file and formatted    */
/*          such that they can be used by the program called "regress."      */
/*                                                                           */
/*****************************************************************************/


main()
{
     int i, j=0, k, line[8], row, col, height, length, minpixel, maxpixel;
     int *red, *green, *blue, *ref, *values;
     long int bin1[256];
     float db, total, temp, curref, thick;
     char ch=0, filename[40];
     FILE *fptr, *outptr;

/*****************************************************************************/
/*                                                                           */
/* i,j,k - counters and array pointers                                       */
/* line[] - array used to store line data to frame grabber for drawing       */
/* row, col, height, length - specify the subimage area used for data        */
/* minpixel - the minimum pixel value                                        */
/* maxpixel - the maximum pixel value                                        */
/* red,green,blue - pointers to arrays that hold the values for ILUT         */
/* ref - pointer to array that holds the reference pixel values              */
/* values - pointer to array that holds subimage area pixel values           */
/* bin1[] - array that holds the histogram bin values                        */
/* db - the value calculated for the subimage area                           */
/* total - temporary value used for summing up pixel values in subimage area */
/* temp - temporary variable                                                 */
/* curref - average of the reference values of the current raster line       */
/* thick - specimen thickness                                                */
/* ch - temporary character variable for user input                          */
/* filename[] - user-specified output file name                              */
/* fptr - file pointer to subimage area box data                             */
/* outptr - output file pointer                                              */
/*                                                                           */
/*****************************************************************************/


/*   allocate memory for the following arrays and report allocation errors   */

     red = (int *)calloc( 256, sizeof(int) );
     if (red == NULL)  { printf("red allocation failed"); exit(); }
     green = (int *)calloc( 256, sizeof(int) );
     if (green == NULL)  { printf("green allocation failed"); exit(); }
     blue = (int *)calloc( 256, sizeof(int) );
     if (blue == NULL)  { printf("blue allocation failed"); exit(); }
     ref = (int *)calloc( 1000, sizeof(int) );
     if (ref == NULL)  { printf("ref allocation failed"); exit(); }
     values = (int *)calloc( 10000, sizeof(int) );
     if (values == NULL)  { printf("values allocation failed"); exit(); }

/*  open file that contains box size and location information               */

     fptr = fopen("c:\\slam\\attwin.pos", "r+");
     fscanf(fptr, "%d %d %d %d", &row, &col, &height, &length);
     fclose(fptr);

/*  initialize and reset frame grabber                                      */

     IS_INITIALIZE();
```

APPENDIX A ATTEN PROGRAM LISTING, Continued

```
     IS_RESET();
     IS_SET_SYNC_SOURCE(1);
     IS_PASSTHRU();
     IS_SET_GRAPHIC_POSITION(row,col);
     IS_SET_FOREGROUND(1);

/* set-up input look-up table 7 so that box can be overlayed on screen       */

     for (i=0; i<256; i+=2)
          {
          red[i] = i;
          green[i] = i;
          blue[i] = i;
          red[i+1] = 255;
          green[i+1] = 0;
          blue[i+1] = 0;
          }
     IS_LOAD_OLUT(7,red,green,blue);

/*   draw box and reference line on screen                                    */

     line[0] = row;
     line[1] = col + length - 1;
     line[2] = row + height - 1;
     line[3] = col + length - 1;
     line[4] = row + height - 1;
     line[5] = col;
     line[6] = row;
     line[7] = col;
     IS_FRAME_CLEAR(0);
     IS_DRAW_LINES(0,4,line);
     IS_SET_GRAPHIC_POSITION(0,10);
     line[0] = 479;
     line[1] = 10;
     IS_DRAW_LINES(0,1,line);
     IS_LOAD_MASK(1);
     IS_SELECT_ILUT(5);
     IS_SELECT_OLUT(7);
     IS_PASSTHRU();
     IS_DISPLAY(1);


     printf("\nAdjust SLAM so that image area in box is as uniform and bright");
     printf("\nas possible.  A black reference region must include the area");
     printf("\non the left side of the screen.  When complete, hit any key");
     printf("\nto perform histogram.  Adjust receiver gain to maximize ");
     printf("\nbrightness without saturating.");
     ch = getch();

/* set-up display so that box and line will not be overwritten by image     */

     IS_ACQUIRE(0,1);
     IS_SELECT_ILUT(0);
     IS_LOAD_MASK(0);
     IS_SELECT_OLUT(0);
     IS_SELECT_INPUT_FRAME(1);
     IS_SELECT_OUTPUT_FRAME(1);
     IS_PASSTHRU();
     IS_DISPLAY(1);

/* perform histogram on image for proper dynamic range set-up               */

     hist(1,1);
     IS_LOAD_MASK(1);
     IS_SELECT_OUTPUT_FRAME(0);
     IS_SELECT_INPUT_FRAME(0);
     IS_SELECT_ILUT(5);
     IS_SELECT_OLUT(7);
     IS_PASSTHRU();
     IS_DISPLAY(1);
```

APPENDIX A ATTEN PROGRAM LISTING, Continued

```
/* obtain output file name and specimen thickness                          */

     printf("\nOutput file name:  ");
     scanf("%s", filename);
     outptr = fopen(filename,"a");
     printf("\nSpecimen thickness (micrometers):  ");
     scanf("%f", &thick);
     fprintf(outptr,"t  %f\n",thick/1000.0);

     printf("\npress 'r' for reference and 's' for specimen.  ");
     printf("\npress ENTER to quit.   \n");

/* begin taking data                                                       */


     while ( (ch = getch()) != 13)
         {

/* acquire and average 8 frames to buffer 1 (will not overwrite box and line)*/

         IS_ACQUIRE(0,1);
         IS_SELECT_ILUT(0);
         IS_LOAD_MASK(0);
         acquire(8,1);
         IS_LOAD_MASK(1);
         IS_SELECT_INPUT_FRAME(0);
         IS_SELECT_ILUT(5);
         IS_SET_ACTIVE_REGION(0,0,512,512);

/* perform histogram to get pixel information                              */

         IS_HISTOGRAM(1, bin1);
         IS_SET_ACTIVE_REGION(row,0,height,10);

/* get reference pixel values from left portion of screen                  */

         IS_GET_REGION(1,ref);
         IS_SET_ACTIVE_REGION(row,col,height,length);

/* get subimage area pixels from boxed region                              */

         IS_GET_REGION(1,values);
         IS_PASSTHRU();
         bin1[0] -= 16384;              /* histogram calculated for 512x512   */

/* find the minimum and maximum pixels                                     */

         for (i=0; i<256; i++)
              {
              if ( bin1[i] != 0 )
                   {
                   minpixel = i;
                   i = 256;
                   }
              }
         for (i=255; i>=0; i--)
              {
              if ( bin1[i] != 0 )
                   {
                   maxpixel = i;
                   i = -1;
                   }
              }

/* calculate the average reference value and subtract it from each pixel in  */
/* the subimage area.  Then add all of theses normalized pixel values.       */

         total = 0.0;
         for (i=0; i<height; i++)
              {
              curref = 0.0;
```

APPENDIX A ATTEN PROGRAM LISTING, Continued

```
            for (k=0; k<10; k++)
                  curref += (float)ref[i*10 + k];
              curref /= 10.0;
              for (k=0; k<length; k++)
                   {
                   temp = (float)values[i*length + k] - curref;
                   total += temp;
                   }
              }

         if (total <= 0)
              total = 1.0;

/* convert the total value to decibels and display it on the screen, and     */
/* write to the output file if necessary.                                    */

         db = ( 10.0 * log10( total/(float)(length*height) ) );
         printf("<min: %3d  max: %3d>  atten:  \x1B[7m%7.3f dB\x1B[0m\n",
                        minpixel,maxpixel,db);
         if ( (ch == 'r') || (ch == 's') )
              fprintf(outptr, "%c  %7.3f\n", ch, db);
         }

/* close output file and reset frame grabber                                 */

    fclose(outptr);
    IS_RESET();
    IS_END();
}
```

APPENDIX B SPEED PROGRAM LISTING

```
#include <math.h>
#include <stdio.h>
#include <isdefs.h>
#include <iserrs.h>          /*************************************************/
#define WINLEN 26            /* DFT window length, no. sample points used     */
#define LOW 28               /* lower bound of DFT frequency indecies checked */
#define HIGH 48              /* upper bound of DFT frequency indecies checked */
#define SLAMFRQ 100.0        /* ultrasonic frequency of SLAM--100 MHz         */
#define REFVEL 1520.0        /* reference speed of sound in saline--1520 m/s  */
#define DFTSIZE 512          /* DFT size used in calculation                  */
#define SAMPLFREQ 9752.0     /* sampling frequency in kHz                     */
#define LIMIT 3.14           /* limit in radians used for unwrapping phase    */
#define DEFAULTCOLUMN 232    /* default analysis starting column              */
#define DEFAULTTOP1 0        /* default of top of reference region 1          */
#define DEFAULTBOT1 79       /* default of bottom of reference region 1       */
#define DEFAULTTOP2 400      /* default of top of reference region 2          */
#define DEFAULTBOT2 479      /* default of bottom of reference region 2       */
                             /*************************************************/

/*****************************************************************************/
/* This program uses the selected interference image of a specimen to find   */
/* the speed of sound in that specimen.  The program uses a spatial frequency*/
/* method to calculate the speed after selecting a desired reference region  */
/* and then stores and plots the resulting data.                             */
/*****************************************************************************/

float phase[480];
main()
{
    FILE *outptr;
    int i, j, k, ref[2][2], xbeg, thick, skip=0, total=0, datbuff[WINLEN],
             reftotal=0, flag1=0, flag2=0, curbuf=1, avgnum, lines[2];
    float dcoffst=0, delta, oldphase, corcoef=0, x, r, z, tempmag,
             pi=4.0*atan(1.0), mag[480], freq[480], phs[480],
             recoef[HIGH-LOW+1][WINLEN], imcoef[HIGH-LOW+1][WINLEN];
    char  ch, filename[80], infile[80], repeat='y', specname[80], input[20];

int startcol[480],topref1[512],botref1[512],topref2[512],botref2[512],refspeed,
             stage = 0;

/*****************************************************************************/
/*                                                                           */
/*  phase[] - contains initially all phase values, then converted to speed   */
/*  ref[][] - array that contains the boundaries for the reference regions   */
/*  xbeg - beginning column of analysis                                      */
/*  thick - thickness of current specimen                                    */
/*  skip - skip value of analysis--raster line (0 + skip*i)                  */
/*  total - total number of raster lines used in analysis                    */
/*  datbuff[] - array containing the pixel values from frame grabber         */
/*  reftotal - total number of reference lines used in analysis              */
/*  flag1 - when set, indicates that reference region 1 is valid             */
/*  flag2 - when set, indicates that reference region 2 is valid             */
/*  curbuf - currently displayed buffer                                      */
/*  avgnum - number of frames averaged for analysis                          */
/*  lines[] - array containing end point of a drawn line                     */
/*  dcoffst - the average pixel value of the current window (and raster)     */
/*  delta - difference between current phase value and old phase value       */
/*  oldphase - phase value immediately preceding current phase value         */
/*  corcoef - correlation coefficient of best fit phase reference line       */
/*  x - current pixel value minus the dc offset value (dcoffst)              */
/*  r - sum of real parts of DFT calculation of current frequency            */
/*  z - sum of imaginary parts of DFT calculation of current frequency       */
/*  tempmag - sum of r squared plus z squared                                */
/*  mag[] - array containing all the maximum magnitudes                      */
/*  freq[] - array containing all the selected DFT frequencies               */
/*  phs[] - array containing the unwrapped phase values                      */
/*  recoef[][] - array containing the real coefficients of DFT summation     */
/*  imcoef[][] - array containing the imaginary coefficients of DFT sum      */
/*  repeat - flag used to continue analysis--while 'y', continue             */
/*  input[] - array used to contain input number string                      */
/*  startcol, topref1, botref1, topref2, botref2 - arrays temporarily        */
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
/*          containing pixel values overwritten by display              */
/*  refspeed - speed of sound in reference medium                       */
/*  stage - flag signifying which stage is being used                   */
/*                                                                      */
/************************************************************************/

    printf("\nThis program calculates the cross-sectional speed from a ");
    printf("displayed image.\nDefault values for inputs are given by \"[]\".\n");

    while (repeat != 'n' )              /* continue analysis until changed  */
    {

/***** set up frame grabber board for analysis calculation of speed ********/

    IS_INITIALIZE();                    /* initialize frame grabber         */
    IS_SELECT_OUTPUT_FRAME(0);          /* select buffer 0 for image output */
    IS_SELECT_ILUT(0);
    IS_SELECT_OLUT(0);                  /* select output lookup table 0     */
    IS_LOAD_MASK(0);
    IS_SET_ACTIVE_REGION(0,0,512,512);
    IS_DISPLAY(1);                      /* turn image display on            */

/********** obtain desired image on which to perform analysis **************/

    printf("\nIs desired image currently being displayed?  [n]  ");

    if ( (ch=getche()) != 'y')          /* if image is display go to analysis */

        {
        printf("\nDo you want to use an image stored in a file?  [n]  ");
         if ( (ch=getche()) == 'y')

            {
            printf("\nfile name:  ");     /* retrieve image from file */
            scanf("%s",infile);
            while ( (i=IS_RESTORE(0,0,0,infile)) != 0)
                {
                printf("\nreenter file name:  ");
                scanf("%s",infile);
                }
            }

        else

            {                           /* if real time image desired,   */
            IS_RESET();                 /* reset frame grabber,          */
                                        /* set sync source to external,  */
            if ( (i = IS_SET_SYNC_SOURCE(1)) != 0)
                {
                IS_END();               /* if there is no sync source,   */
                exit();                 /*     exit program.             */
                }
            IS_PASSTHRU();              /* set display in pass-thru mode.*/
            IS_DISPLAY(1);
            printf("\nWould you like to perform a histogram first?  [n]  ");
            if ( (ch=getche()) == 'y')
                hist(0, 1);             /* call histogram function.      */
            fflush(stdin);
            printf("\nHow many frames would you like to average?  [8]  ");
            gets(input);
            if (input[0] == 0)
                avgnum = 8;
            else
                avgnum = atoi(input);

            if (avgnum < 1)
                avgnum = 1;
            printf("\nAcquiring and averaging %d frame(s) from SLAM.",avgnum);
            acquire(avgnum,0);          /* call acquire funtion to acquire  */
            }                           /* and average j frames to buffer 0.*/
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
        }
    IS_SET_SYNC_SOURCE(0);              /* set sync source to internal.     */

/********************** gather analysis information ***************************/

    printf("\nEnter specimen thickness in micrometers:  ");
    scanf("%d",&thick);

    fflush(stdin);
    printf("\nWhat is the reference medium speed (m/s) ?  [1520]");
    gets(input);
    if (input[0] == 0)
         refspeed = 1520;
    else
         refspeed = atoi(input);
    if (refspeed <= 0)
        refspeed = 1520;

  /* determine which stage is being used (effects speed calculation)       */

    stage = 0;
    printf("\nWhich stage is being used? ('g' for glass, 'w' for water) [g] ");
    if ( (ch = getche()) == 'w')
        stage = 1;


  /* skip value determines whether a full analysis (480 lines) or a reduced */
  /* (and faster) analysis is performed. A skip of k uses every kth line.   */

    while ((skip < 1) || (skip > 20))
         {
         fflush(stdin);
         printf("\nEnter skip value ([1] uses all 480 raster lines):  ");
         gets(input);
         if (input[0] == 0)
              skip = 1;
         else
              skip = atoi(input);
         if ((skip < 1) || (skip > 20))
              printf("\n** skip value must be between 1 and 20. **");
         }

    IS_GET_PIXEL(0,DEFAULTTOP1,0,512,topref1);
    IS_GET_PIXEL(0,DEFAULTBOT1,0,512,botref1);
    IS_GET_PIXEL(0,DEFAULTTOP2,0,512,topref2);
    IS_GET_PIXEL(0,DEFAULTBOT2,0,512,botref2);
    IS_SET_ACTIVE_REGION(0,DEFAULTCOLUMN,480,1);
    IS_GET_REGION(0,startcol);
    IS_SET_FOREGROUND(255);
    IS_SET_GRAPHIC_POSITION(DEFAULTTOP1,0);
    lines[0] = DEFAULTTOP1; lines[1] = 511;
    IS_DRAW_LINES(0,1,lines);
    IS_SET_GRAPHIC_POSITION(DEFAULTBOT1,0);
    lines[0] = DEFAULTBOT1;
    IS_DRAW_LINES(0,1,lines);
    IS_SET_GRAPHIC_POSITION(DEFAULTTOP2,0);
    lines[0] = DEFAULTTOP2;
    IS_DRAW_LINES(0,1,lines);
    IS_SET_GRAPHIC_POSITION(DEFAULTBOT2,0);
    lines[0] = DEFAULTBOT2;
    IS_DRAW_LINES(0,1,lines);
    IS_SET_GRAPHIC_POSITION(0,DEFAULTCOLUMN);
    lines[0] = 479;  lines[1] = DEFAULTCOLUMN;
    IS_DRAW_LINES(0,1,lines);

    printf("\nDo you want to use the default region specifications?  [n]  ");
    ch = getch();

    IS_PUT_REGION(0,startcol);
    IS_PUT_PIXEL(0,DEFAULTTOP1,0,512,topref1);
    IS_PUT_PIXEL(0,DEFAULTBOT1,0,512,botref1);
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
    IS_PUT_PIXEL(0,DEFAULTTOP2,0,512,topref2);
    IS_PUT_PIXEL(0,DEFAULTBOT2,0,512,botref2);


    if ( ch != 'y')
                                /* set reference regions manually */
         {
         printf("\nUse arrow keys to move cursor to desired postion.");
         printf("\nArrow keys on number pad with NUM LOCK on move by 10's");
         printf("\nHit \"enter\" to select postion.");
         printf("\nSelect column where velocity will be calculated.");
         cursor(&j,&xbeg);

         do                         /* continue region setting until        */
                                    /* both selected regions are valid.     */
         {
         flag1 = 0;
         flag2 = 0;
         printf("\nSelect start and end rows for two reference regions; ");
         printf("\nsetting bottom row at value lower than top row disables");
         printf(" region.");
         printf("\nReference region 1:  select top row.  ");
                              /* cursor function returns current position */
         cursor(&ref[0][0],&j);
         printf("\nReference region 1:  select bottom row.");
         cursor(&ref[0][1],&j);
                             /* if bottom row < top row, disable region 1  */
                             /* by setting flag1.                          */
         if (ref[0][0] > ref[0][1])
              flag1 = 1;
         printf("\nReference region 2:  select top row. ");
         cursor(&ref[1][0],&j);
         printf("\nReference region 2:  select bottom row.   ");
         cursor(&ref[1][1],&j);
         if (ref[1][0] > ref[1][1])
              flag2 = 1;        /* if bottom row < top row, disable region 2 */
                                /* by setting flag2.                         */

                                /* if both regions disabled, report error.   */
         if ((flag2 == 1) && (flag1 == 1))
              printf("\n** Can't disable both regions. **");
         }

         while ((flag1 == 1) && (flag2 == 1)); /* condition of do while loop */

         }

    else
                                  /* default region specifications:  */
         {
         xbeg = DEFAULTCOLUMN;    /* starting column.                */
         ref[0][0] = DEFAULTTOP1; /* region 1 start.                 */
         ref[0][1] = DEFAULTBOT1; /* region 1 end.                   */
         ref[1][0] = DEFAULTTOP2; /* region 2 start.                 */
         ref[1][1] = DEFAULTBOT2; /* region 2 end.                   */
         }

    /* obtain information for output data file.                        */

    printf("\nEnter output data file name:  ");
    scanf("%s",filename);
    fflush(stdin);
    printf("\nEnter the name of the specimen:  \n--> ");
    gets(specname);
    printf("\n\n");

/************ calculate DFT summation exponential coefficients *************/

    for (j=LOW; j<=HIGH; j++)          /* only calculate for necessary      */
         {                             /* frequencies.                      */
         for (k=0; k<WINLEN; k++)      /* also only for WINLEN values       */
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
            {                           /* (the other DFTSIZE-WINLEN values */
            x = 2.0*pi*(float)(j*k)/(float)DFTSIZE;
                                        /* are effectively zero-padded).    */
            recoef[j-LOW][k] = cos(x);  /* calculate real part of coef.     */
            imcoef[j-LOW][k] = -sin(x); /* calculate imag. part of coef.    */
            }
        }

/*****************************************************************************/
/* begin analysis:  find frequency with peak DFT magnitude and record phase */
/*****************************************************************************/

    total = 0;                          /* reset raster line total.         */
    for (i=0; i<480; i+=skip)           /* use only every (skip value) line */
        {
        dcoffst = 0.0;                  /* reset dc offset.                 */
        mag[i] = 0.0;                   /* reset current maximum magnitude. */
        if (i%50==0)                    /* print every 50th index as time.  */
        printf("%d ",i);                /* until completion indicator.      */

/* obtain data from frame grabber length WINLEN starting with value at xbeg */
        IS_GET_PIXEL(0,i,xbeg,WINLEN,datbuff);
        for (j=0; j<WINLEN; j++)
            dcoffst += (float)datbuff[j];
        dcoffst /= (float)WINLEN;       /* calculate average (dc) value.    */

/* calculate DFT of current data sequence using precalculated exponentials. */
        for (j=0; j<=HIGH-LOW; j++)
            {
            r = 0.0; z = 0.0;           /* reset DFT summations.            */
            for (k=0; k<WINLEN; k++)
                {                       /* subtract dc value before mult.   */
                x = ( (float)datbuff[k] - dcoffst );
                r += x * recoef[j][k];
                z += x * imcoef[j][k];
                }
     /* find if magnitude ofDFT of current frequency is greater than max.  */
            if ( (tempmag=sqrt(r*r + z*z) ) > mag[i])
                {                           /* if new maximum,              */
                mag[i] = tempmag;               /* change maximum,          */
                phase[i] = -atan2(z,r) + pi;    /* calculate phase,         */
                freq[i] = SAMPLFREQ/DFTSIZE*(j+LOW);  /* find frequency     */
                }
            }
        total = total + 1;          /* increment total and continue on next */
        }                           /* raster line.                         */
    IS_END();                       /* turn off frame grabber driver.       */

/***************** unwrap phase data ***************************************/

    oldphase = phase[0];            /* set old phase to first phase value   */
    phs[0] = phase[0];
    for (i=skip; i<480; i+=skip)
        {
        delta = phase[i] - oldphase;    /* calculate delta.                 */
        oldphase = phase[i];            /* update old phase.                */

   /* unwrap the data: if the difference between current phase and the most */
   /* recent phase value (oldphase) is greater the limit then offset phase  */
   /* by the require amount.                                                */
        if (fabs(delta) < LIMIT)
            phase[i] = phase[i-skip] + delta;
        else
            phase[i] = phase[i-skip] + delta - 2.0*pi*(int)(delta/pi);
        phs[i] = phase[i];
        }
/************ call speed funtion to convert phase to speed ******************/

    speed(ref, thick, &corcoef, skip, refspeed, stage, &reftotal);
/* open file to record data                                                  */
    while ( (outptr=fopen(filename,"w")) == NULL)
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
        {
        printf("\nCan't open file %s, enter new name:  ",filename);
        scanf("%s",filename);
        fflush(stdin);
        }
    fprintf(outptr, "\nTotal number of raster lines used:  %d", total);
    fprintf(outptr, "\nFor specimen:  ");
    fputs(specname,outptr);
    fprintf(outptr, "\nSpeed analysis calculated at column %d", xbeg);
    if (flag1 != 1)
    fprintf(outptr, "\nReference region 1: starting row - %d  ending row - %d",
           ref[0][0],ref[0][1]);
    else
        fprintf(outptr, "\nReference region 1:  DISABLED");
    if (flag2 != 1)
    fprintf(outptr, "\nReference region 2: starting row - %d  ending row - %d",
           ref[1][0],ref[1][1]);
    else
        fprintf(outptr, "\nReference region 2:  DISABLED");
    fprintf(outptr, "\nTotal number of reference lines used:  %d", reftotal);
    fprintf(outptr, "\nCorrelation of best fit reference phase:  %f",corcoef);
    fprintf(outptr, "\nSpecimen thickness:  %d micrometers", thick);
    fprintf(outptr, "  Reference speed:  %d m/s", refspeed);
    fprintf(outptr, "\n\nLine     Frequency      Phase         Speed  \n");
    fprintf(outptr, "           (kHz)        (rad)          (m/s)\n\n");
    for (i=0; i<480; i+=skip)
        fprintf(outptr, "%-4d      %5.1f        %-8.5f     %10.5f\n",
                        i,freq[i],phs[i], phase[i]);
    fclose(outptr);
/*   call velplot funtion to plot the speed data                              */
    velplot(phase,total,skip);

    IS_INITIALIZE();
    IS_SET_ACTIVE_REGION(0,xbeg,480,1);
    IS_GET_REGION(0,startcol);
    IS_SET_GRAPHIC_POSITION(0,xbeg);
    lines[0] = 479;  lines[1] = xbeg;
    IS_DRAW_LINES(0,1,lines);

    printf("\nHit SPACE BAR to toggle between image and plot or ENTER ");
    printf("to continue");
    curbuf = 1;
    while ( (ch=getch()) != 13)
        {
        if (curbuf == 0)
            {
            IS_SELECT_OLUT(7);
            curbuf = 1;
            }
        else
            {
            IS_SELECT_OLUT(0);
            curbuf = 0;
            }
        IS_SELECT_OUTPUT_FRAME(curbuf);
        }
    IS_PUT_REGION(0,startcol);
    IS_END();
    printf("\nDo you want to save this image in a file?  [y]  ");
    if ( (ch=getche()) != 'n')
        {
        printf("\nEnter file name (8 characters + 3 for extension:  ");
        scanf("%s",filename);
        IS_INITIALIZE();
        while ( (i=IS_SAVE(0,0,1,0,filename)) != 0)
            {
            printf("\nreenter file name:  ");
            scanf("%s",filename);
            }
        IS_END();
        }
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
    printf("\nWould you like to perform statistical analysis on the current ");
    printf("speed plot?\n");
    printf("(analysis can only be performed if skip = 1 )  [n] ");
    if ( ( (ch = getche()) == 'y') && (skip == 1) )
         stats(phase, thick, xbeg, specname, filename);

    printf("\nWould you like to continue?  [y]  ");
    repeat = getche();
    skip = 0;                          /* reset skip value  */
    }
}



/*****************************************************************************/
/* Speed  Function                                                           */
/*   This function takes an array of phases and converts it to an array of  */
/*   speeds for each raster line used.  The speeds are calculated using the */
/*   spatial frequency domain algorithm.                                     */
/*****************************************************************************/


speed(int ref[2][2], int thick, float *r, int skip, int refspeed, int stage,
            int *total)
{
    int i, j, k, l;
    float wavelng, theta, sum_x=0., sum_y=0., sum_xx=0., sum_yy=0.,temp,n=0.0,
          sum_xy=0., delta, b, m, fn, v1, v2, v3, v4, pi=4.0*atan(1.0);

/*****************************************************************************/
/*  ref[][] - the array of the start and end points of the reference regions */
/*  thick - thickness of the specimen                                        */
/*  *r - pointer to the correlation coefficient                              */
/*  skip -  the skip value                                                   */
/*  refspeed - the reference speed in meters per second                      */
/*  stage - flag to tell which stage is being used                           */
/*  *total - pointer to the total number of reference lines used             */
/*  wavelng - wavelength of sound in the reference medium                    */
/*  theta - angle from normal of the acoustic beam in the reference medium   */
/*  sum_x - sum of the raster line indices                                   */
/*  sum_y - sum of the phases                                                */
/*  sum_xx - sum of the squares of the raster line indices                   */
/*  sum_yy - sum of of the squares of the phases                             */
/*  sum_xy - sum of the product of the raster line indices and phases        */
/*  delta, fn, v1, v2, v3, v4, temp - temporary storage variables            */
/*  m - slope of the best-fit reference-phase line                           */
/*  b - intercept of the best-fit reference-phase line                       */
/*  n - total number of reference lines used for least squares algorithm     */
/*****************************************************************************/

/* calculate some values required for algorithm                              */

    wavelng = (float)refspeed/SLAMFRQ;

    if (stage == 0)                                    /* glass stage    */
         theta = asin((float)refspeed*0.707107/5968.0);
    else                                               /* water stage    */
         theta = asin((float)refspeed*sin(10.0*pi/180.0)/1509.0);

/*  This section calculates a best-fit reference-phase using a least        */
/*    squares algorithm.                                                     */


/*  using the ref[][] array and the skip value find the correct values      */

    if ((ref[0][0] % skip) != 0)
         j = (ref[0][0] - (ref[0][0] % skip) + skip);
    else
         j = ref[0][0];
    k = ref[0][1];
```

APPENDIX B SPEED PROGRAM LISTING, Continued

```
    for (l=1; l<=2; l++)
         {
         for (i=j; i<=k; i+=skip)
              {
              n += 1.0;
              sum_x += (float)i;
              sum_y += phase[i];
              sum_xx += (float)i * (float)i;
              sum_yy += phase[i] * phase[i];
              sum_xy += (float)i * phase[i];
              }
         if ((ref[1][0] % skip) != 0)
              j = (ref[1][0] - (ref[1][0] % skip) + skip);
         else
              j = ref[1][0];
         k = ref[1][1];
         }

/*  calculate the slope and intercept of the best-fit reference phase line   */

    delta = sum_xx*n - sum_x*sum_x;
    m = (sum_xy*n - sum_x*sum_y)/delta;
    b = (sum_xx*sum_y - sum_x*sum_xy)/delta;
    temp = delta * fabs(sum_yy*n-sum_y*sum_y);

/* calculate the correlation coefficient                                     */

    if (temp > 0.0)
         *r = (sum_xy*n - sum_x*sum_y)/sqrt(temp);
    v1 = thick * sin(theta);
    v2 = v1 * tan(theta);
    v3 = (float)refspeed/sin(theta);

/* once all values and the best-fit line have been calculated, perform      */
/*   conversion of the phase values to speed values for each raster line.   */

    for (i=0; i<480; i+=skip)
         {
         fn = (phase[i] - (m * (float)i + b))/(2.0*pi);
         v4 = fn * wavelng * tan(theta);
         phase[i] = v3 * sin(atan2(v2, (v1 - v4)));
         }
    *total = (int)n;
}
```

APPENDIX C DISPLAY PROGRAM LISTING

```
#include <isdefs.h>
#include <iserrs.h>
main(argc,argv)
int argc;
char *argv[];
{
     IS_INITIALIZE();
     IS_RESET();
     if (strcmp(argv[1],"off") != 0)
     {
          IS_DISPLAY(1);
          if (argv[1] != 0)
               IS_RESTORE(0,0,0,argv[1]);
          else
               {
               IS_SET_SYNC_SOURCE(1);
               IS_PASSTHRU();
               }
     }
     IS_END();
}
```

APPENDIX D GRAB PROGRAM LISTING

```
#include <isdefs.h>
#include <iserrs.h>


/*****************************************************************************/
/*                                                                           */
/* 'grab' <filename> acquires and averages 8 images, and stores the          */
/*        resulting image at the user-specified name from the command line.  */
/*                                                                           */
/*****************************************************************************/


main(argc,argv)
int argc;
char *argv[80];
{
     int i;                                  /***********************************/
     if (argv[1] != 0)                       /* if there are command line       */
          {                                  /* arguments, then initialize      */
          IS_INITIALIZE();                   /* frame grabber.                  */
          IS_RESET();                        /***********************************/
          IS_DISPLAY(1);
          if ( (i = IS_SET_SYNC_SOURCE(1)) != 0)
              {
              IS_END();
              exit();
              }                              /***********************************/
                                             /* acquire and average 8 frames    */
                                             /***********************************/

          printf("\nAcquiring and averaging 8 frames from SLAM.");
          acquire(8,0);
          IS_SET_SYNC_SOURCE(0);
                                             /***********************************/
                                             /* store the resulting image at    */
                                             /* the prespecified name           */
                                             /***********************************/

          while ( (i=IS_SAVE(0,0,1,0,argv[1])) != 0)
              {
              printf("\nreenter file name:  ");
              scanf("%s",argv[1]);
              }
          IS_END();
          }
      else
         printf("\nprogram format:  grab <desired file name>");
}
```

APPENDIX E PLOT PROGRAM LISTING

```
#include <stdio.h>
#include <graphs.h>

/*****************************************************************************/
/*  'plot' uses velplot to plot speed data values output from the program   */
/*         speed.  It automatically finds the number of points to plot      */
/*         If any character is entered on the command line other than the   */
/*         desired file name, the plot is made on the computer monitor.     */
/*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
    float x[480], y[480], temp1, temp2;
    int i, total, offset, temp3, nxdiv, nydiv, npts;
    char line[80],ch1[10],ch2[10],ch3[10],ch4[10],ch5[10],ch6[10];
    FILE *fptr, *outptr;

    if ((fptr=fopen(argv[1],"r")) == NULL)
        {printf("can't open file %s.",argv[1]);exit();}
    fgets(line,80,fptr);
    fscanf(fptr, "%s %s %s %s %s %s %d",ch1,ch2,ch3,ch4,ch5,ch6,&total);
    for (i=0; i<12; i++)
        fgets(line,80,fptr);
    offset = 480/total;

    if (argc == 2)
        {
        for (i=0; i<480; i+=offset)
            fscanf(fptr, "%d %f %f %f", &temp3,&temp1,&temp2,&x[i]);
        }
    else
        {
        for (i=0; i<total; i++ )
            {
            fscanf(fptr, "%d %f %f %f",&temp3,&temp1,&temp2,&y[i]);
            x[i] = (float)temp3;
            }
        }

    fclose(fptr);

/* check if any other character was entered on the command line             */

    if (argc == 2)
        velplot(x,total,offset);          /* if not, plot on image monitor  */

/* otherwise, use GraphiC routine to plot on computer monitor and allow for  */
/* a hard-copy of speed plot.                                                */

    else
        {
         printf("\nSpecifying an output file name of anything but 'notek'");
         printf("\nallows a speed plot to be printed out.\n  For printout,");
         printf(" hit space bar upon plot completion,\n and follow the");
         printf(" instructions given.");
         printf("\n\noutput file name:  ");
         scanf("%s",line);

/*  GraphiC functions for plotting                                           */

         bgnplot(1,'g',line);
         startplot(8);
         font(1,"simplex.fnt",'\310'," ", ' ',"  ",' ',"  ",' ');
         page(9.0, 6.855);
         area2d(8.0, 6.);
         upright(1);
         nxdiv = 12;
         nydiv = 10;
         npts = total;
```

APPENDIX E PLOT PROGRAM LISTING, Continued

```
        scales(nxdiv,nydiv,x,y,npts);
        xname("\310Raster Line Index");
        heading("\310SPEED OF SOUND vs RASTER LINE");
        yname("\310Speed (m/s)");
        curve(x,y,npts,0);
        endplot();
        stopplot();
        }
}
```

APPENDIX F STATS PROGRAM LISTING

```
#include <stdio.h>
#include <math.h>
main(argc,argv)
int argc;
char *argv[];
{
    FILE *fptr,*outfptr;
    float speed[480],temp1,temp2,mean,stdev,sum;
    int i,j,total=480,offset,rline,region=0,leftb,rightb;
    char line[80],ch1[40],ch2[10],ch3[10],ch4[10],ch5[10],ch6[10],
          title_string[80],column_string[80],thickness_string[80];
    if ((fptr=fopen(argv[1],"r")) == NULL)
        {printf("can't open file %s.",argv[1]);exit();}
    printf("output file name (specifying an existing file will ");
    printf("cause current\ndata to be appended to previous data):  ");
    scanf("%s",ch1);
    while ( (outfptr = fopen(ch1, "a")) == NULL)
        {
        printf("\nCan't open file %s.  Enter new file name:  ",ch1);
        scanf("%s",ch1);
        }
    fgets(line,80,fptr);
    fscanf(fptr, "%s %s %s %s %s %d",ch1,ch2,ch3,ch4,ch5,ch6,&total);
    fgets(line,80,fptr);
    fgets(title_string,80,fptr);
    fgets(column_string,80,fptr);
    for (i=0; i<4; i++)
        fgets(line,80,fptr);
    fgets(thickness_string,80,fptr);
    for (i=0; i<4; i++)
        fgets(line,80,fptr);
    if (total != 480)
        {
        printf("\nThis program can only be run on data with all 480 lines.");
        exit();
        }
    for (i=0; i<480; i++)
        fscanf(fptr, "%d %f %f %f", &rline,&temp1,&temp2,&speed[i]);
    fclose(fptr);
    velplot(speed,total,1);
    fprintf(outfptr,"Data from the file: %s\n",argv[1]);
    fputs(title_string,outfptr);
    fputs(column_string,outfptr);
    fputs(thickness_string,outfptr);
    printf("\nHow many regions would you like to specify?  ");
    scanf("%d", &region);
    for (i=1; i<=region; i++)
        {
        printf("\nSelect left boundary of region %d.  ", i);
        index_pointer(&leftb);
        printf("%d",leftb);
        printf("\nSelect right boundary of region %d.  ", i);
        index_pointer(&rightb);
        while (rightb < leftb)
              {
              printf("\nRIGHT BOUNDARY MUST BE GREATER THAN LEFT BOUNDARY.");
              printf("\nSelect right boundary of region %d.  ", i);
              index_pointer(&rightb);
              }
        printf("%d",rightb);
        total = rightb - leftb + 1;
        sum = 0.0;
        for (j=leftb; j<=rightb; j++)
             sum += speed[j];
        mean = sum/(float)total;
        sum = 0.0;
        for (j=leftb; j<=rightb; j++)
             sum += (speed[j] - mean)*(speed[j] - mean);
        stdev = sqrt((double)(sum/(float)total));
        printf("\nmean = %10.5f    std = %8.5f", mean, stdev);
        fprintf(outfptr,
```

APPENDIX F STATS PROGRAM LISTING, Continued

```
            "\nREGION %d:  lines %3d to %3d    mean = %10.5f   std = %8.5f",
              i, leftb,rightb,mean,stdev);
        }
    fprintf(outfptr,"\n\n\n");
    fclose(outfptr);
}
```

APPENDIX G CHANGE WINDOW PROGRAM LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <conio.h>
#include <stdio.h>
main()
{
     int i,j,k,line[8], row, col, height, length, temp1, temp2;
     char ch;
     FILE *fptr;
     fptr = fopen("c:\\slam\\attwin.pos", "r+");
     fscanf(fptr, "%d %d %d %d", &row, &col, &height, &length);
     IS_INITIALIZE();
     IS_RESET();
     IS_SET_SYNC_SOURCE(1);
     IS_SET_ACTIVE_REGION(row,col,height,length);
     IS_DISPLAY(1);
     IS_SET_GRAPHIC_POSITION(row,col);
     IS_SET_FOREGROUND(1);
     for (i=1; i<256; i+=2)
          IS_LOAD_OLUT_SVAL(0,i,255,0,0);
     line[0] = row;
     line[1] = col + length - 1;
     line[2] = row + height - 1;
     line[3] = col + length - 1;
     line[4] = row + height - 1;
     line[5] = col;
     line[6] = row;
     line[7] = col;
     IS_FRAME_CLEAR(0);
     IS_DRAW_LINES(0,4,line);
     IS_LOAD_MASK(1);
     IS_SELECT_ILUT(5);
     IS_PASSTHRU();
     IS_DISPLAY(1);
     printf("\nChange current attenuation measurement region?  ");
     if ( (ch = getche()) == 'y')
         {
         printf("\nMove cursor to desired upper left corner and hit RETURN");
         cursor(&row,&col);
         printf("\next move to desired lower right corner and hit RETURN");
         cursor(&temp1,&temp2);
         height = temp1 - row + 1;
         length = temp2 - col + 1;
         if ((height < 1) || (length < 1))
              {
              printf("\nSelected region not valid");
              fclose(fptr);
              IS_RESET();
              IS_END();
              exit();
              }
         if (height*length > 10000)
              {
              printf("\nSelected region too large");
              fclose(fptr);
              IS_RESET();
              IS_END();
              exit();
              }
         fseek(fptr,(long int)0,0);
         fprintf(fptr,"%3d %3d %3d %3d",row,col,height,length);
         IS_SET_ACTIVE_REGION(row,col,height,length);
         IS_SET_GRAPHIC_POSITION(row,col);
         IS_SET_FOREGROUND(1);
         IS_LOAD_OLUT_SVAL(0,1,255,0,0);
         line[0] = row;
         line[1] = col + length - 1;
         line[2] = row + height - 1;
         line[3] = col + length - 1;
         line[4] = row + height - 1;
         line[5] = col;
```

APPENDIX G CHANGE WINDOW PROGRAM LISTING, Continued

```
            line[6] = row;
            line[7] = col;
            IS_LOAD_MASK(0);
            IS_FRAME_CLEAR(0);
            IS_DRAW_LINES(0,4,line);
            IS_LOAD_MASK(1);
            IS_SELECT_ILUT(5);
            IS_PASSTHRU();
            IS_DISPLAY(1);
            }
        fclose(fptr);
        IS_END();
}
```

APPENDIX H ACQUIRE FUNCTION LISTING

```
#include <conio.h>
#include <stdio.h>
#include <5158io.h>
#define DELAY  16000    /* delay value between grabbing frames                */

/*****************************************************************************/
/*                                                                           */
/* acquire() is a function which replaces the IS_ACQUIRE() subroutine in     */
/* the DT-IRIS subroutine library.  The frame grabber board has a hardware   */
/* sync problem which causes the acquire and average function of IS_ACQUIRE  */
/* to produce a jittery image.  acquire() performs the same acquire and      */
/* average as IS_ACQUIRE but adds a delay in between grabbing frames to      */
/* allow the frame grabber to regain sync.  It requires two inputs:          */
/*          frames - specifies the number of frames to average               */
/*          buffer -  specifies to which buffer the final image is sent      */
/*                                                                           */
/* (this function is very hardware-interaction intensive.  for a detailed    */
/*  understanding of what is being performed, consult the DT2851 manual.)    */
/*                                                                           */
/*****************************************************************************/

acquire(int frames, int buffer)
{
    unsigned int input,output,index=0,j,temp1,temp2,temp3,temp4,buf=0;
    int flag = 0,i;

/*****************************************************************************/
/*                                                                           */
/* input - used for reading data from the I/O registers                      */
/* output - used for writing data to the I/O registers                       */
/* index, j - are loop indecies                                              */
/* temp1, temp2, temp3 - used to store original status register contents     */
/* temp4 - temporary variable                                                */
/* buf - mask used for setting status registers to use correct frame buffer  */
/* flag - is a flag used for checking when frame grabber and/or frame        */
/*           processor are ready                                             */
/* i - a loop counter                                                        */
/*                                                                           */
/*****************************************************************************/

    input = inpw(INCSR1);              /* set the frame grabber to finish    */
    output = input | 0x0008;           /* current function                   */
    outpw(INCSR1,output);
    while (flag == 0)                  /* wait for frame grabber to signal   */
         {                             /* that it is ready                   */
         input = inpw(INCSR1);
         if ((input & 0x0080) == 0)
              flag = 1;
         }

    if (buffer != 0)                   /* set buffer mask to selected buffer */
         buf = 0x0080;

    temp1 = inpw(INCSR1);              /* save current status register values */
    temp2 = inpw(INCSR2);
    temp3 = inpw(OUTCSR);

    outpw(INCSR2,0x0010 | buf);        /* select buffer and video input funct */
    outpw(INCSR1,0x0088);
    flag = 0;
    while (flag == 0)                  /* wait for ready                     */
         {
         input = inpw(INCSR1);
         if ((input & 0x0080) == 0)
              flag = 1;
         }                             /* when complete, one frame has been  */
                                       /* grabbed.  if only 1 frame selected,*/
                                       /* no averaging required--can return  */
                                       /* to main program.                   */
    if (frames > 1 )                   /* Otherwise, must send this frame and */
         {                             /* subsequent frames to frame processor*/
```

APPENDIX H · ACQUIRE FUNCTION LISTING, Continued

```
outpw(STATUS,0x0000);        /* for averaging.                        */
outpw(CONTROL,0x0004);
flag = 0;
while (flag == 0)            /* wait for frame processor ready      */
     {
     input = inpw(STATUS);
     if ((input & 0x0010) == 0)
          flag = 1;
     }

outpw(CONTROL,0x0004);              /* set look-up table 0  with     */
for (index=0; index<256; index++)   /* 1 to 1 values                 */
     {
     outpw(LUT_INDEX,index);
     outpw(LUT_DATA,index);
     }

outpw(CONTROL,0x0024);              /* set look-up table 1 with      */
for (index=0; index<256; index++)   /* values for calculating        */
     {                              /* average.                      */
     outpw(LUT_INDEX,index);
     outpw(LUT_DATA,(unsigned int)(index*frames));
     }

outpw(CONTROL,0x0004);              /* send first frame to frame     */
outpw(X_OFFSET,0x0000);             /* processor                     */
outpw(Y_OFFSET,0x0000);
outpw(INCSR2,0x0060 | buf);
outpw(INCSR1,0x0088);
flag = 0;
while (flag == 0)
     {
     input = inpw(STATUS);
     if ((input & 0x0002) != 0)
          {
          outpw(CONTROL,0x0004);
          flag = 1;
          }
     }

outpw(CONTROL,0x0005);
flag = 0;
while (flag == 0)
     {
     input = inpw(STATUS);
     if ((input & 0x0001) != 0)
          {
          outpw(CONTROL,0x0004);
          flag = 1;
          }
     }

for (i=1; i<frames; i++)            /* acquire and add additional    */
     {                              /* frames to frame processor     */
     flag = 0;                      /* memory until requested total  */
     while (flag == 0)              /* frame value is reached        */
          {
          input = inpw(INCSR1);
          if ((input & 0x0080) == 0)
               flag = 1;
          }
     outpw(INCSR2,0x0010 | buf);
     outpw(INCSR1,0x0080);
     flag = 0;
     for (j=0; j<DELAY; j++) flag=0; /* wait the required delay      */
     outpw(INCSR1,0x0088);
     while (flag == 0)
          {
          input = inpw(INCSR1);
          if ((input & 0x0080) == 0)
               flag = 1;
```

APPENDIX H ACQUIRE FUNCTION LISTING, Continued

```
        }
    flag = 0;
    while (flag == 0)
        {
        input = inpw(STATUS);
        if ((input & 0x0010) == 0)
            flag = 1;
        }
    outpw(CONTROL,0x1204);
    outpw(X_OFFSET,0x0000);
    outpw(Y_OFFSET,0x0000);
    outpw(INCSR2,0x0060 | buf);
    outpw(INCSR1,0x0083);
    flag = 0;
    while (flag == 0)
        {
        input = inpw(STATUS);
        if ((input & 0x0002) != 0)
            {
            outpw(CONTROL,0x1204);
            flag = 1;
            }
        }
    outpw(CONTROL,0x1205);
    flag = 0;
    while (flag == 0)
        {
        input = inpw(STATUS);
        if ((input & 0x0001) != 0)
            {
            outpw(CONTROL,0x1204);
            flag = 1;
            }
        }
    }

/* when all frames have been sent and added in the frame processor, perform */
/* averaging by division of sum using DT2858 successive approximation funct */

    flag = 0;
    while (flag == 0)
        {
        input = inpw(STATUS);
        if ((input & 0x0010) == 0)
            {
            outpw(CONTROL,0x0022);
            flag = 1;
            }
        }
    outpw(X_OFFSET,0x0000);
    outpw(Y_OFFSET,0x0000);
    outpw(CONTROL,0x0023);
    flag = 0;
    while (flag == 0)
        {
        input = inpw(STATUS);
        if ((input & 0x0001) != 0)
            {
            outpw(CONTROL,0x0022);
            flag = 1;
            }
        }


/* after division is completed, send the averaged image back to the selected */
/* frame buffer.                                                             */

    flag = 0;
    while (flag == 0)
        {
        input = inpw(INCSR1);
```

APPENDIX H ACQUIRE FUNCTION LISTING, Continued

```
            if ((input & 0x0080) == 0)
                flag = 1;
            }
        outpw(INCSR1,0x0008);
        outpw(INCSR2,0x0070 | buf);
        flag = 0;
        while (flag == 0)
            {
            input = inpw(STATUS);
            if ((input & 0x0010) == 0)
                flag = 1;
            }
        outpw(CONTROL,0x0008);
        outpw(X_OFFSET,0x0000);
        outpw(Y_OFFSET,0x0000);
        outpw(INCSR1,0x0088);
        outpw(CONTROL,0x0009);
        flag = 0;
        while (flag == 0)
            {
            input = inpw(STATUS);
            if ((input & 0x0001) != 0)
                {
                outpw(CONTROL,0x0008);
                flag = 1;
                }
            }
    }
    outpw(INCSR1,temp1);                /* restore original status      */
    outpw(INCSR2,temp2);                /* register contents and return */
    outpw(OUTCSR,temp3);                /* to main program              */
}
```

APPENDIX I HISTOGRAM FUNCTION LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <graphs.h>
hist(int buffer, int flag)
{
    unsigned long int bin[256], temp, maxvalue, a;
    int i, maxpixel, minpixel, curx, maxvpix, contflag=0;
    float total;
    char ch;
    while (contflag == 0)
         {
         maxvalue = 0;
         curx = 100;
         if (flag == 0)
              contflag = 1;
         IS_HISTOGRAM(buffer, bin);
         if (flag != 0)
              IS_PASSTHRU();
         bin[0] -= 16384;
         for (i=0; i<256; i++)
              {
              if ( bin[i] != 0 )
                   {
                   minpixel = i;
                   i = 256;
                   }
              }
         for (i=255; i>=0; i--)
              {
              if ( bin[i] != 0 )
                   {
                   maxpixel = i;
                   i = -1;
                   }
              }
         total = 0.0;
         for (i=0; i<256; i++)
              {
              total += (float)bin[i]*(float)i;
              if ( bin[i] > maxvalue)
                   {
                   maxvalue = bin[i];
                   maxvpix = i;
                   }
              }
         total /= (512.0*480.0);
         a = maxvalue/320;
         for (i=0; i<256; i++)
              {
              temp = bin[i];
              bin[i] = bin[i]/a;
              if ( (bin[i] == 0) && (temp != 0) )
                   bin[i] = 1;
              }
         _setvideomode(_ERESNOCOLOR);
         for (i=0; i<256; i++)
              {
              if (bin[i] != 0)
                   _rectangle(_GBORDER,curx,330-(int)bin[i],curx+1,330);
              curx += 2;
              }
         _moveto(100,330);
         _lineto(611,330);
         _settextposition(25,0);
         printf("low: %3d  high: %3d  mean: %6.2f  max pixel: %3d",
                 minpixel,maxpixel,total,maxvpix);
         _settextposition(1,0);
         if (flag == 0)
              printf("press any key to continue.");
         else
            printf("press SPACE BAR for another histogram or RETURN to exit.");
```

APPENDIX I HISTOGRAM FUNCTION LISTING, Continued

```
        ch=getch();
        if ( (flag != 0) && (ch == 13) )
            contflag = 1;
    }
    _displaycursor(_GCURSORON);
    _setvideomode(_DEFAULTMODE);
}
```

APPENDIX J SPEED PLOT FUNCTION LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <stdlib.h>
#include <string.h>


/*****************************************************************************/
/*                                                                           */
/*    velplot() is a function that plots the speed data on the image         */
/*    monitor.   It requires the following inputs:                           */
/*        y[] - array of data points to be plotted (480 maximum)             */
/*        count - the total number of points to be plotted                   */
/*        offset - is the skip value between indecies for each data point    */
/*                                                                           */
/*****************************************************************************/


velplot(float d[480], int count, int offset)
{
    int ch[128],i,j,line[960],row,col,max=0,min=5000,b,col1,col2,temp;
    float a,temp2, y[480];
static char cstr[]="0    40   80  120  160  200  240  280  320  360  400  440";
    static char xstr[]="raster line index", vstr[]="Speed   m/s";
    static char hstr[]="SPEED OF SOUND IN SPECIMEN VS RASTER LINE";
    char ystr[40], tstr[10];

/*****************************************************************************/
/*                                                                           */
/* ch[] - temporary array to send characters to be displayed to framegrabber */
/* i, j - loop counters                                                      */
/* line[] - array to send line plotting coordinates to frame-grabber         */
/* row, col - contain current graphic position for frame-grabber             */
/* max - highest value of points to be plotted                               */
/* min - lowest value of points to be plotted                                */
/* b - normalized minimum value                                              */
/* col1, col2, temp - temporary variables                                    */
/* a - scaling factor for plots                                              */
/* temp2 - temporary floating point variable                                 */
/* cstr[], xstr[], hstr[], vstr[] - static character strings for plot labels */
/* ystr[], tstr[] - temporary character string                               */
/*                                                                           */
/*****************************************************************************/


    IS_INITIALIZE();                        /* initialize frame-grabber     */
    IS_FRAME_CLEAR(1);                      /* clear frame buffer 1         */
    IS_SET_SYNC_SOURCE(0);                  /* set sync source to internal  */
    IS_LOAD_OLUT_SVAL(7,0,0,0,115);         /* set colors for:  background  */
    IS_LOAD_OLUT_SVAL(7,255,250,250,250);   /*                  heading     */
    IS_LOAD_OLUT_SVAL(7,254,180,180,190);   /*                  borders     */
    IS_LOAD_OLUT_SVAL(7,253,210,200,200);   /*                  numbers     */
    IS_LOAD_OLUT_SVAL(7,252,255,255,000);   /*                  plot        */
    IS_SET_FOREGROUND(252);
    IS_SELECT_OUTPUT_FRAME(1);
    IS_SELECT_OLUT(7);                      /* select output look-up tbl 7  */

    for (i=0; i<480; i+=offset)
         y[i] = d[i];
    for (i=0; i<480; i+=offset)
         {
         if ((int)y[i] > max) max=(int)y[i]+1;  /* find maximum value       */
         if ((int)y[i] < min) min=(int)y[i];    /* find minimum value       */
         }

    b = min - (min % 10);                   /* set normalized minimum to a  */
                                            /* next lowest value divisible  */
                                            /* by 10.                       */

/* calculate the scaling factor 'a' by dividing the available number of      */
/* lines by the difference between the maximum and normalized minimum        */
```

APPENDIX J SPEED PLOT FUNCTION LISTING, Continued

```
    if ((max - b) > 0)                      /* if max - b > 0, calculate a  */
         a = 400.0/( (float)(max - b) );    /* accordingly                  */
    else
         a = 1.0;                           /* otherwise set a to 1.0       */

    for (i=0; i<480; i+=offset)             /* normalize, scale, and round  */
         {                                  /* all array values.            */
         y[i] = (y[i] - b)*a;
         rnd(&y[i]);
         }

/*  Using rescaled array values, plot the speed values                      */

    col = 30;
    row = 440 - (int)y[0];
    IS_SET_GRAPHIC_POSITION(row,col);
    for (i=0; i<2*(count-1); i+=2)
         {
         line[i] = 440 - (int)y[offset*(i/2+1)];
         line[i+1] = 30 + offset*(i/2+1);
         }
    IS_DRAW_LINES(1,count-1,line);

/* Calculate and plot the borders for the graph                             */

    IS_SET_FOREGROUND(254);
    IS_SET_GRAPHIC_POSITION(440,30);
    for (i=0; i<24; i++)
         {
         line[i*8] = 440;
         line[i*8+1] = 30+(i+1)*20;
         line[i*8+2] = 438;
         line[i*8+3] = 30+(i+1)*20;
         line[i*8+4] = 442;
         line[i*8+5] = 30+(i+1)*20;
         line[i*8+6] = 440;
         line[i*8+7] = 30+(i+1)*20;
         }
    IS_DRAW_LINES(1,96,line);
    IS_SET_GRAPHIC_POSITION(440,30);
    col1 = 30;
    col2 = 2;
    for (j=0; j<2; j++)
         {
         for (i=0; i<8; i++)
              {
              line[i*8] = 440-(i+1)*50;
              line[i*8+1] = col1;
              line[i*8+2] = 440-(i+1)*50;
              line[i*8+3] = col1-2;
              line[i*8+4] = 440-(i+1)*50;
              line[i*8+5] = col1+col2;
              line[i*8+6] = 440-(i+1)*50;
              line[i*8+7] = col1;
              }
         IS_DRAW_LINES(1,32,line);
         IS_SET_GRAPHIC_POSITION(440,510);
         col1 = 510;
         col2 = 1;
         }


/* Calculate the speed values for the y-axis using a and b and convert the  */
/* values to characters.  Send the characters and the static strings to the */
/* frame grabber for plotting.                                              */

    IS_SET_FOREGROUND(253);
    IS_SET_GRAPHIC_POSITION(444,27);
    for (i=0; i<57; i++)
         ch[i] = (int)cstr[i];
    IS_DRAW_TEXT(1,57,ch);
    temp = (b + (int)(100.0/a)*4);
```

APPENDIX J SPEED PLOT FUNCTION LISTING, Continued

```
    itoa(temp,tstr,10);
    strcpy(ystr,tstr);
    for (i=3; i>=0; i--)
         {
         temp = (b + (int)(100.0/a)*i);
         itoa(temp,tstr,10);
         strcat(ystr,tstr);
         }
    for (j=0; j<=4; j++)
         {
         for (i=0; i<4; i++)
              {
              IS_SET_GRAPHIC_POSITION(12+100*j+14*i,19);
              ch[0] = (int)ystr[i+j*4];
              IS_DRAW_TEXT(1,1,ch);
              }
         }
    IS_SET_FOREGROUND(255);
    IS_SET_GRAPHIC_POSITION(464,190);
    for (i=0; i<17; i++)
         ch[i] = (int)xstr[i];
    IS_DRAW_TEXT(1,17,ch);
    for (i=0; i<11; i++)
         {
         IS_SET_GRAPHIC_POSITION(140+i*14,8);
         ch[0] = (int)vstr[i];
         IS_DRAW_TEXT(1,1,ch);
         }
    IS_SET_GRAPHIC_POSITION(10,100);
    for (i=0; i<41; i++)
         ch[i] = (int)hstr[i];
    IS_DRAW_TEXT(1,41,ch);
    IS_DISPLAY(1);
    IS_END();
}

rnd(float *value)
{
    int rvalue,ivalue;
    float temp;
    temp = *value;
    *value = (float)((int)(temp) + (int)(2*(temp - (int)temp)));
}
```

APPENDIX K CURSOR FUNCTION LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <string.h>
#define UP_ARROW 72
#define DOWN_ARROW 80
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define SHIFT_UP 56
#define SHIFT_DOWN 50
#define SHIFT_LEFT 52
#define SHIFT_RIGHT 54
#define ENTER 13
cursor(int *row, int *col)
{
    int currow, curcol, temp[3400], ch[20], i, temp2;
    char key1, key2, cstr[5], rstr[5], fstr[20];
    static char str1[]="ROW: ", str2[]=" COL: ", str3[]="0", str4[]="00";
    IS_SET_ACTIVE_REGION(457,361,22,151);
    IS_GET_REGION(0,temp);
    IS_SET_GRAPHIC_POSITION(462,370);
    IS_SET_FOREGROUND(255);
    IS_SET_BACKGROUND(0);
    IS_SET_MODE(1);
    IS_REPORT_CURSOR_POSITION(&currow, &curcol);
    IS_CURSOR(1);
    printf("\n ROW:  %3d  COL:  %3d\x1B[%dD", currow, curcol, 21);
    temp2 = currow;
    itoa(temp2,rstr,10);
    strcpy(cstr,rstr);
    if (currow < 10)
         {
         strcpy(cstr,str4);
         strcat(cstr,rstr);
         }
    else
         {
         if (currow < 100)
              {
              strcpy(cstr,str3);
              strcat(cstr,rstr);
              }
         }
    strcpy(fstr,str1);
    strcat(fstr,cstr);
    strcat(fstr,str2);
    temp2 = curcol;
    itoa(temp2,rstr,10);
    strcpy(cstr,rstr);
    if (curcol < 10)
         {
         strcpy(cstr,str4);
         strcat(cstr,rstr);
         }
    else
         {
         if (curcol < 100)
              {
              strcpy(cstr,str3);
              strcat(cstr,rstr);
              }
         }
    strcat(fstr,cstr);
    for (i=0; i<18; i++)
         ch[i] = (int)fstr[i];
    IS_DRAW_TEXT(0,18,ch);
    while ( (key1=getch()) != ENTER)
         {
         if (key1 == 0)
              {
              key2 = getch();
              switch( key2 )
```

APPENDIX K CURSOR FUNCTION LISTING, Continued

```
          {
          case UP_ARROW:
               currow -= 1;
               if (currow < 0)
                    currow = 479;
               break;
          case DOWN_ARROW:
               currow += 1;
               if (currow > 479)
                    currow = 0;
               break;
          case LEFT_ARROW:
               curcol -= 1;
               if (curcol < 0)
                    curcol = 511;
               break;
          case RIGHT_ARROW:
               curcol += 1;
               if (curcol > 511)
                    curcol = 0;
               break;
          default:
               break;
          }
     }
else
     {
     switch( key1 )
          {
          case SHIFT_UP:
               currow -= 10;
               if (currow < 0)
                    currow += 480;
               break;
          case SHIFT_DOWN:
               currow += 10;
               if (currow > 479)
                    currow -= 480;
               break;
          case SHIFT_LEFT:
               curcol -= 10;
               if (curcol < 0)
                    curcol += 512;
               break;
          case SHIFT_RIGHT:
               curcol += 10;
               if (curcol > 511)
                    curcol -= 512;
               break;
          default:
               break;
          }
     }
     IS_SET_CURSOR_POSITION(currow, curcol);
     printf(" ROW:  %3d  COL:  %3d\x1B[%dD", currow, curcol, 21);
    temp2 = currow;
    itoa(temp2,rstr,10);
    strcpy(cstr,rstr);
    if (currow < 10)
          {
          strcpy(cstr,str4);
          strcat(cstr,rstr);
          }
    else
          {
          if (currow < 100)
               {
               strcpy(cstr,str3);
               strcat(cstr,rstr);
               }
          }
```

APPENDIX K CURSOR FUNCTION LISTING, Continued

```
            strcpy(fstr,str1);
            strcat(fstr,cstr);
            strcat(fstr,str2);
            temp2 = curcol;
            itoa(temp2,rstr,10);
            strcpy(cstr,rstr);
            if (curcol < 10)
                 {
                 strcpy(cstr,str4);
                 strcat(cstr,rstr);
                 }
            else
                 {
                 if (curcol < 100)
                      {
                      strcpy(cstr,str3);
                      strcat(cstr,rstr);
                      }
                 }
            strcat(fstr,cstr);
            for (i=0; i<18; i++)
                 ch[i] = (int)fstr[i];
            IS_SET_GRAPHIC_POSITION(462,370);
            IS_DRAW_TEXT(0,18,ch);
       }
    IS_CURSOR(0);
    IS_PUT_REGION(0,temp);
    *row = currow;
    *col = curcol;
}
```

APPENDIX L INDEX POINTER FUNCTION LISTING

```
#include <isdefs.h>
#include <iserrs.h>
#include <string.h>
#define LEFT_ARROW 75
#define RIGHT_ARROW 77
#define SHIFT_LEFT 52
#define SHIFT_RIGHT 54
#define ENTER 13
index_pointer(int *col)
{
    int currow, curcol, ch[20], i, temp2;
    char key1, key2, cstr[5], rstr[5], fstr[20];
    static char str1[]="CURRENT INDEX: ", str3[]="0", str4[]="00";
    IS_INITIALIZE();
    IS_SELECT_OUTPUT_FRAME(1);
    IS_SELECT_OLUT(7);
    IS_SET_ACTIVE_REGION(457,361,22,151);
    IS_SET_GRAPHIC_POSITION(462,370);
    IS_LOAD_OLUT_SVAL(7,251,0,0,250);
    IS_SET_FOREGROUND(251);
    IS_SET_BACKGROUND(255);
    IS_SET_MODE(1);
    IS_REPORT_CURSOR_POSITION(&currow,&curcol);
    currow = 479;
    if ( (curcol < 30) || (curcol > 509) )
        curcol = 30;
    IS_SET_CURSOR_POSITION(currow, curcol);
    IS_CURSOR(1);
    temp2 = curcol - 30;
    itoa(temp2,rstr,10);
    strcpy(cstr,rstr);
    if (temp2 < 10)
        {
        strcpy(cstr,str4);
        strcat(cstr,rstr);
        }
    else
        {
        if (temp2 < 100)
            {
            strcpy(cstr,str3);
            strcat(cstr,rstr);
            }
        }
    strcpy(fstr,str1);
    strcat(fstr,cstr);
    for (i=0; i<18; i++)
        ch[i] = (int)fstr[i];
    IS_DRAW_TEXT(1,18,ch);
    while ( (key1=getch()) != ENTER)
        {
        if (key1 == 0)
            {
            key2 = getch();
            switch( key2 )
                {
                case LEFT_ARROW:
                    curcol -= 1;
                    if (curcol < 30)
                        curcol = 509;
                    break;
                case RIGHT_ARROW:
                    curcol += 1;
                    if (curcol > 509)
                        curcol = 30;
                    break;
                default:
                    break;
                }
            }
        else
```

APPENDIX L INDEX POINTER FUNCTION LISTING

```
            {
            switch( key1 )
                 {
                 case SHIFT_LEFT:
                      curcol -= 10;
                      if (curcol < 30)
                           curcol += 480;
                      break;
                 case SHIFT_RIGHT:
                      curcol += 10;
                      if (curcol > 509)
                           curcol -= 480;
                      break;
                 default:
                      break;
                 }
            }
            IS_SET_CURSOR_POSITION(currow, curcol);
           temp2 = curcol - 30;
           itoa(temp2,rstr,10);
           strcpy(cstr,rstr);
           if (temp2 < 10)
                {
                strcpy(cstr,str4);
                strcat(cstr,rstr);
                }
           else
                {
                if (temp2 < 100)
                     {
                     strcpy(cstr,str3);
                     strcat(cstr,rstr);
                     }
                }
           strcpy(fstr,str1);
           strcat(fstr,cstr);
           for (i=0; i<18; i++)
                ch[i] = (int)fstr[i];
           IS_SET_GRAPHIC_POSITION(462,370);
           IS_DRAW_TEXT(1,18,ch);
      }
    IS_CURSOR(0);
    IS_SET_CONSTANT(1,0);
    IS_END();
    *col = curcol - 30;
}
```

APPENDIX M STATS FUNCTION LISTING

```
#include <math.h>
#include <stdio.h>
stats(float speed[480], int thick, int col, char title[], char filename[])
{
    FILE *outfptr;
    float mean,stdev,sum;
    int i,j,total=480,offset,rline,region=0,leftb,rightb;
    char line[80];
    printf("\n\noutput file name (specifying an existing file will ");
    printf(" cause current data \nto be appended to previous data):  ");
    scanf("%s",line);
    while ( (outfptr = fopen(line, "a")) == NULL)
         {
         printf("\nCan't open file %s.  Enter new file name:  ",line);
         scanf("%s",line);
         }
    fprintf(outfptr,"Data from the file: %s\n", filename);
    fputs(title, outfptr);
    fprintf(outfptr, "\nSpeed analysis calculated at column %d\n", col);
    fprintf(outfptr, "Specimen thickness:  %d micrometers\n", thick);
    printf("\nHow many regions would you like to specify?  ");
    scanf("%d", &region);
    for (i=1; i<=region; i++)
         {
         printf("\nSelect left boundary of region %d.  ", i);
         index_pointer(&leftb);
         printf("%d",leftb);
         printf("\nSelect right boundary of region %d.  ", i);
         index_pointer(&rightb);
         while (rightb < leftb)
               {
               printf("\nRIGHT BOUNDARY MUST BE GREATER THAN LEFT BOUNDARY.");
               printf("\nSelect right boundary of region %d.  ", i);
               index_pointer(&rightb);
               }
         printf("%d",rightb);
         total = rightb - leftb + 1;
         sum = 0.0;
         for (j=leftb; j<=rightb; j++)
              sum += speed[j];
         mean = sum/(float)total;
         sum = 0.0;
         for (j=leftb; j<=rightb; j++)
              sum += (speed[j] - mean)*(speed[j] - mean);
         stdev = sqrt((double)(sum/(float)total));
         printf("\nREGION %d: lines %3d to %3d  mean = %10.5f std = %8.5f\n",
                i, leftb, rightb, mean, stdev);
         fprintf(outfptr,
              "\nREGION %d:  lines %3d to %3d    mean = %10.5f   std = %8.5f",
               i, leftb, rightb, mean, stdev);
         }
    fprintf(outfptr,"\n\n\n");
    printf("\n\n");
    fclose(outfptr);
}
```