An Acoustic Scatter-Mapping Imaging System

by

Garfield Richard Mellema

BSE, Calvin College, 1986

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department of Electrical Engineering

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

February 1990

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date February 9, 1990

ABSTRACT

The development of improved models of seismic diffraction is assisted by the availability of accurate scattering data. An acoustic scatter-mapping system was developed for the purpose of providing such data rapidly and at low cost.

This system uses a source-receiver pair suspended on a trolley over the structure to be mapped. Signal generation, acquisition, processing, and plotting are performed on an AT-compatible microcomputer and a laser printer. The entire process can be performed in an automated manner within five hours, generating scatter-mapping plots in a format familiar to the geophysical industry.

The system hardware was similar to those of Hilterman [1] and others referenced by him, but used a controlled source transducer. The available processing power of a microcomputer allowed the use of a 1 to 15 KHz swept-frequency source signal, similar to that used in *Vibroseis* and Chirp Radar, which is later crosscorrelated with received signal to provide precise scatter-mapping data for the target structure. Several examples of theoretical and experimental acoustic scatter-mappings are provided for comparison.

The novelty of this system lies in its use of a swept frequency source signal. While common in the fields of seismology and radar, swept frequency source signals are new to the area of acoustic scatter mapping. When compared to a similar system using a pulsed source signal, this system produces a better controlled source signal of greater energy, resulting in a more useful resultant signal and better mapping characteristics. The system was able to map scattering from features in the target structure smaller than one percent of the crosscorrelated source signal's 37 mm dominant wavelength.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENTS

The author would like to thank his supervisor, E.V. Jull, and his co-supervisors, M.J. Yedlin and B.B. Narod, for their guidance and suggestions. Thanks is also due to P.D. Lawrence for his encouragement. He would also like to acknowledge the assistance rendered by his colleagues, including David Aldridge, David Dalton, and Steven Dods.

This work was funded under a Natural Sciences and Engineering Research Council strategic grant.

1 INTRODUCTION

1.1 Acoustic Imaging of Scattering

The scattering of acoustic pulses has long been used in seismology to determine the characteristics of buried structures. As shown in Fig. 1.1, a known source signal is introduced into the earth at a source point, S, and signals resulting from the interaction of the source signal and the underground structures are received at the receiving point, R, recorded, and plotted. Examination of the signals received at a series of receiver locations yields information as to the location and nature of those objects from which the source signal was scattered. A source signal impinging on a smooth surface will be reflected from that surface at an angle equal but opposite to the angle at which it arrived at that surface. A somewhat different form of scattering occurs when the source signal impinges on a non-smooth surface. A source signal impinging on a non-smooth surface, or one with discontinuities, results in a signal which appears to have originated from that surface. This signal, containing both reflection and diffraction components, scatters in all directions from the surface, its amplitude and phase varying with its angular direction. This complex form of scattering, called diffraction, is a topic of great interest in the areas of seismology and electromagnetics.

Exact solutions for the diffractive scattering of an impinging wave from a non-smooth surface are sufficiently complex and time-consuming to determine that they have been determined for only a limited number of structures. Diffractive scattering from other structures is typically estimated using one of several simplified models of diffraction. In order to provide experimental data for comparison with theoretical data generated using these models, acoustic scatter-mapping systems have been constructed [1]-[5].

One of the early systems, constructed by Angona [2], consisted of several joined sections of solid materials into which pressure pulses were transmitted at regular time intervals. The

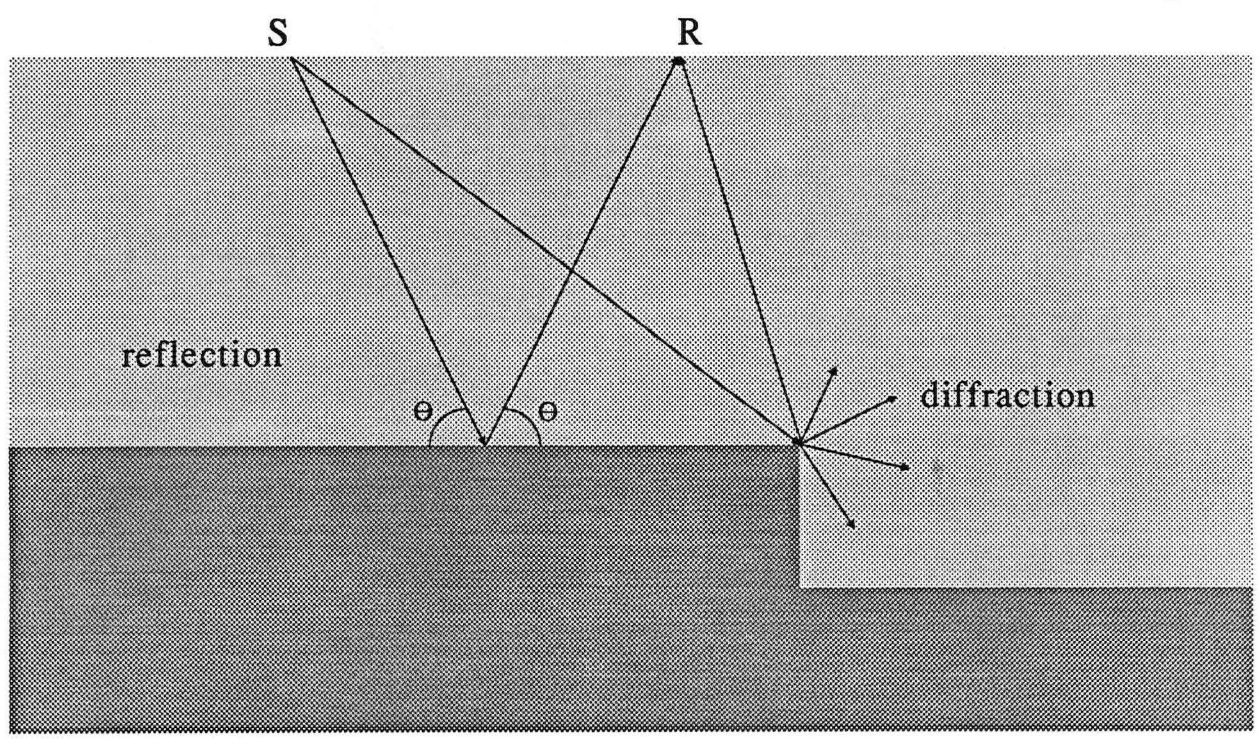

Fig. 1.1 Typical Seismic Reflection and Diffraction

pressure pulses received at a receiver location were translated into electrical signals and photographed on an oscilloscope. Scattering, occurring at each boundary between adjacent sections of dissimilar materials, within known structures could in this way be observed and measured for comparison with calculated or geophysical scattering. This technique, while useful for the acquisition of data in the case of scattering from multiple layers, such as may be found in real geophysical scattering, required substantial skill and resources in the construction and examination of models under study.

An improved seismic modeling apparatus was constructed by Hilterman [1]. This system used air as the medium into which single-layer models were placed. A fixed spark gap source and microphone were used to generate and receive acoustic pulses, and the models under study were placed on a movable platform. Received signals were photographed on an oscilloscope. Data from this system also showed good agreement with calculated experimental results, but was limited in its accuracy.

The system described here was similar to that constructed by Hilterman but took advantage of recent improvements in data acquisition and processing. The major difference between this system and those previous was the application of a crosscorrelated swept-frequency source signal, similar to those found in seismology and radar. A swept-frequency source signal is emitted into air from a loudspeaker source and is scattered from a target structure below. Scattered copies of the source signal, shifted in time and possibly in phase, are received at a microphone and digitized. The digitized received signal is then crosscorrelated with a copy of the source signal to determine the time and phase shift of any copies of the source signal contained within them. The effect of this crosscorrelation is to replace copies of the source signal in the received data with autocorrelations of the source signal, without altering the time or phase information in the data. By careful selection of the source signal characteristics it was possible to generate autocorrelated source signal waveforms of more desirable parameters than those of the traditional pulsed source signal. The use of this improved source signal resulted in greatly improved accuracy and resolution.

1.2 The Purpose of This Acoustic Imaging System

Although exact time-domain solutions are calculable for both high and low-frequency diffraction for a few structures, their development, in general, is typically both complex and time-consuming. For this reason, various simplified models have been used to estimate the diffraction of a wave impinging on a given structure, especially a complex structure or one with surfaces smaller than or equal to the wavelength of the impinging wave. One of the objectives of the Seismic Pulse-Diffraction Project, of which this system was a part, was to improve and further develop such models.

One of the preferred ways to demonstrate the validity of a model is to compare data generated by the model with experimental data. A modeling facility was thus needed to collect real scattering data from scale models similar to those modeled theoretically. This acoustic scatter-mapping facility was constructed in response to the requirements of the larger diffraction research project with the intent of using it to collect scattering data for comparison with that generated by synthetic diffraction models.

1.3 Previous Work on This System

The initial work on this acoustic imaging system was carried out by A. Adler and B. B. Narod between November 1988 and March 1988 [6], [7]. The core of the system consisted of an RC Electronics data-acquisition card installed in an XT- compatible microcomputer. The data-acquisition card was used to digitally generate a pulsed-source signal, which was sent to a loudspeaker, and to digitize an analogue signal received from a microphone. The speaker and microphone were suspended on a trolley over the region in which models were placed. The trolley was moved by a synchronous motor under the control of the computer via the data acquisition card. Both the outgoing source pulse and the returning signal were sampled at position intervals of 2 mm across the region containing the model, until data at a thousand positions had been collected. Data at each position was acquired sixteen times to increase the resolution of the received data from twelve to sixteen bits. After the entire scatter-mapping had been acquired it was plotted on a 72 dpi dot matrix printer, using a grey scale technique, in a standard seismic profile format. The data-acquisition and plotting software were written, especially for the project, in FORTH.

1.4 Thesis Outline

The previously described acoustic scatter-mapping system was complete in itself yet offered many opportunities for improvement. The pulsed source, as produced by the loudspeaker, had an annoying tendency to ring. This was an area which demanded attention since the acoustic source signal eventually became the indicator of the location, amplitude, and phase of scattering from the target structure. Increasing the precision of the acoustic source signal directly increases the precision of the overall system results. This relationship becomes strikingly clear when one compares the improvement in resolution between the images of similar targets at different stages of the system's development. Therefore, one of the criteria for evaluating improvement of the acoustic imaging system was the improvement of the acoustic source signal.

The format of this thesis was selected to reflect the importance of the source signal. However, it is not useful to discuss the source signal out of the context of the overall acoustic-imaging system. For this reason, those aspects of the system less directly related to the system signal path, such as mechanical aspects of the overhead trolley, are also discussed, but in less depth. These aspects of the research occupied a significant portion of our time.

Chapter 1 is intended to provide an overview of the context of this system in general, by discussing it in a historical and logistical perspective. Having shown the general context of the system, the next step is to discuss specific aspects of the system. The path of the source signal, from its calculation to its incorporation in a finished data set, is discussed in Chapter 2. This portion is worthy of at least a chapter in its own right as this path forms the heart of the system and was the site of most of the system's improvements. System components of interest, but not directly related to the signal path, such as the plotting of data and repositioning of the transducers, are discussed in Chapter 3.

Once the reader has been provided an insight into the acoustic imaging system, the development of an improved source signal may be described. This task was undertaken in Chapter 4, followed in Chapter 5 by several examples of scatter-mappings acquired from simple models. Chapter 6 discusses the relevance of the project and suggests some avenues for future improvements.

2 THE SYSTEM SIGNAL PATH

2.1 The System Signal Path

This acoustic scatter-mapping system relied on the emission and reception of a signal used to gather the information that this system was constructed to acquire. As this signal and its derivatives formed the core of the system, we will begin discussion of the system with a discussion of the factors influencing the source signal. The following sections are ordered according to the location of their topic in the source signal path shown in Fig. 2.1.

2.2 Source Signal Calculation

While recent analog signal generation, processing, and storage techniques are good, the use of digital techniques to perform similar tasks has greatly improved the ease, flexibility, and accuracy with which such operations can be performed. Our generation of a desired source signal is a case in point. Analog techniques typically require the variation of control voltages to amplifiers and voltage controlled oscillators. Digital techniques require amplitude information at a sequence of points along the desired source waveform. Below certain frequency limits, the generation of a signal requires no more data than that required to describe an accurate amplitude versus time plot of the signal. This acoustic imaging system made use of fully digital techniques for signal calculation, generation, acquisition, processing, and storage.

In order to generate a specific source signal using digital techniques, it was necessary to describe the signal as a time-series of signal amplitudes. This task was handled by a program called *Wavegen* which used embedded parameters to calculate a time-series of amplitudes to be sent to a digital-to-analog convertor (DAC). Knowledge of the specific DAC conversion rate and range to be used was required to select the correct time interval and amplitude range of the calculated signals. The system had an amplitude range from 0 to 4095, corresponding to an output range of -5 to +5 V, and a minimum time step interval of 3 us. The range employed was between -2.5 and +2.5 V, so as to prevent overloading the input of the power amplifier following

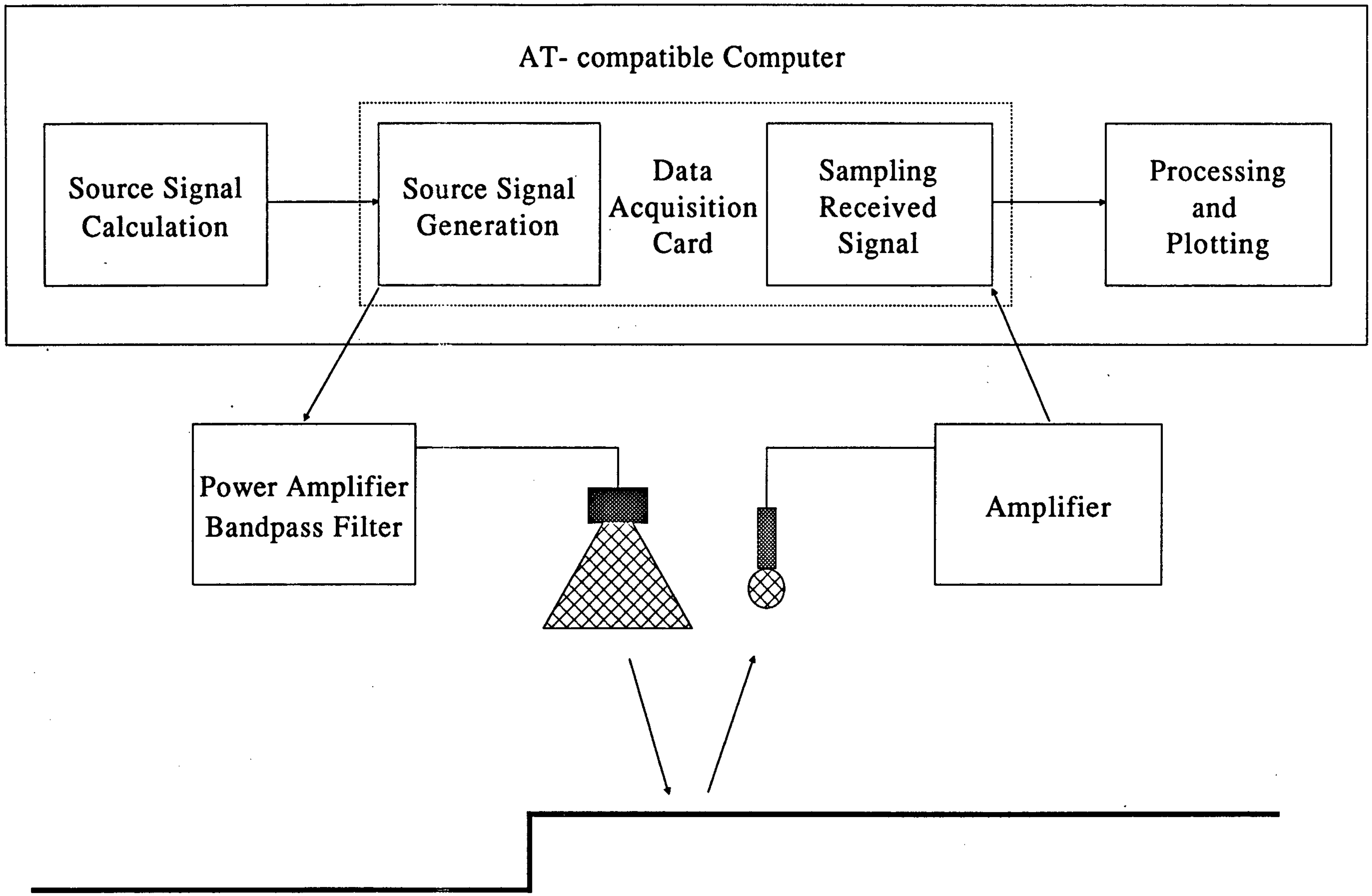

Fig. 2.1 The System Signal Path

the DAC. The minimum time step interval was achieved using an assembly language loop to copy values from an array in computer memory to the DAC at a computer port. The use of a 3 us sample rate gave a Nyquist frequency of 166 KHz, well above the range of any signal we wished to reproduce.

The Nyquist frequency is the highest frequency which can be fully determined by a time-series of points at a given rate. Shannon's theorem states that a signal can be fully determined by a time-series of points at a rate twice the frequency of the highest frequency signal component [8]. Each time that an analog signal was represented digitally in this system it was oversampled, i.e. sampled at a rate greater than that required by Shannon's theorem. In the first case, that of a calculated signal being sent to the DAC, the signal was oversampled to ensure that the Nyquist frequency was well above the upper frequency limit of the power amplifier used to bandpass filter the source signal prior to passing it to the loudspeaker. If the Nyquist frequency of the calculated source signal had been below the upper frequency limit of the power amplifier, aliasing could have occurred, resulting in undesired signal components at frequencies between the Nyquist frequency and the upper frequency limit of the power amplifier. The higher sampling rate presented no difficulties with respect to the computer apparatus. The second case, that of oversampling the signal received by the microphone, was done to ensure sufficient data points for accurate plotting of the finished data.

2.3 Source Signal Generation

Most of the tasks required to collect an acoustic image of a target structure were controlled by a single program called *Multrace*. This program, listed in Appendix A, controlled the generation of the source signal, the acquisition of the information signal, the crosscorrelation of the information and reference signals, and the repositioning of the transducers after gathering data, at each of the 125 scatter-mapping positions. We will now examine the way in which *Multrace* controlled the output of the source signal.

Once calculated, the source signal was stored in a file called *Inwave*. This file was read by *Multrace* into an array at the start of the scatter-mapping operation and passed to an assembly language function, named *Ecko*, whenever generation of the source signal was required. The function, upon receiving the size of the array and a pointer to the beginning of the array, triggered acquisition of the information signal and copied the array containing the calculated source signal to a DAC. The DAC used by this system was a component of an R.C. Electronics ComputerScope data acquisition card. The DAC in this configuration had no decoded parameters, other than its address. At the highest conversion rate, 330 KHz, the output of the DAC appeared as a series of discrete voltage levels at 3 us intervals and had, therefore, frequency components well in excess of the range of most audio equipment, 20 Hz to 20 KHz. This signal was intended as a reference only and lacked sufficient power to drive any sizeable load.

Following the conversion from digital to analog by the DAC, the source signal was passed to an audio power amplifier, an MEI PA300H. This amplifier had two main functions. The first function was that of amplifying the signal passed to it so that it could be passed to the loudspeaker. Voltage gain occurred at this point. The output impedance of the amplifier was also much lower, 4 Ω, than that of the DAC, 50 Ω. Note that current gain also occurred at this point, as the input impedance of the loudspeaker driver, 16 Ω, was much less than that of the power amplifier, 20 KΩ.

The second function of the power amplifier section was low-pass filtering of the source signal. In order to correctly generate the calculated source waveform, only those frequency components which were intended to be in the waveform could be passed. As discussed in the previous section, with a high digital to analog conversion rate the undesired frequency components were well out of the 20 Hz to 20 KHz range of the audio power amplifier.

2.4 **Translating Between Electrical and Acoustic Signals**

The translation of signals from electrical to acoustic and vice-versa was accomplished using loudspeakers and microphones. These two devices share many characteristics, and for that reason are discussed concurrently. Those characteristics unique to microphones or loudspeakers, or unique to either signal translation will be mentioned separately. The loudspeaker, a JBL

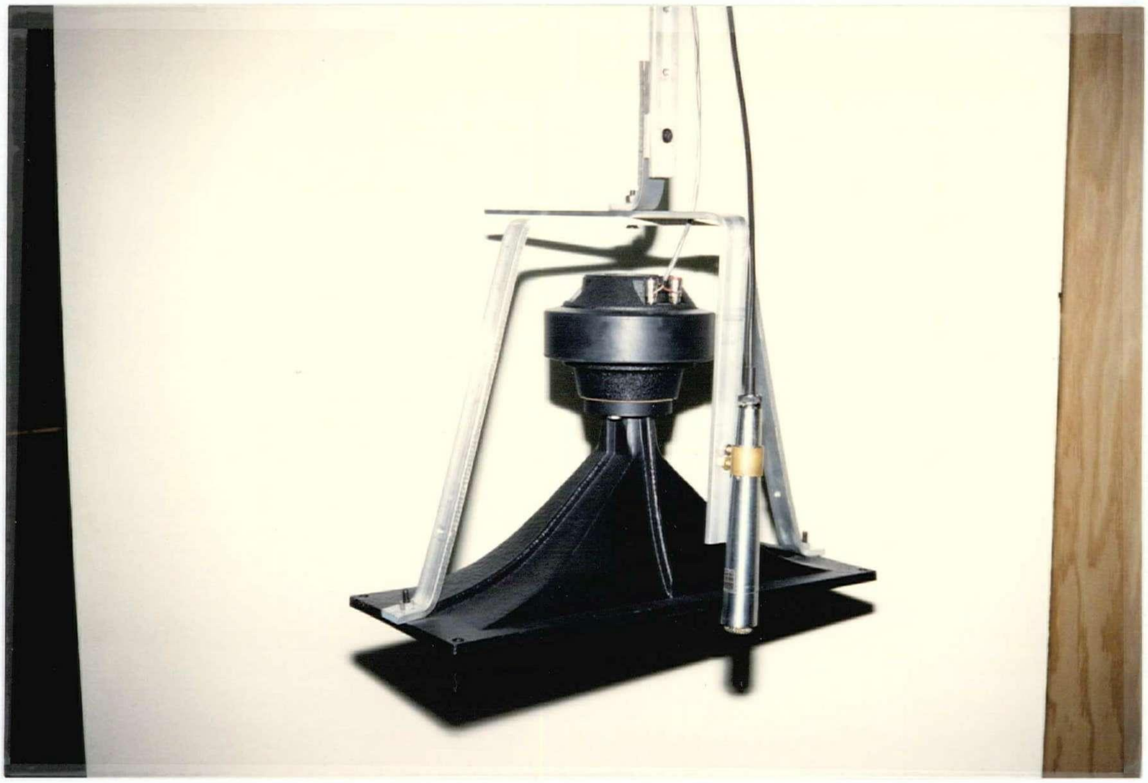

Fig. 2.2 Photograph of the Loudspeaker and Microphone

model 2425J driver attached to a JBL model 2370A horn, and the microphone, an Altec Lansing (now University Sound) model 681A, is shown in Fig. 2.2.

Let us assume that one had an electrical source signal in the audio frequency range, i.e. between 20 Hz and 20 KHz, and that one wished to translate this signal into an acoustic signal. Because of the large frequency range of interest, and the inherent complexity of these transducers, we should consider several aspects before simply sending the electrical signal to a loudspeaker. The first characteristic of concern would be the frequency response of the speaker. The response of a speaker is normally mapped as Sound Pressure Level (SPL) in dB versus frequency in hertz. The SPL is a measure of the air pressure increase, in dB referenced to 20 uPa, at a stated electrical power input, and distance from the speaker, often one watt at one meter for a loudspeaker. A similar measure applies for microphones. The frequency response of a loudspeaker is usually flat within several dB over most of its frequency range and usually ends with fairly sharp roll-offs. This variation of response with frequency indicates that most loudspeakers do not translate single frequency tones uniformly over the audio frequency range. The repeated translation of particular individual tones is, however, quite reliable. The SPL versus frequency responses, from the manufacturer's data, of the loudspeaker and microphone used in this system were shown in Figs. 2.3 and 2.4, respectively.

The second important characteristic, also strongly frequency dependent, would be the electrical impedance of the loudspeaker. This parameter is especially important for loudspeakers. As with the SPL of the speaker, this aspect takes into consideration the acoustic impedance of the environment into which the acoustic signal will be travelling. That is, the impedance of a speaker driver attached, via a well-matched horn, to free space will have a different SPL and impedance-versus-frequency response, compared to the same driver attached to a terminated plane-wave tube. This variation of impedance could affect the output level of the device driving the loudspeaker and cause further variation of the SPL when generating single frequency tones. The impedance versus frequency response of the loudspeaker of this system was shown in Fig. 2.5.

The third important characteristic of both loudspeakers and microphones would be their polar response. This parameter, usually measured at several frequencies, describes the relationship between the SPL of a signal and the location of an observer with respect to the front of the loudspeaker horn. An emitted signal usually has greatest amplitude directly in front of the horn, the amplitude decreasing as the angle between the observer and the horn face increased. The distance between the observer and the horn must remain fixed for this measurement. The polar response may be described in a number of ways, including off-axis response, or contours of constant sound pressure . The off-axis response of the loudspeaker is shown in Fig. 2.6 and the polar response of the microphone is shown in Fig. 2.7.

We have seen that some limitations apply to the ability of a loudspeaker to translate single-frequency electrical signals into acoustic signals. Further restrictions apply to the translation of frequency-varying electrical signals into acoustic signals, with most of the restrictions due to the inertial aspect of air and other physical components. Having finite mass, a loudspeaker driver's voice coil and diaphragm, as well as the air in the loudspeaker horn, tend to

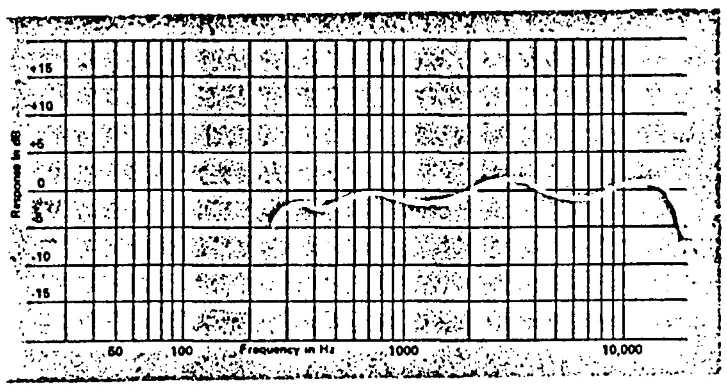

Fig. 2.3 Sound Pressure Level versus Frequency Response for the Loudspeaker

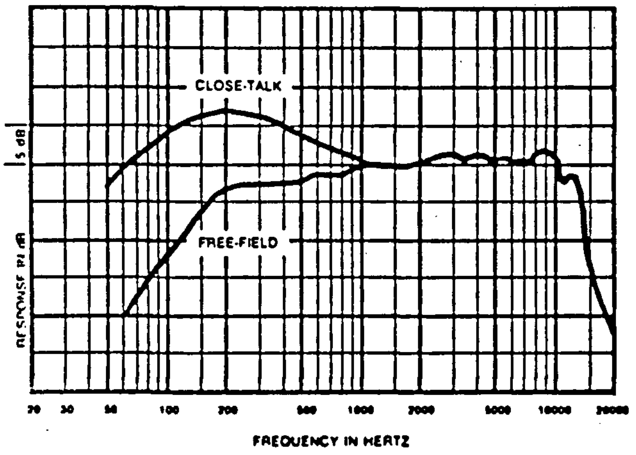

Fig. 2.4 Sound Pressure Level versus Frequency Response for the Microphone

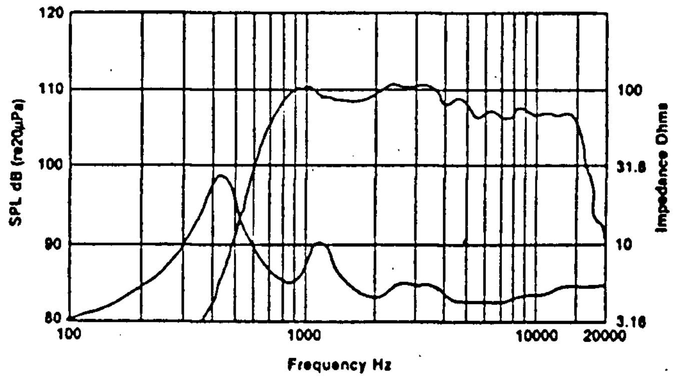

Fig. 2.5 Impedance versus Frequency Response for the Loudspeaker

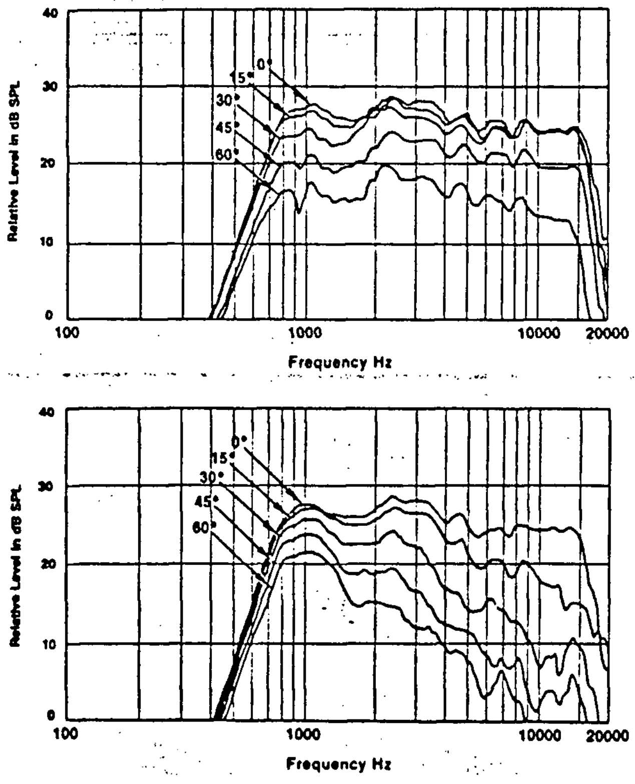

Fig. 2.6 Horizontal and Vertical Off-Axis Response for the Loudspeaker

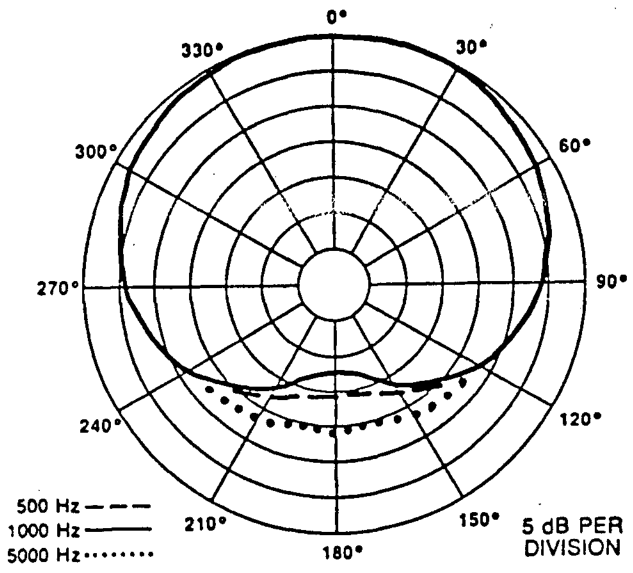

Fig. 2.7 Polar Response Pattern for the Microphone

continue moving in the same pattern in which they were previously moving. This means that the electrical input level required by a loudspeaker to produce a desired SPL is dependent, not only on the frequency of the desired tone, but on the conditions existing in the loudspeaker at the time the electrical signal is applied as well. This characteristic is often referred to as the attack of the loudspeaker. A similar characteristic, the decay or damping factor, of driver voice coil, diaphragm, and the surrounding air, describes the amount of energy absorbed by these components. The influence of these characteristics on the translation of single and variable frequency electrical signals into acoustic signals in this system was significant, as shall be discussed in Section 4.2.

2.5 Characteristics of an Acoustic Signal in Air

While the medium in which the target structure had been immersed, i.e. air, was not ideal, its characteristics can be idealized, within certain constraints, without significant error. Let us consider two aspects of acoustic transmission through air: signal velocity and attenuation.

When an acoustic signal passes through air, its velocity varies as

$$v_s = \sqrt{\frac{\gamma p}{\rho}} \tag{2.1}$$

where v_s is the velocity of the signal, γ is the ratio of specific heats of air (1.402), p is the air pressure in kilopascals, and ρ is the density of air, which varies as

$$\rho = \frac{1.276 \times 10^{-5} p}{1 + 0.00367 T} \tag{2.2}$$

where T is the temperature in degrees Celsius. Substituting equation 2.2 into equation 2.1 and reducing, we find that

$$v_s = 331 \sqrt{1 + 0.00367 T} \tag{2.3}$$

The velocity of an acoustic signal in air is thus a function of the air temperature. For the situation with which we were concerned, that of an acoustic signal travelling a short distance through still air, we see that the velocity of the signal will vary about one sixth percent per degree Celsius temperature change at room temperature (20°C). Since the images have been acquired at room temperature, variations due to temperature have not been a problem. Although variations in acoustic velocity during data acquisition could have resulted in incorrect distance measurements, observed variations have not been significant. This is primarily due to the location of the target region, within a closed room and bounded by three walls.

As an acoustic signal from a pseudo point source travels through air, its energy is spread over a surface with an area increasing with the square of the distance from the source. It was, therefore, not unexpected that the strength of a signal reflected or diffracted from a surface would decrease as the surface became further from the source and receiver. Variations in reflector distance resulted in signal amplitude variations at the microphone. These variations were carried through the remainder of the system and resulted in signal amplitude variations in the finished plots. These variations were minor, due to the short distances involved. Of greater concern, because of their variation with frequency, was attenuation of the acoustic signal due to characteristics of air such as viscosity, heat conduction, and molecular translation. The effect of this attenuation in the received signal due to absorption of sound in air, shown in Fig. 2.8, varied from approximately 0.2 dB/m at 15 KHz to 0.001 dB/m at 1 KHz [10]. This level of absorption, in comparison to the geometric dispersion of the acoustic signal, was considered relevant, but not sufficiently important to warrant further investigation at the time the system was constructed. Analysis of crosscorrelation responses from near (50 cm) and far (2.0 m) reflectors did not reveal any significant variation in the shape of the autocorrelation wavelet. Had disruptive frequency dependent attenuation been present, the crosscorrelation wavelet would have varied significantly from the autocorrelation wavelet of the reference signal.

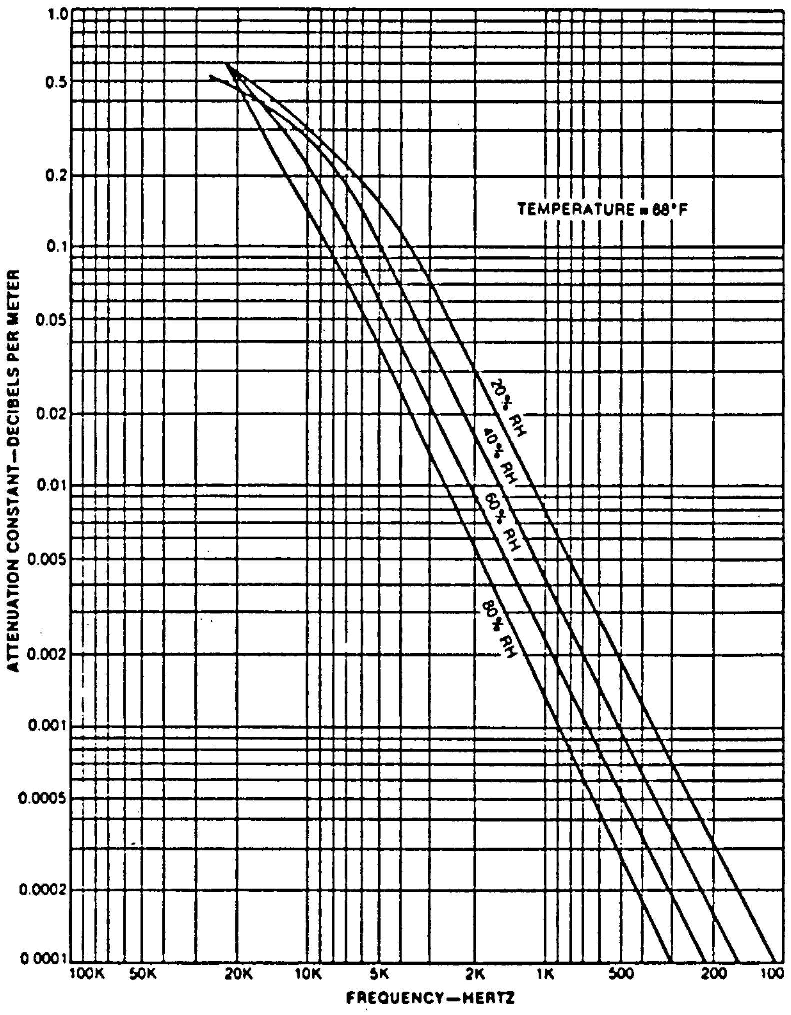

Fig. 2.8 Absorption of Sound in Air [9]

2.6 Scattering Due to Structures in the Target Region

After the source signal was emitted by the loudspeaker, it travelled into the target region until it encountered a scattering structure. The mechanism of this scattering is described below.

When the source signal impinges on a smooth surface, it will be reflected from that surface at an angle equal to the angle at which it struck the surface and with a slightly reduced amplitude, due to absorption by the surface material. Only those surfaces perpendicular to the path of the signal from the loudspeaker to the surface could, therefore, cause a reflected signal to appear in the finished data set.

When the source signal strikes a surface which is not smooth, it will be diffracted from that surface, at all angles and with varying coefficients, back into the air. Although solutions for diffraction from most given structures is sufficiently complex to make the calculation of an exact solution a formidable task, with a few approximations estimated solutions may be generated. The purpose of this acoustic scatter-mapping system, after all, is the generation of experimental data for comparison with numerical solutions. For the case of a zero-offset transducer pair in an acoustic medium of constant velocity and low reflectivity, such as that found in the acoustic scatter-mapping system, Trorey [11] provides the following approximation using a scalar potential to represent the acoustic wave field scattered from a surface θ

$$4\pi\varphi = \frac{2\pi R f(p)\, e^{-2pz/v}}{z} - zRf(p)\int_{\theta} \frac{e^{-2p\xi/v}}{\xi^2}\, d\theta \qquad (2.4)$$

where φ is the potential at a point P, *R* is the reflection coefficient of the surface, *f(p)* is the Laplace transform of the source waveform associated with the source point, *p* is the Laplace transform variable, *v* is the velocity of the medium, and *z* and ξ are the distance from the source-receiver pair to the plane of the surface and the edge of the surface under consideration, respectively. The first term in equation 2.4 is the reflection response of the surface under consideration and the second term is the diffraction response. Notice that the effect of reflection or diffraction on the source can be described as a multiplication of the Laplace Transform of the source signal with a reflection or diffraction coefficient independent of the characteristics of the source signal.

In a practical sense, equation 2.4 means that a source-receiver pair moving past an edge will observe a diffraction response whose amplitude and phase vary with the angle between the source receiver pair and the diffracting edge. The returning signal is diffracted from the edge as if it had originated there, with only minor variation due to the angle of source signal incidence. This variation, however, is important as the diffraction response must change phase by 180 degrees as the transducers pass over edge. This phase change, coupled with the amplitude of the diffraction increasing to 50% of and resembling the reflected response at the diffracting edge are necessary for the overall scattering response from a target plane to be continuous. Reflection from a plane thus appears to originate in the crest of the diffraction hyperbola caused by the plane edge. This will be discussed further in Section 4.3.

2.7 The Relevance of Translation Characteristics

When considering the effect of individual signal path components, it is possible to lose sight of the signal path as a whole. Let us now consider the net effect of the signal path from the calculated source signal to the plotted received signal.

The attenuation of a signal traversing the system signal path was as expected and was not observed to vary significantly with the distance between the transducers and the target structure. The effect, for the most part, of the signal path on a source signal did not vary significantly due to the expected ranges of temperature, humidity, or distance. Those factors which may have had a variable effect on the transmitted signal were limited to reflection and diffraction, and in both cases an effect related to the size and angle of the feature influencing the signal were desired. Thus, while ideal system components would be desirable, the benefits to be gained by their existence were not expected to be substantial.

3 OTHER ASPECTS OF THE IMAGING SYSTEM

3.1 Repositioning the Transducers

The acoustic scatter-mapping imaging system was capable of collecting scatter-mappings in multiple dimensions by repositioning its transducers with respect to the target structure. Had the transducers been fixed in position, the acquired information would have been limited to signal amplitude and phase versus travel time. For the case of a target region radially uniform in the vicinity of the transducers, this information could be used to describe the structure of the target region in the radial direction. Were the target region to vary angularly, i.e. in a second dimension, this acoustic scatter-mapping configuration would be unable to distinguish dissimilar features occurring at differing angles but at similar radii. The data for all features at a given radius would be a combined function of those features. The amount of useful data from the study of radially uniform models is quite limited.

By collecting data at a number of transducer positions, it became possible to collect signal amplitude information in a second dimension, as shown in Fig. 3.1. Assume that the location of the transducers was fixed in the x and z coordinates, but that their position in the y coordinate could be shifted a known amount between data acquisition cycles. Assume also that the target structure was far enough from the transducers and the coverage angle small enough that a plane, constant in the x coordinate, could be approximated as a surface of constant radius. If the target region was uniform in the x-direction, within the vicinity of the transducers, but varied in the y-direction, the amplitude of the received signal could be mapped in two dimensions: signal travel time and the y-coordinate of the transducers. This was the format used in the development of the system described here. The potential for mapping variations in all coordinates will be discussed in Chapter 6.

In order that the transducer position be variable in the z-coordinate it was necessary to mount the transducers on a mobile frame as shown in Fig. 3.2. An overhead trolley was established to transport the transducers across the target region and an overhead rail was fixed in

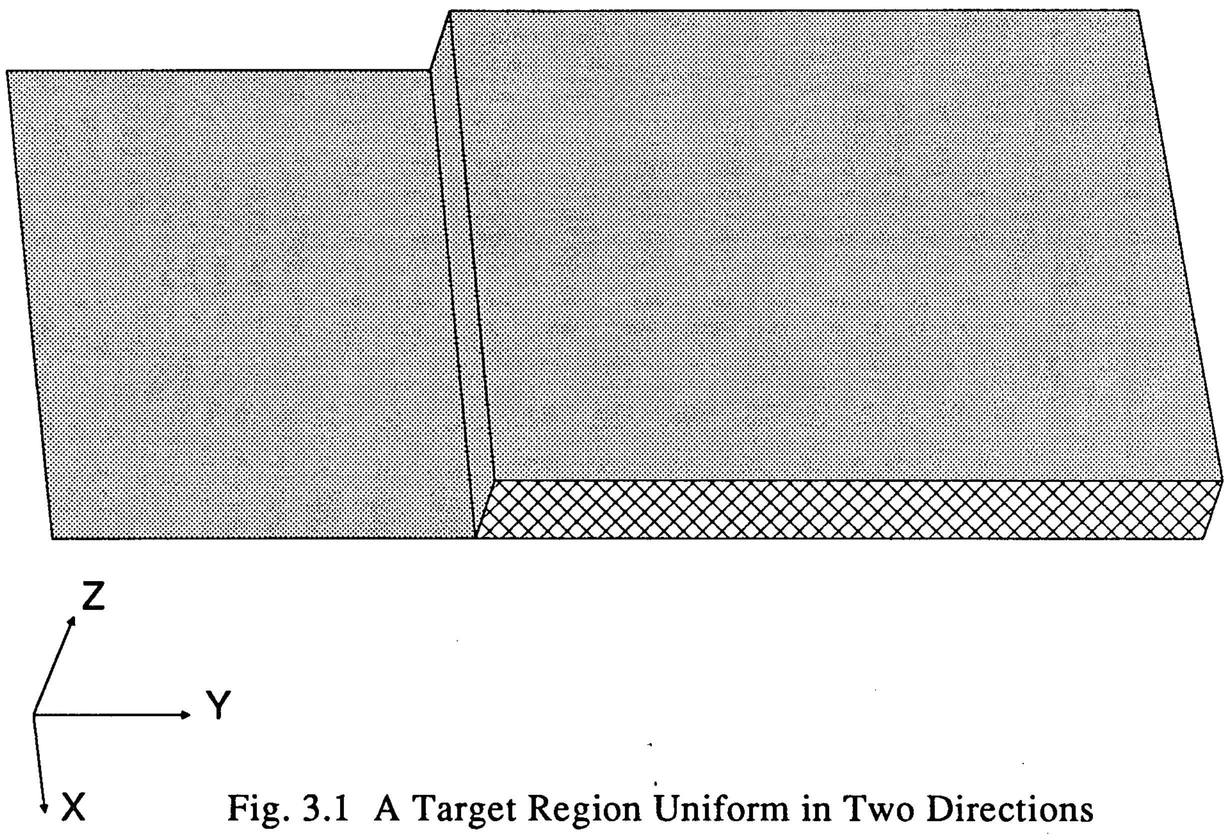

Fig. 3.1 A Target Region Uniform in Two Directions

the *z*-coordinate. In order that the movement of the trolley along the rail be both precise and remotely controlled, a synchronous motor was used to turn a rod, mounted alongside the rail, which in turn operated a linear actuator attached to the trolley. The synchronous motor was operated from the mains and controlled via a data acquisition card installed in the microcomputer. Limit switches at each end of the trolley rail interrupted power to the synchronous motor in the appropriate direction whenever the switch was depressed. This arrangement allowed automatic positioning of the transducers before and during data acquisition sequences. Prior to data acquisition, the trolley was positioned to rest against the east limit switch. Following each data acquisition cycle, the trolley was moved to the next data acquisition location by controlling the duration and direction of motor operation.

Although the target structure and signal transducers could be accurately positioned with respect to a fixed point of reference in the target region, the data sets generated by the system

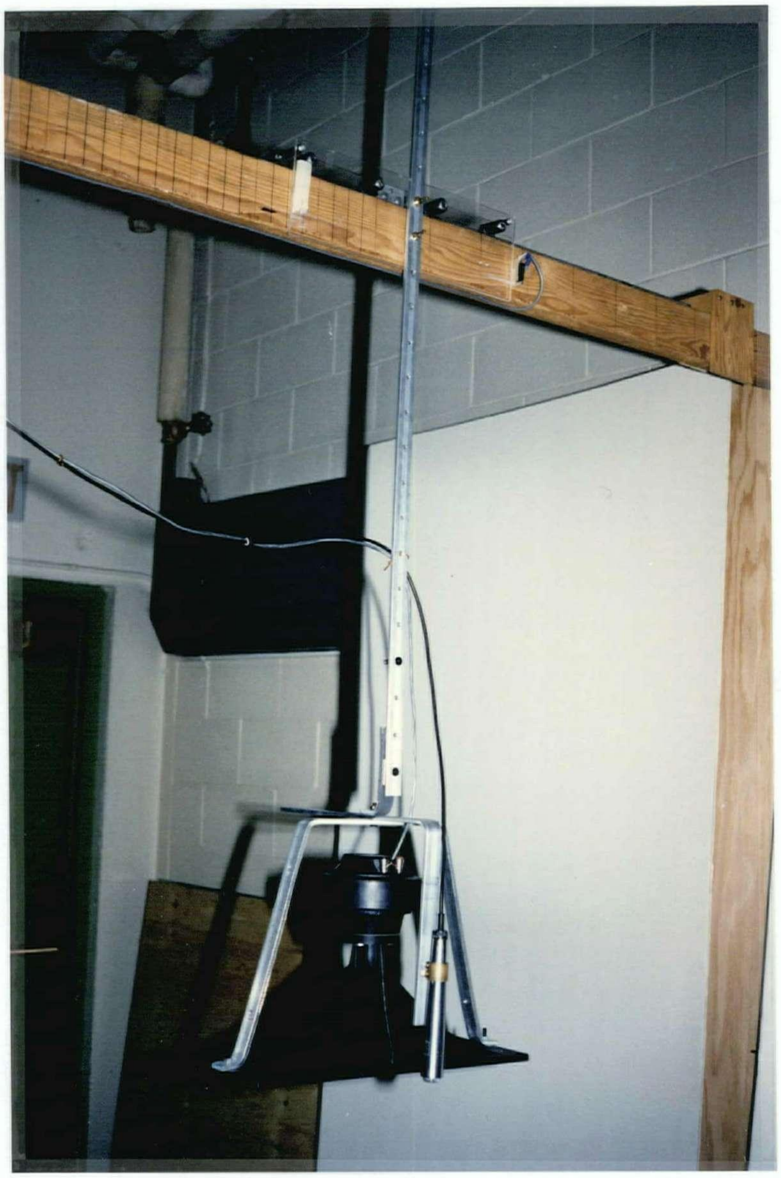

Fig. 3.2 Photograph of the Transducers as Mounted on the Trolley

lacked this physical reference point. They had, instead, a reference line in the z coordinate resulting from the direct reception of the transmitted signal by the receiver transducer, as well as information as to the position of the transducers relative to their initial position. It was therefore necessary to relate the coordinates of the plotted data generated by the system to the reference coordinates in the target region. This was done by acquiring an image of a plane ending in a sharp edge. The reflection from the plane surface was used to relate the z coordinates, while the apex of the hyperbola resulting from the edge was used to relate the y coordinates. The error in relating the z coordinates was less than 4 mm and the error in relating the y coordinates was less than 10 mm. Most of this error can be attributed to the scale at which the data is plotted.

3.2 Plotting of Acquired Data

For the information acquired by the acoustic scatter-mapping system to be useful, it had to be available in a readable format. This kind of data is most often presented as a series of adjacent traces on a y - z plot, each trace representing a single data acquisition cycle location. The horizontal axis represented the position of the trolley, in centimeters, relative to the position at which scatter-mapping was initiated. This initial position coincided with the location of the east limit switch. No reference was provided as to the amplitude variations of each trace. Traces were normally plotted using the same amplitude scale for the entire plot and optimized to make as complete use of the available plotting region as possible. When a reference was required, a known reflector at a known distance was used.

The vertical axis of the y - z plots represent time delayed between the emission of the source signal from the loudspeaker and the collection of the information signal by the microphone. The time delay was described in terms of distance to assist in the interpretation of the data sets. The distance was calculated assuming two way travel at a velocity of 342 meters per second, calculated using equation 2.3 as discussed in Section 2.5.

The system used a plotting routine, *Lasrplot*, which was written in C and generated a Postscript file which was sent to a laser printer. Queries were directed to the user as to the names of the source and destination files as well as the number and length of the traces the user wishes to plot. The plot data was scaled, in amplitude, on either a trace by trace basis or over the entire set of data. To assist in the interpretation of the plotted data, the data could be plotted such that data points above the zero trace were shaded, below the trace zero line were shaded, both above and below the trace zero line were shaded, or no points were shaded. Plot and axis titles were also user-selectable. A listing of this program is in Appendix A.

3.3 Further Processing of Data

The data acquired by the acoustic scatter-mapping system was processed, compressed, and then filed. When the information signal was received by the microphone, it was passed, via an amplifier, to an analog to digital convertor (ADC). The ADC digitized the information signal of a typical sweep at 5 us intervals for a period of 81.92 ms. About two million samples were acquired from a typical 125 trace run. After data from each trace had been collected, it was processed and the result of the processing was windowed, to about 465 KB of data, and filed. The data within the filed window contained all of the information available about the target region from the acoustic scatter-mapping system. The data outside of the filed window contained information about signals which arrived at the microphone before or after sufficient time had elapsed for the source signal to scatter from structures in the target region. This, otherwise superfluous, data was valuable in determining the exact nature of the finished data. The entire set of collected data was not filed due to the amount of data collected, about 8 MB per run, the rate at which it was collected, almost daily, the intimate relationship between the data collected and the reference waveform used to process it, and the ease with which replacement data could be acquired.

The use of a standardized data file format within the acoustic scatter-mapping system encouraged the use of postprocessing. That is, a file could be further processed, after it had been initially acquired and processed, without jeopardizing the system's ability to plot the data. The data could also be converted to ASCII format and then transferred to another computer for further processing. As an example of postprocessing, *Median3*, a three-point space-median filter program, was written. *Median3* was used to reduce the amplitude of horizontally recurring reflections on finished sets of data. A median filter examines a set of data points, determines the median of these points and uses it to generate a new set of points. By using the median, or middle value, of these data points, the filter restricts the rate at which data may change in line with the filter. A time-median filter restricts the frequency components of a set of data. A space-median filter restricts the rate at which data appearing at a regular time-position in a set of space-adjacent traces may change. *Median3* examined space-adjacent, time-similar data points, three at a time, to determine the median value. These median values, had they been plotted as a set of data, would have appeared to be a blurred version of the original set of data. Instead, they were subtracted from the original set of data to reduce the influence of those features which tend to appear at a regular time-position through a number of traces. The effect of median filtering as used by this system can be seen by comparing Figs. 5.8 and 5.9, which are plots of the same set of data before and after filtering.

4 DEVELOPMENT OF A SUITABLE SOURCE SIGNAL

4.1 Characteristics of a Good Source Signal

The ideal source signal is that which could be used to provide the maximum amount of information about the structure under study. With the understanding that the ideal source signal was probably not realizable, we chose to evaluate potential source signals by comparing the results of their use with expected results of the ideal source signal. Let us begin by describing the results possible using the ideal source signal, and later use this reference to evaluate actual source signals.

The ideal source signal will be capable of accurately representing the location of a reflection or diffraction in a plot representative of the target region. That is to say, it must be possible to determine the precise location of the source signal, or a derivative thereof, on a scatter-mapping plot of the target region. This location information is essential to the accurate locating of target structure features in the acquired data. This specification encompasses not only the location in time of the source signal, or its derivative, but the amplitude signature of the source signal, or its derivative, as well. An accurately located feature of unpredictable signature will not be considered useful.

The ideal source signal will be of very short time duration. Often, reflection and/or diffraction returns have extremely short time separations. For these adjacent returns to be easily discernable, the ideal source signal, or its derivative, must be composed of no more than a single positive lobe and one or, at most, two negative lobes. The signal may be inverted if desired.

The ideal source signal will be transmitted through air. Because air is a frequency dependent attenuator, this means that the source signal must be either bandwidth limited or compensated for this attenuation. The complexity of compensation would be increased by the use of the source signal, or its derivative, to determine the distance travelled by the source signal. Bandwidth limitations, however, will limit the minimum size of the signal time window.

We expected that the selection of a source signal, even the ideal source signal, would not be straightforward, given the non-ideal conditions under which it would be used. We applied a final condition, that the source signal be reproducible using the acoustic scatter-mapping system hardware. By imposing this condition, we demanded that the source signal accommodate the attack and decay characteristics of the loudspeaker as well as the frequency response characteristics of the loudspeaker and microphone. While these appeared to be great limitations on the choice of source signal, the crux of the matter remained unchanged. A good source signal had to be repeatable. None of the limitations far affected this, as this acoustic scatter-mapping system was established with the goal of generating repeatable data.

4.2 Selection of a Usable Source Signal

In spite of the discussed limitations imposed on the source signal, the possible choices of source signal form remained virtually unlimited. The constraints on the source signal required only that the end result of the source signal, after having been calculated, generated, translated to an acoustic signal, transmitted through air, translated back to an electrical signal, digitized, and plotted, was of a specified nature. No further restrictions were placed on the form of the source signal or the processing of the received signal.

Prior to my involvement in this project, the acoustic scatter-mapping system made use of a pulsed source. While a pulsed source was quite simple to calculate, the result of passing this signal through a digital to analog convertor (DAC), an amplifier and a loudspeaker, was that the desired pulse was malformed and followed a ringing signal of approximately the same amplitude and frequency as the desired pulse. A typical pulsed-source signal and the resultant received signal are shown in Figs. 4.1 and 4.2. This difficulty invited further consideration of possible source signals.

The distortion seen in the signal received by the microphone, when a calculated pulse source was generated, was, predominantly, the result of two factors:

1. The limited bandwidth of components in the signal path. The loudspeaker, as discussed in Section 4.3, had a limited response outside of its 800 Hz to 15 KHz range. The microphone responded similarly outside its 200 Hz to 13 KHz range.

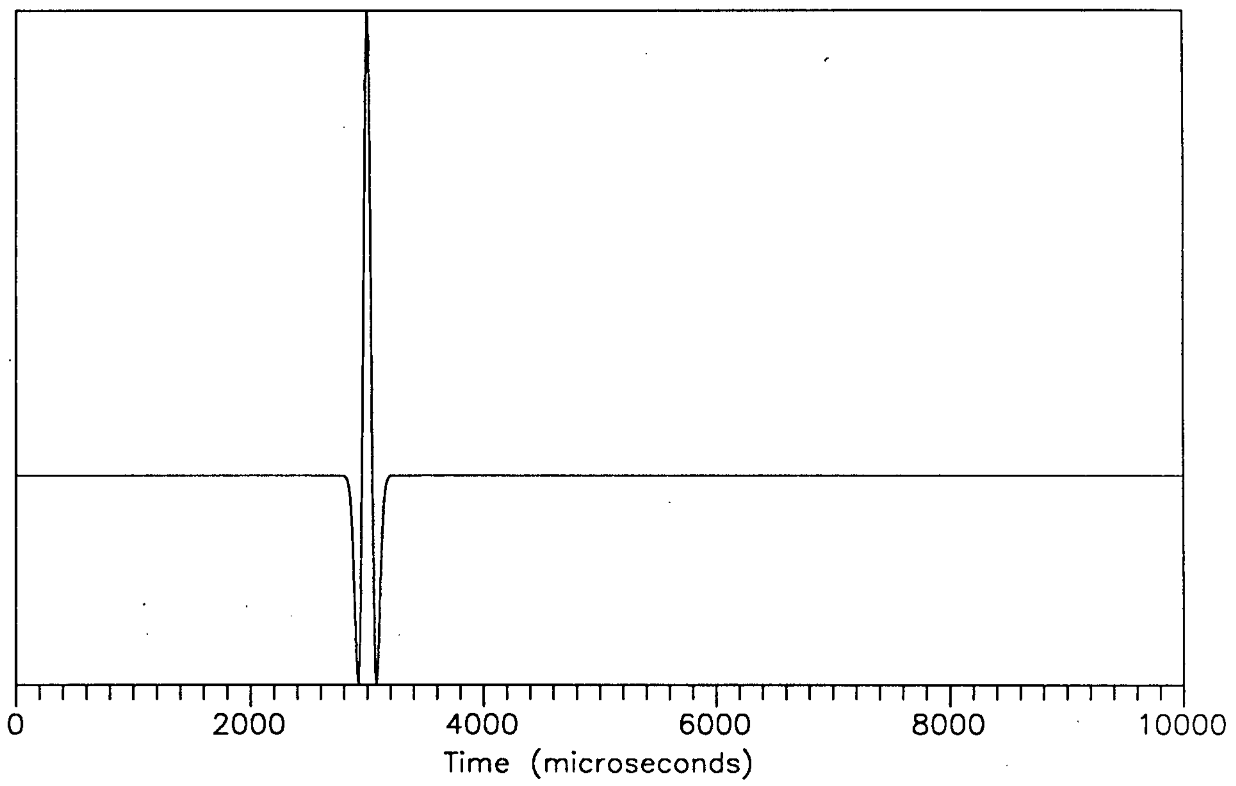

Fig. 4.1 A Calculated Pulse Source Signal

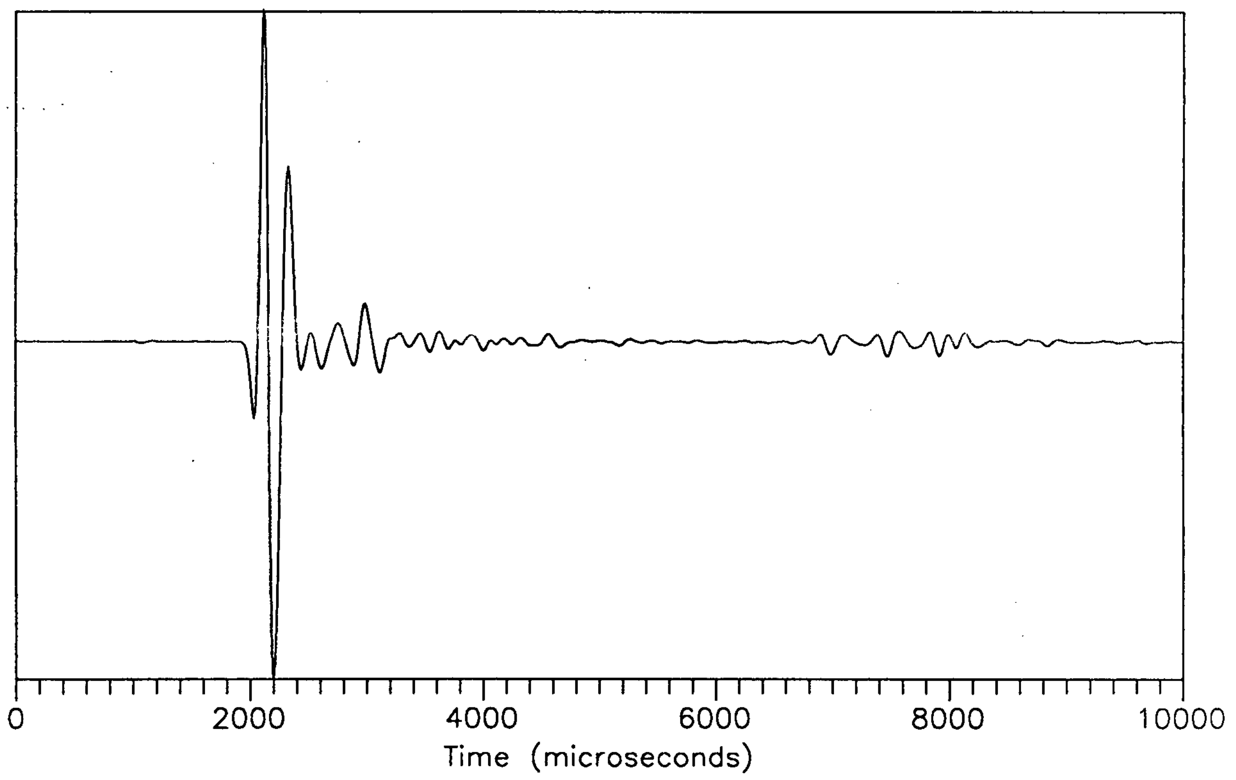

Fig. 4.2 Distortion of a Pulse Source by the Signal Path

2. The attack and decay parameters of the transducers. These parameters, not normally available for loudspeakers other than those designed for use at low audio frequencies, were found to be on the order of several cycles for the loudspeaker used in this system.

Having better understood the limitations imposed on the calculated source signal, we intuited that any desired source signal must be a pre-distorted version of the waveform occurring as an indicator on the plotted data sets. Thus, attention shifted from the question of which kind of pulse signal was most desirable in the finished plots to which kind of source signal would cause the most desirable signal to appear in the finished data sets. At this point the issue of whether or not to use a pulsed source signal at all was raised.

In order for the acoustic scatter-mapping system to make use of a pulsed source signal, the non-ideal parameters of the loudspeaker and microphone required that the signal be pre-distorted. The exact characteristics of the pre-distortion, however, were unknown, although the large size and low damping factor of the loudspeaker suggested that its attack and decay parameters were quite large. Three options were apparent:

1. Determine the response parameters of this loudspeaker and determine a matching predistortion such that the predistortion and distortion effectively cancel each other out.
2. Replace the loudspeaker, and possibly the microphone, with another of improved and/or known parameters and determine a suitable pre-distortion for this new configuration.
3. Replace the pulsed source signal with a signal more compatible with the system components and capable of causing pulses to appear at the appropriate locations in the finished data sets.

We decided to make use of the third option, that of replacing the form of the calculated source signal.

In the field of seismology there is a technique called *Vibroseis* [12], wherein a swept-frequency mechanical vibrator source replaces the traditional explosive charge source used to study underground structures. While recordings made of the signal transmitted through the ground as a result of pulsed source signals were used to directly determine the underground structures, in the *Vibroseis* technique the received swept-frequency signal was crosscorrelated with a copy of the source signal. The result was similar to the traditional method in that a pulse appeared at the relative location of significant events, but differed in that the pulse was not the originally transmitted signal, but the autocorrelation of the originally transmitted signal. A similar technique in radar is called *Chirp Radar* [13]. We decided to make use of a similar technique in this acoustic scatter-mapping system.

4.3 Swept-Frequency Source Signal Description

In order to best appreciate the differences between a directly-recorded pulsed source system and a crosscorrelated swept-frequency system, let us consider the characteristics of signals of both types as they are acquired, processed, and plotted. In order to simplify analysis of the signals, we will first determine the ideal response and then present data for the non-ideal case to demonstrate the difficulties inherent in using non-ideal components. Let us assume that the system is immersed in a homogenous medium of constant velocity v and contains a single reflector at a distance r from the source-receiver pair. The system response, observed at the receiver, to a step function signal emitted from the source, would be

$$k(t) = u(t_o) + a_d(r)\, u(t - 2r/v) \tag{4.1}$$

where $u(t)$ is the step function, $a_d(r)$ is the spatial attenuation of the signal due to the distance from the source-receiver pair to the reflector, and t_0 is the time required for the signal to leave the source and enter the adjacent receiver.. The first term is the result of a zero-offset source receiver pair and the second is identical to first but delayed by the time required for the signal to travel to and from the reflector. More generally, the signal $h(t)$, seen at the receiver when the signal $g(t)$ is emitted by the source, can be described as the convolution of the source signal and the system response

$$h(t) = g(t) \otimes k(t) = g(t) \otimes u(t_o) + a_d(r)\, g(t) \otimes u(t - 2r/v). \tag{4.2}$$

Or, in the frequency domain,

$$H(f) = G(f)\,K(f). \tag{4.3}$$

Now let us assume that an ideal unit amplitude pulse source is used. The expected result

$$h(t) = \delta(t_o) + a_d(r)\,\delta(t - 2r/v) \tag{4.4}$$

would be as shown in Fig. 4.1. Unfortunately, generating signals using a non-ideal source transducer tended to result in the response shown in Fig. 4.2. This was a result of the factors discussed in the previous section.

Let us now consider a swept-frequency source signal. Swept-frequency source signals may be described as [12]-[13]

$$s(t) = a_s(t)\cos[\theta(t)] \tag{4.5}$$

where $a_s(t)$ is the amplitude function, typically

$$a_s(t) = A[u(t_1) - u(t_2)] \tag{4.6}$$

where t_1 is the start and t_2 is the end of the sweep. A is the peak amplitude, a constant. The phase function, $\theta(t)$, typically quadratic, is expressed as

$$\theta(t) = \frac{3\pi}{2} + 2\pi\int_{t_2}^{t_1} [c_1 + c_2(\tau - t_1)]\,d\tau \tag{4.7}$$

where c_1 and c_2 are constants.

Now let us assume that this swept-frequency signal is used as a source. The expected result

$$h(t) = a_s(t)\cos[\theta(t)] + a_d(r)\,a_s(t - 2r/v)\cos[\theta(t - 2r/v)] \tag{4.8}$$

would be as shown in Fig. 4.3. While the signal seen at the receiver, shown in Fig. 4.4, in response to a calculated signal of this form would be slightly different from that expected, it would not differ as severely as the response to the pulse source signal.

After the received signal resulting from a calculated swept-frequency source signal had been digitized, it would be crosscorrelated with a copy of the source signal. The crosscorrelation function [14] is defined as

$$R_{gh}(t) = \int_{-\infty}^{\infty} g(\tau + t)\, h(\tau)\, d\tau \tag{4.9}$$

in the time domain. In the frequency domain, this is expressed as [14]

$$\mathcal{R}_{gh} = G(f)\, H^*(f). \tag{4.10}$$

Substituting in equation 4.3 makes the situation much clearer

$$\mathcal{R}_{gh} = G(f)\, G^*(f)\, K(f) \tag{4.11}$$

The effect of crosscorrelating the received signal with a copy of the source signal was the same as convolving the autocorrelated source signal with equation 4.1, the response of the system. This allowed greater latitude in the choice of source signal, as the precise envelope of the signal is no longer important so long as the uniqueness of the source signal with respect to a time shifted copy of itself can be optimized. The crosscorrelation of the received swept-frequency signal with a copy of the source signal is shown in Fig. 4.5.

The system response to a source waveform does not change dramatically for the case of multiple reflectors or diffractors replacing reflectors. For a case of reflections from more than one point, equation 4.1 would become

$$w(t) = u(t_0) + \sum_{r=0}^{n} a_d(r_r)\, u(t - 2r_r/v). \tag{4.12}$$

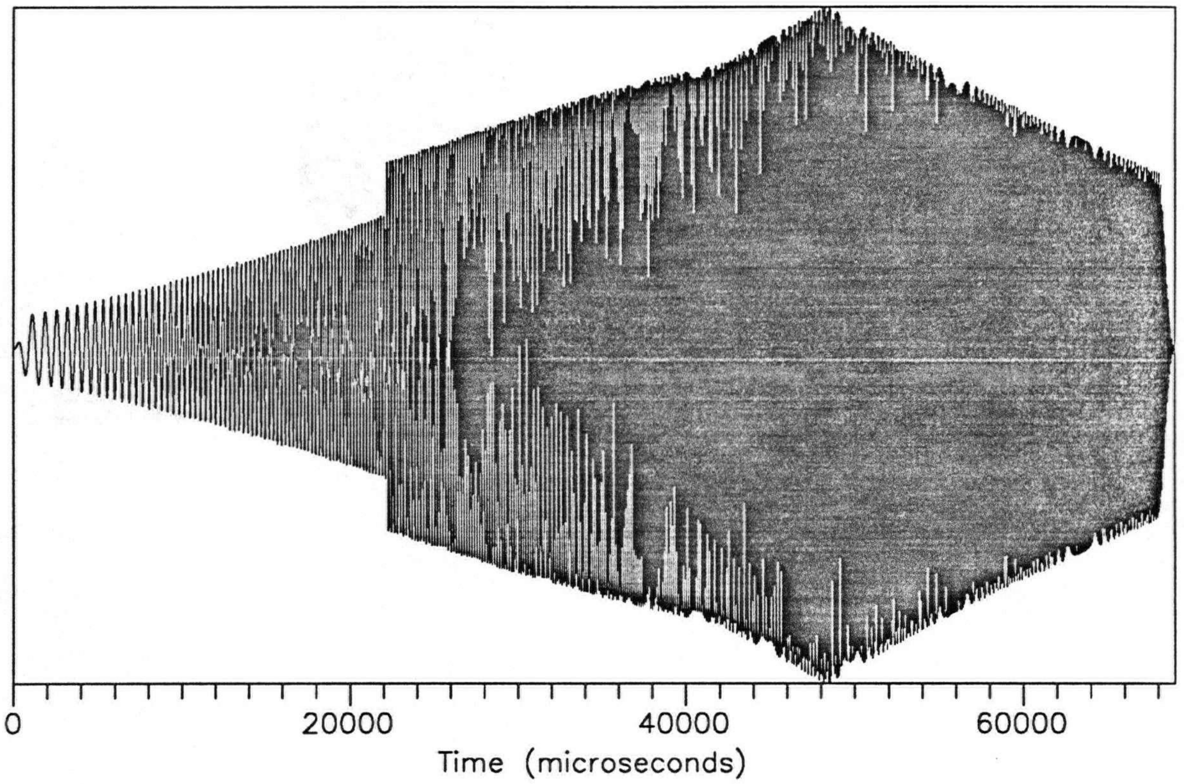

Fig. 4.3 A Calculated Swept-Frequency Source Signal

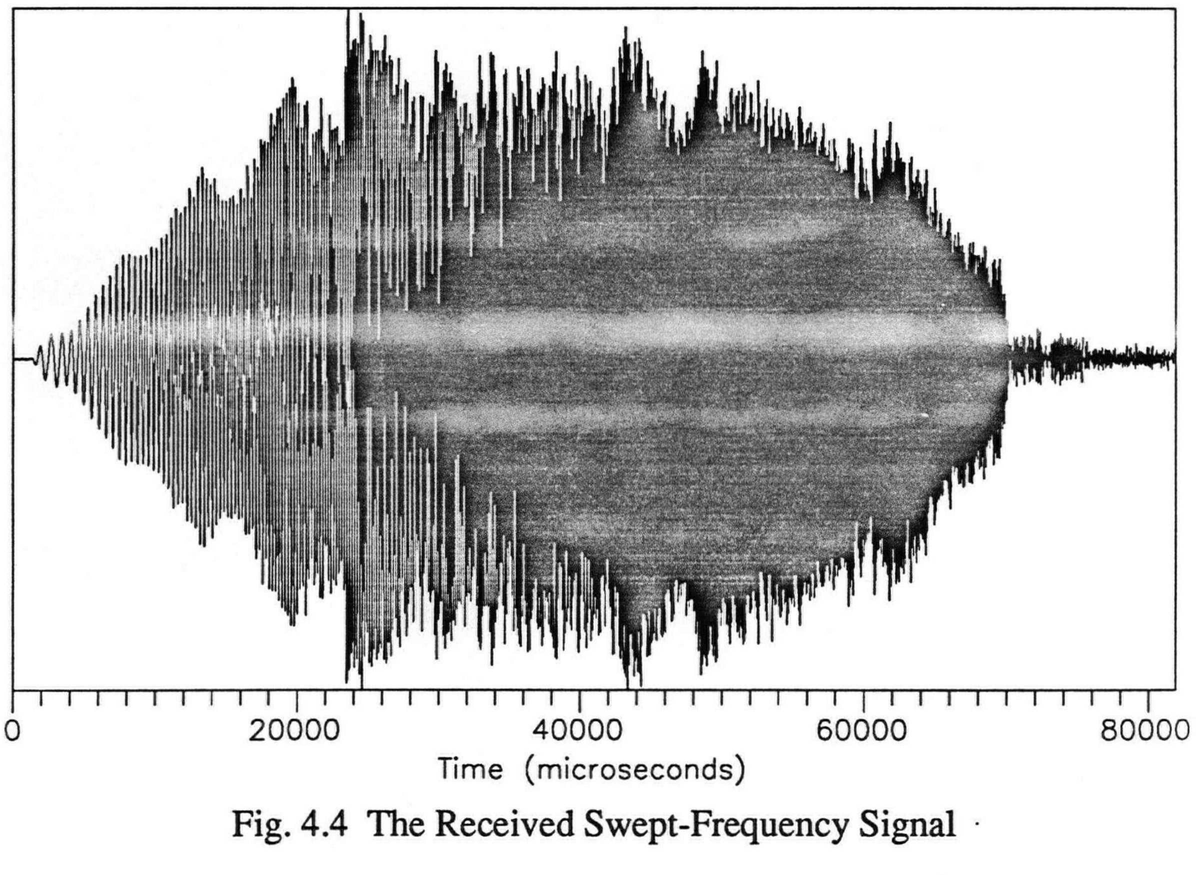

Fig. 4.4 The Received Swept-Frequency Signal

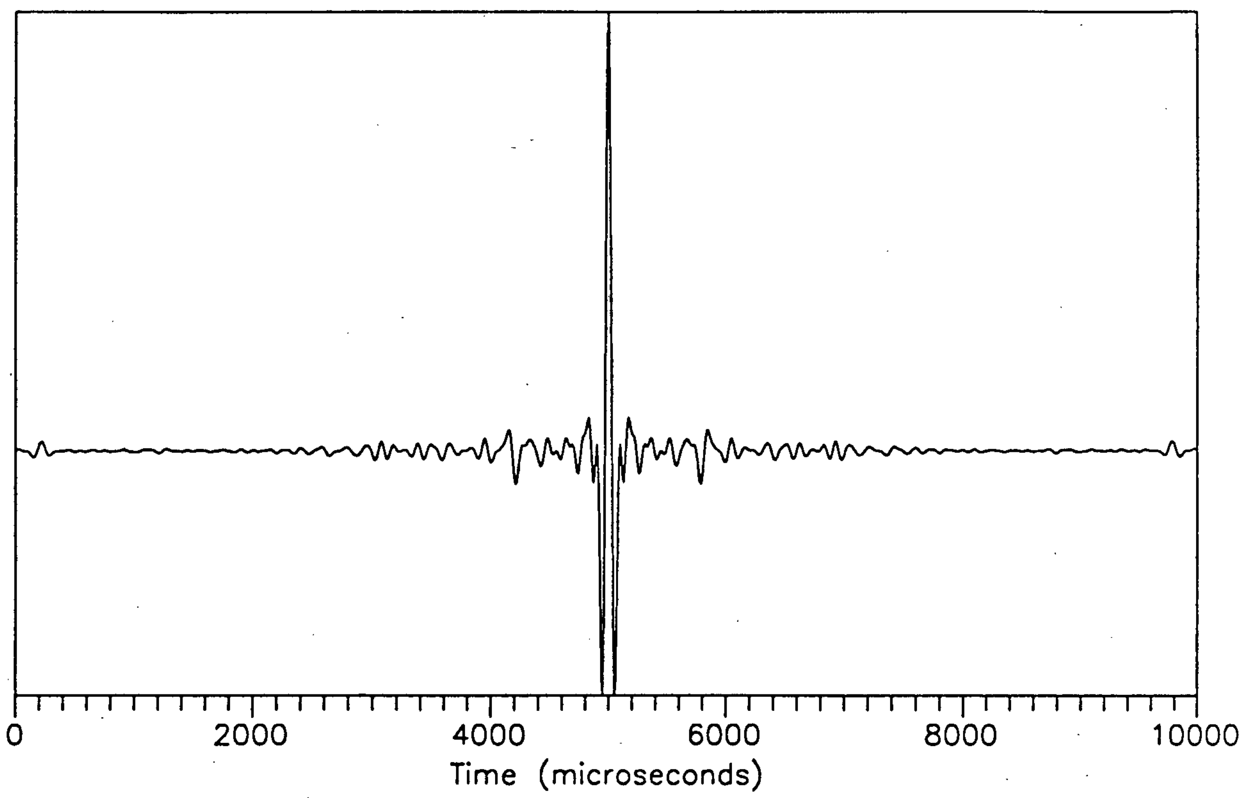

Fig. 4.5 The Crosscorrelated Received Swept-Frequency Signal

The situation for diffractors is similar, as discussed in Section 2.6, but causes a phase shift also. Thus, the system response would become

$$w(t) = u(t_o) = \sum_{r=0}^{n} a_d(r_r)\, u(t - 2r_r/v) + \sum_{d=0}^{m} a_d(r_d)\, u(t + \theta_d/\omega - 2r_d/v) \qquad (4.13)$$

where θ_d is the phase shift caused by the diffractor and ω is the instantaneous frequency of the source signal. Whereas the existence of a reflector in the target region would cause a time-shifted copy of source signal to appear in the received signal, the existence of a diffractor would cause a time and phase shifted copy of the source signal to appear in the received signal.

4.4 Selection of a Suitable Swept-Frequency Source Signal

Having determined the characteristics of a good swept-frequency source signal, and having understood the stages through which the signal must pass, the final step was to develop a swept-frequency source signal suited to the exact characteristics of the other components of the acoustic scatter-mapping system. This involved trading off signal bandwidth, signal-to-noise ratio, sweep rate, and amplitude envelope to find the autocorrelation pulse best approximating the ideal source signal.

The bandwidth and sweep rate of a swept-frequency source signal were predominantly limited by the characteristics of the source transducer, the loudspeaker. This was a result of both the relative power and complexity of the source transducer. The first step, therefore, was to determine the relative tradeoff between the signal bandwidth and sweep rate. We found that a linearly swept source signal, i.e. a sweep whose instantaneous frequency varied linearly with time, could be swept at a high rate if the sweep bandwidth was quite narrow. As the bandwidth of the signal was increased, the amplitude envelope of the received signal began to show successive maxima and minima at regular intervals. When the bandwidth was held constant and the sweep rate reduced, the distance between the minima increased.

In order to eliminate the successive maxima and minima within the envelope of the received signal, we decreased the sweep rate until only two minima were observed, one at each end of the sweep. A maximum also appeared near the center of the envelope. This solution to the bandwidth versus sweep rate tradeoff left unresolved the tradeoff between bandwidth, sweep rate, and autocorrelated pulse shape. The signal-to-noise ratio of the crosscorrelated received signal, varying with the square root of the total sweep time [12] was not directly considered.

The width of the pulse which appeared at the center of autocorrelated source signal was inversely related to the bandwidth of the autocorrelated received signal and the received signal prior to autocorrelation. It was therefore decided that the source signal should have as wide a bandwidth as possible within the limitations of the transducers. This left the problem of selecting the most useful signal amplitude envelope and it became necessary to establish quantitative criteria to evaluate the effect of the envelope shape on the autocorrelated signal. We also found that the realization of a particular received signal was not directly attainable. That is, given a particular autocorrelated pulse, we were not able to specify the exact nature of the calculated source signal necessary to realize the particular pulse as its crosscorrelated received signal. Instead, we knew the relationships between the envelope of a signal and its autocorrelation and between a calculated source signal and the resulting received signal. An iterative process was then required to optimize the calculated source signal to realize the desired autocorrelated signal.

Koefoed [15] selects three aspects of autocorrelated swept-frequency signals and discusses the effect of signal characteristics on each. These aspects were used as aids in the evaluation of potential source signals. The first characteristic, the breadth of the central lobe, is expressed as the time difference between the extrema of the two side lobes. The second, the side lobe ratio, is expressed as the ratio between the amplitude of the side lobes and that of the central lobe. The final aspect, the amplitude of the side-tail oscillations, is described as the shape of the wavelet at a time larger than three-quarters of the dominant period from the center of the wavelet. For the purposes of this thesis, this characteristic will be described by the maximum amplitude of lobes in this region with respect to the amplitude of the main lobe.

Koefoed states that the infinite integral of the wavelet must be zero, requiring sufficient area of the negative portions of the wavelet to balance the area of the positive portions. Should the side lobe ratio be less than 0.5, side-tail oscillations must occur to balance the main lobe area unless the side lobes are sufficiently wide to accommodate their low amplitude. While the prospect of large side-tail oscillations or a large side lobe amplitude were undesirable, the alternative situation of wide side lobes was even less so. We chose to use an autocorrelation envelope containing large, narrow side lobes and minimal side-tail oscillations. Koefoed recommended the autocorrelation of a signal with a triangularly shaped amplitude envelope.

Having determined the characteristics of a suitable received signal, we calculated a source signal to realize the desired received signal. This was done iteratively, beginning with a calculated signal similar to the desired received signal. As we observed the effects of transducer attack and decay we compensated for them and calculated a new source signal. In addition to fitting the amplitude envelope of the received signal, we had to further taper the ends of the calculated source signal to limit the background level of the autocorrelation [16]. We did this with a cosine taper at both the start and end of the sweep. The cosine taper was of the form

1 - cos ($n \pi / N$), where n was the number of data points from the start or end, respectively, of the signal, and N was the number of points over which the taper was to occur.

The spectral density of the calculated and received signals are shown in Figs. 4.6 and 4.7 respectively. The autocorrelation of the received signal is shown in Figs. 4.8 and 4.9. The listing of the program, *Wavegen*, used to calculate the source signal is in Appendix A. The dominant period of the autocorrelated received signal was 107 μs, corresponding to a dominant frequency of 9.3 KHz. The sidelobe ratio was 63% and the side-tail oscillations were no greater than 7% beyond the first sidetail at 11%. A copy of this received signal, acquired by directing the loudspeaker into an area effectively free of scatterers and directing the microphone toward the loudspeaker at a distance of 50 cm, was used as the reference signal, with which to crosscorrelate a received signal, in the subsequent operation of the acoustic scatter-mapping system.

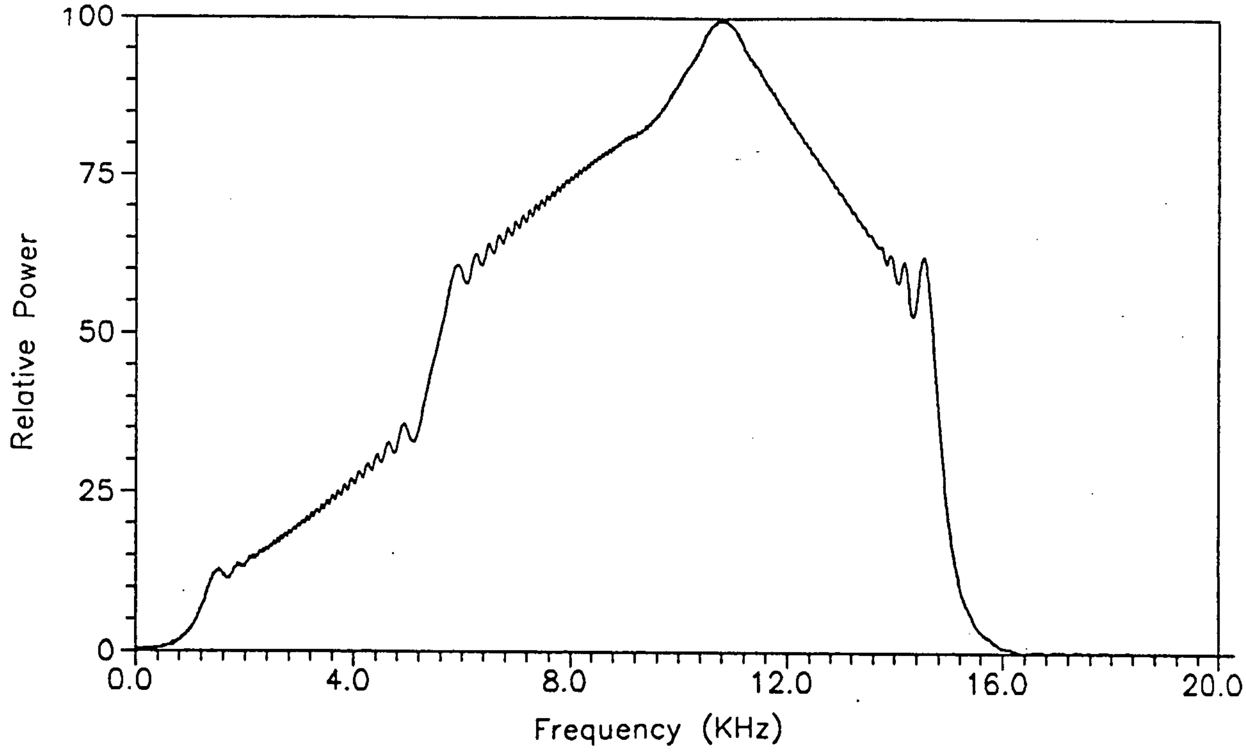

Fig. 4.6 Spectral Density of the Calculated Waveform

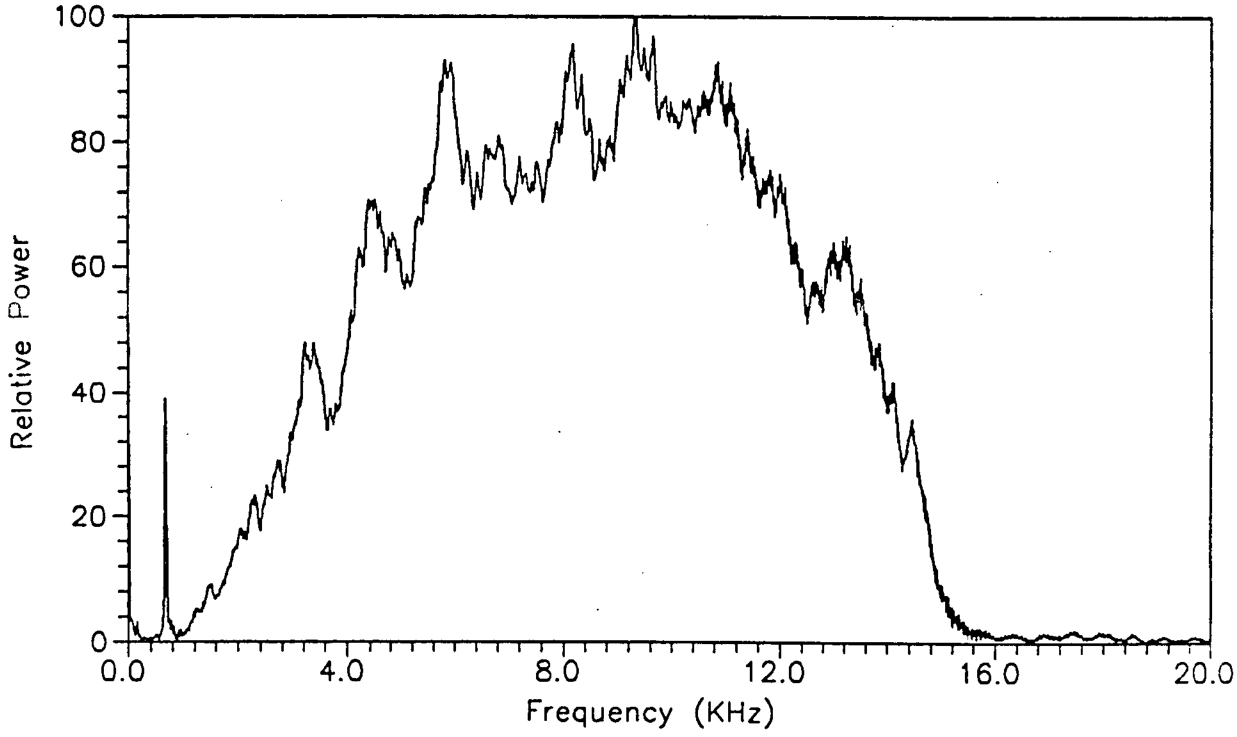

Fig. 4.7 Spectral Density of the Generated Waveform

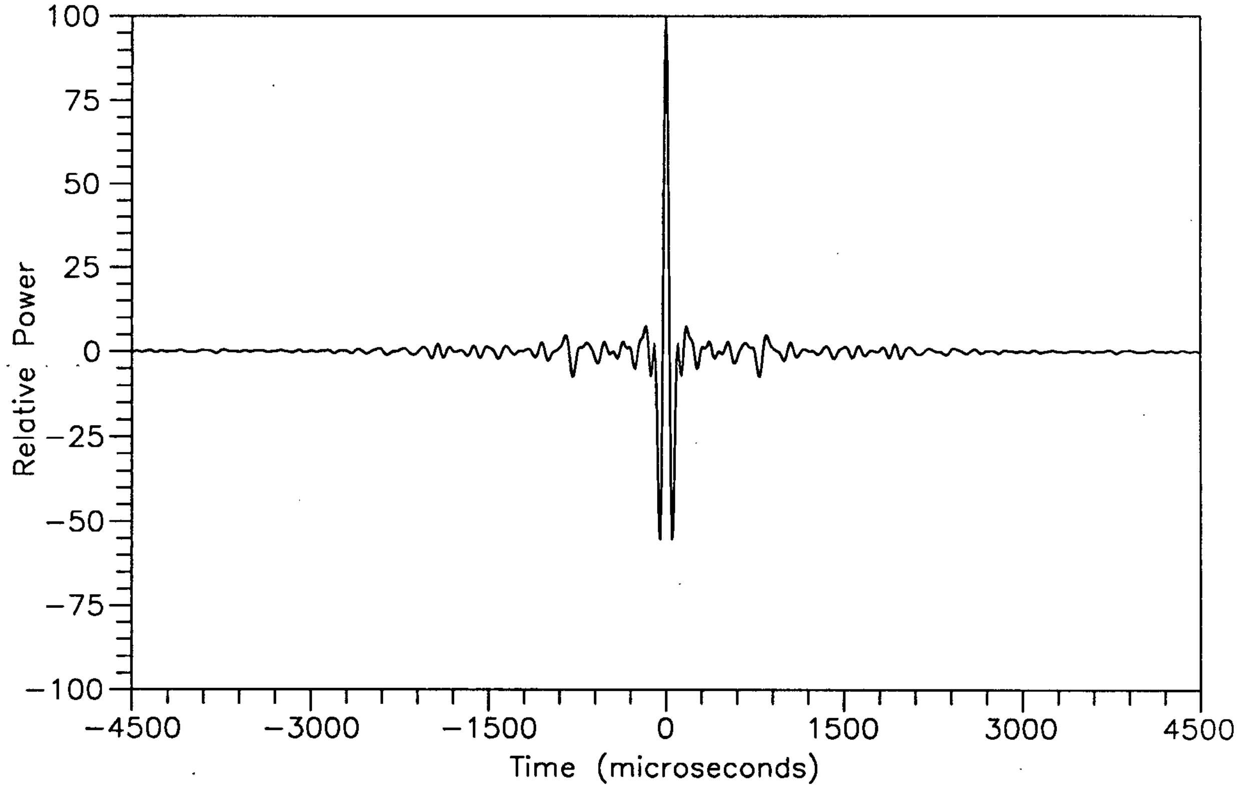

Fig. 4.8 Autocorrelation of the Reference Waveform

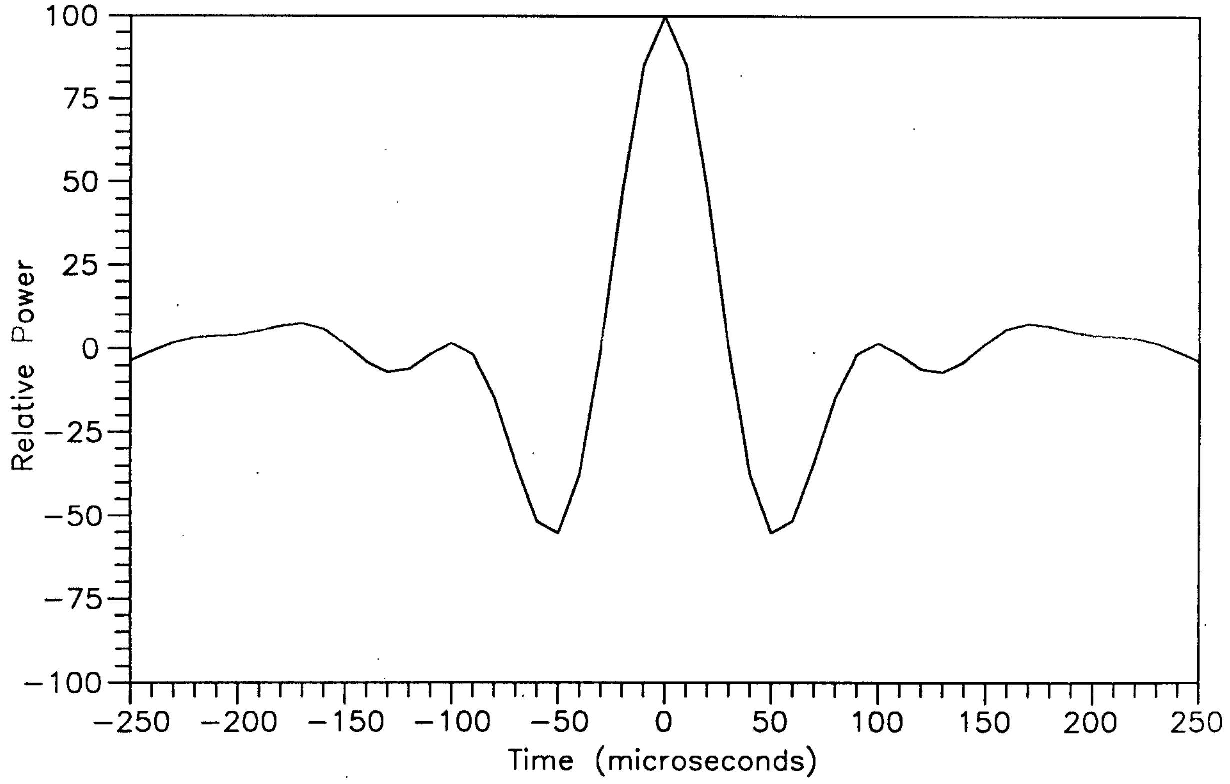

Fig. 4.9 Autocorrelation of the Reference Waveform

5 EVALUATION OF THE SYSTEM ON SOME SIMPLE MODELS

5.1 Target Region Description

In order to demonstrate the applicability of this acoustic scatter-mapping system, scatter-mappings acquired from three models are provided. In each case, the model is described along with the acoustic scattering expected due to features of interest on the model. The acoustic scatter mapping of the model is then presented with a discussion of any relevant differences between the expected and mapped acoustic scattering. The first model, that of no model, is provided to familiarize the reader with the characteristics of target region prior to the entry of a model. The second model, that of a buried edge, is presented to demonstrate the ability of the system for the case of a well-known structure. The final model, that of a plane ending in a rounded edge of constant radius, is provided to demonstrate the sensitivity of the system and corroborate the results of a diffraction model prediction.

5.2 The Vacant Target Region

The first model to be considered is that of the vacant target region. While this may not appear to be a useful model, its importance lies in the baseline it will provide for other models. That is, effects observed in the acoustic scatter-mapping of the vacant target region are likely to appear as well in acoustic scatter-mappings of other target structures. This baseline mapping is valuable also as a simplified introduction to the data sets generated by this system.

When considering the diagram of the vacant target region in Fig. 5.1, one should take note of several important features. The first is that the target region is virtually continuous in the x - direction in the vicinity of the transducers. This is important as the variations in which we are interested are restricted to those in the y and z directions. Should there be variations in a third dimension, the system, able to vary its acquisition of information in only two dimensions, time

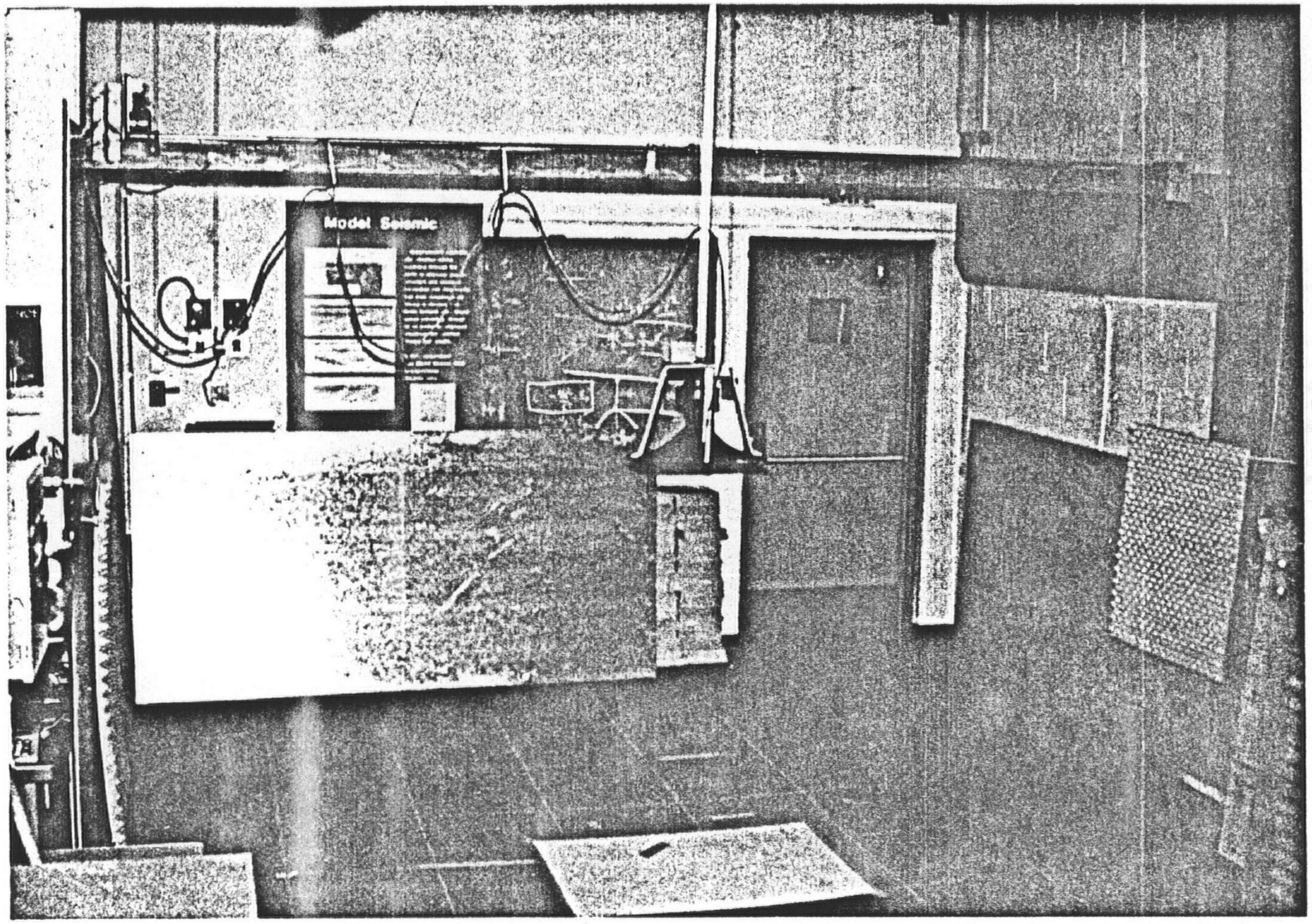

Fig. 5.1 Photograph of the Vacant Target Region

and the *y* - direction, would not be able to provide sufficient information to the user for the user to locate the features responsible for all of the returns in the scatter mapping.

A second consideration of the physical layout of the acoustic scatter mapping system in the target region is the adjacent position of the transducers. This is important as the system was initially developed to acquire data for the simplified case of a zero-offset source-receiver pair. This desire was thwarted by the physical dimensions of the transducers as well as interference between transducers in close proximity. The compromise places the face plate of the loudspeaker, the point on the loudspeaker from which locational measurements were made, horizontally adjacent to and 9 cm beside the microphone cartridge. The result is that there is a measurable delay between the signal being sent to the loudspeaker and a copy of the signal, travelling from the loudspeaker driver, through the loudspeaker horn, diffracting from the face plate of the loudspeaker, and travelling to the microphone cartridge, being sampled. This delay does not vary unless the transducers are repositioned relative to each other, a situation which does not normally occur. The result of this delay is a phase shifted copy of the source signal appearing in the generated scatter mapping which may be used for relative amplitude and time measurements. The time delayed signal resulting from the non-zero offset of the source and receiver appeared in plots of scatter mappings as if it were the result of a feature located 4.5 cm below the speaker face plate. This apparent impossibility was the result of assuming that features responsible for scattering are sufficiently far away from the transducers to validate the assumption that the transducers are zero-offset.

A third feature to consider in the environment of the acoustic scatter mapping system is the location of the nearby walls, floor, and ceiling. The south wall, in the background of the vacant target region diagram, is 270 cm away from the transducers. Because the net time window in which the signal received by the microphone is considered is only 9.3 ms, returns from features more than 162 cm away from the transducers will not appear in the scatter mappings. Thus, the south wall will not influence the scatter mappings. Neither will the ceiling, at a distance of 260 cm nor the nearest object on the north side, at 335 cm from the transducers. The concrete floor of the region, however, being only 140 cm from the transducers will be responsible for a reflection appearing in the generated data sets. The edges of this reflective material, however, are the cause of a pair of weaker diffractions. The east and west walls, on the left and right sides of the target region diagram respectively, are only 55 cm and 87 cm from the transducers at their respective ends of the trolley position. These will result in returns appearing in the generated plots which will vary with the distance between the transducers and the wall. Corrugated foam absorbers were added to the most significant portions of the east and west walls to reduce their reflection coefficient.

Fig. 5.2 is a diagram of the scattering expected due to observed features in the vacant target region. Fig. 5.3. is a diagram of anticipated returning signal paths. The two most easily predicted returns are those generated by the non-zero-offset of the transducers, from (0, 4.5) to (248, 4.5), and reflection from the floor, from (0, 139) to (248, 139), sloping slightly toward a floor drain from (88, 141) to (190, 141). A corner reflection, expected from the corner between the floor and the east wall, appears from (0, 150) to (26, 163). Reflections from the east and west walls are expected from (0, 53) to (110, 163) and from (168, 163) to (248, 84) respectively. The wall reflections are expected to be much weaker than those from the floor, as the walls were covered, in relevant areas, with acoustically absorptive foam. The floor drain in the center of the target region was covered with reflective material to reduce diffraction from the grating over the drain, although the edges of the material were expected to be responsible for diffraction hyperbolae centered at (88, 141) and (190, 141).

Fig. 5.4 is an acoustic scatter mapping of the vacant target region. Comparison of this mapping with that expected, Fig. 5.2, shows that the two are nearly identical. The direct signal from the loudspeaker to the microphone appears as a strong phase-shifted return and the floor appears as a strong zero-phase return. Fig. 5.5 shows the same acoustic scatter mapping data following the application of the three-point space median filter discussed in Section 3.3. In all median filtered data presented, the amplitude gain of the plot has been increased significantly to emphasize the smallest details of the data. In this plot we can see that the returns from the uneven floor were not as well filtered out as those of the direct loudspeaker to microphone signal. The weak reflection from the west wall and the weaker reflection from the east wall are

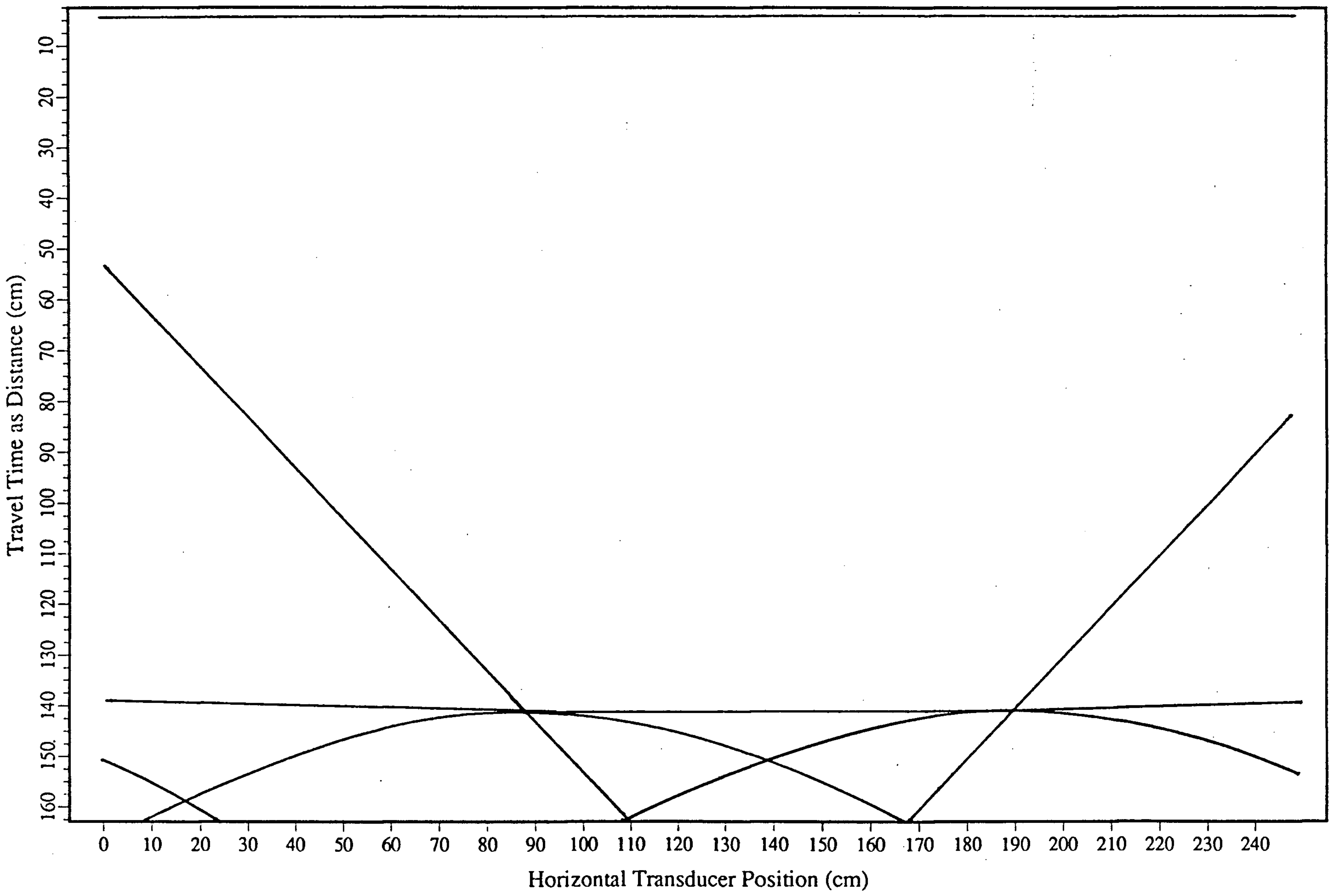

Fig. 5.2 Expected Acoustic Scatter Mapping of the Vacant Target Region

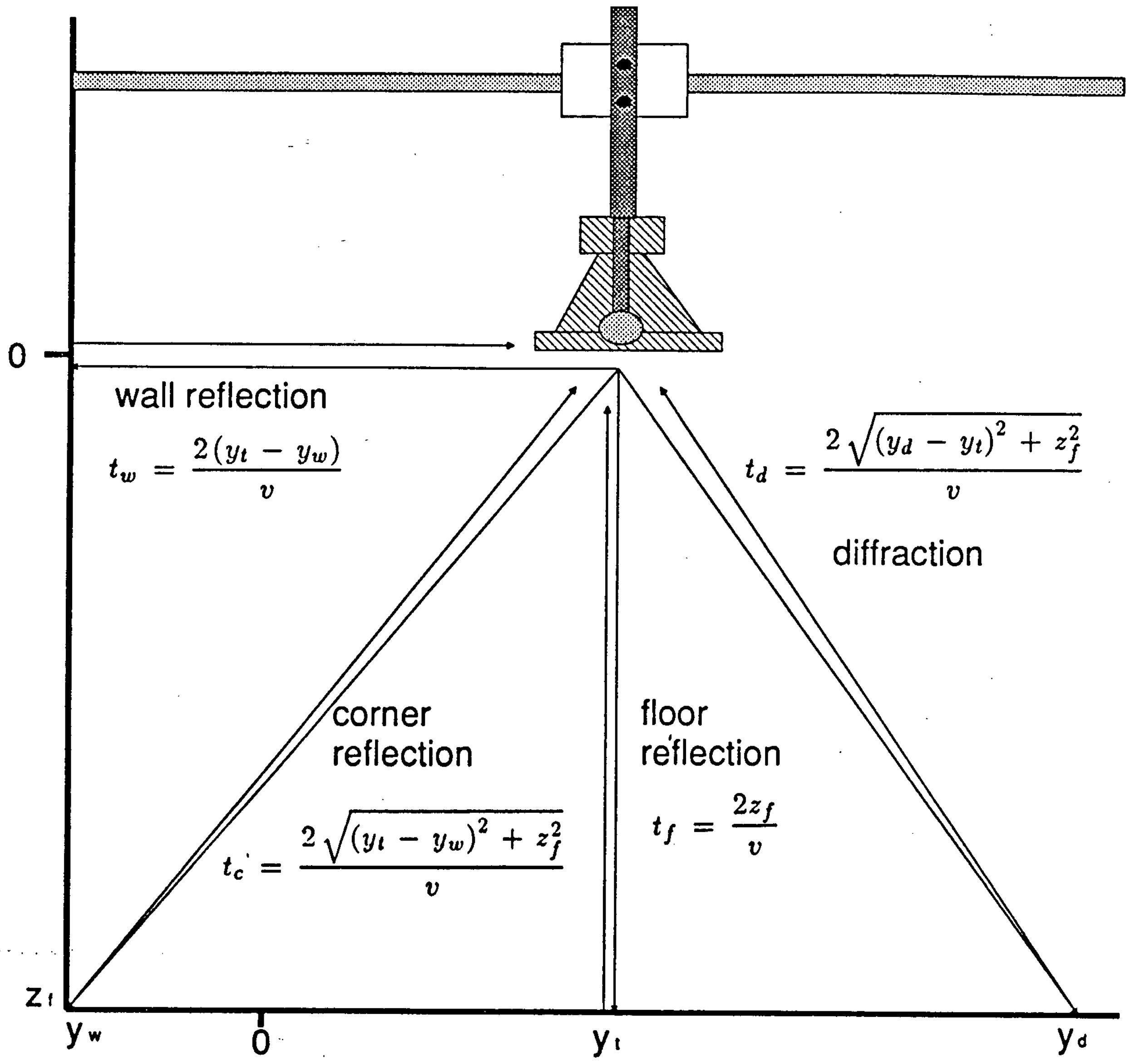
0
wall reflection
$t_w = \frac{2(y_t - y_w)}{v}$
$t_d = \frac{2\sqrt{(y_d - y_t)^2 + z_f^2}}{v}$
diffraction
corner reflection
floor reflection
$t_c' = \frac{2\sqrt{(y_t - y_w)^2 + z_f^2}}{v}$
$t_f = \frac{2z_f}{v}$
z_f
y_w
0
y_t
y_d

Fig. 5.3 Diagram of Expected Returning Signal Paths and Times

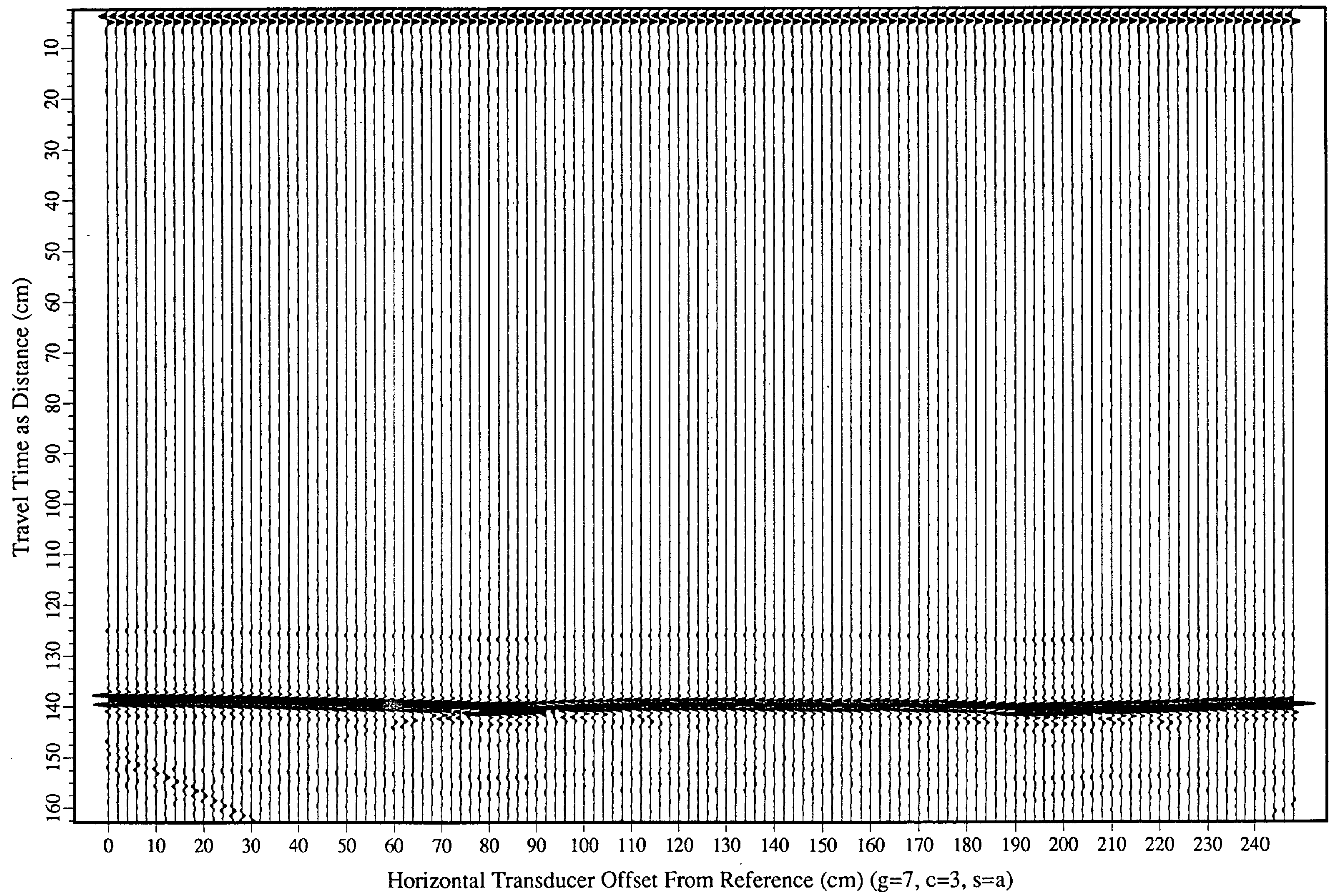

Fig. 5.4 Acoustic Scatter Mapping of the Vacant Target Region

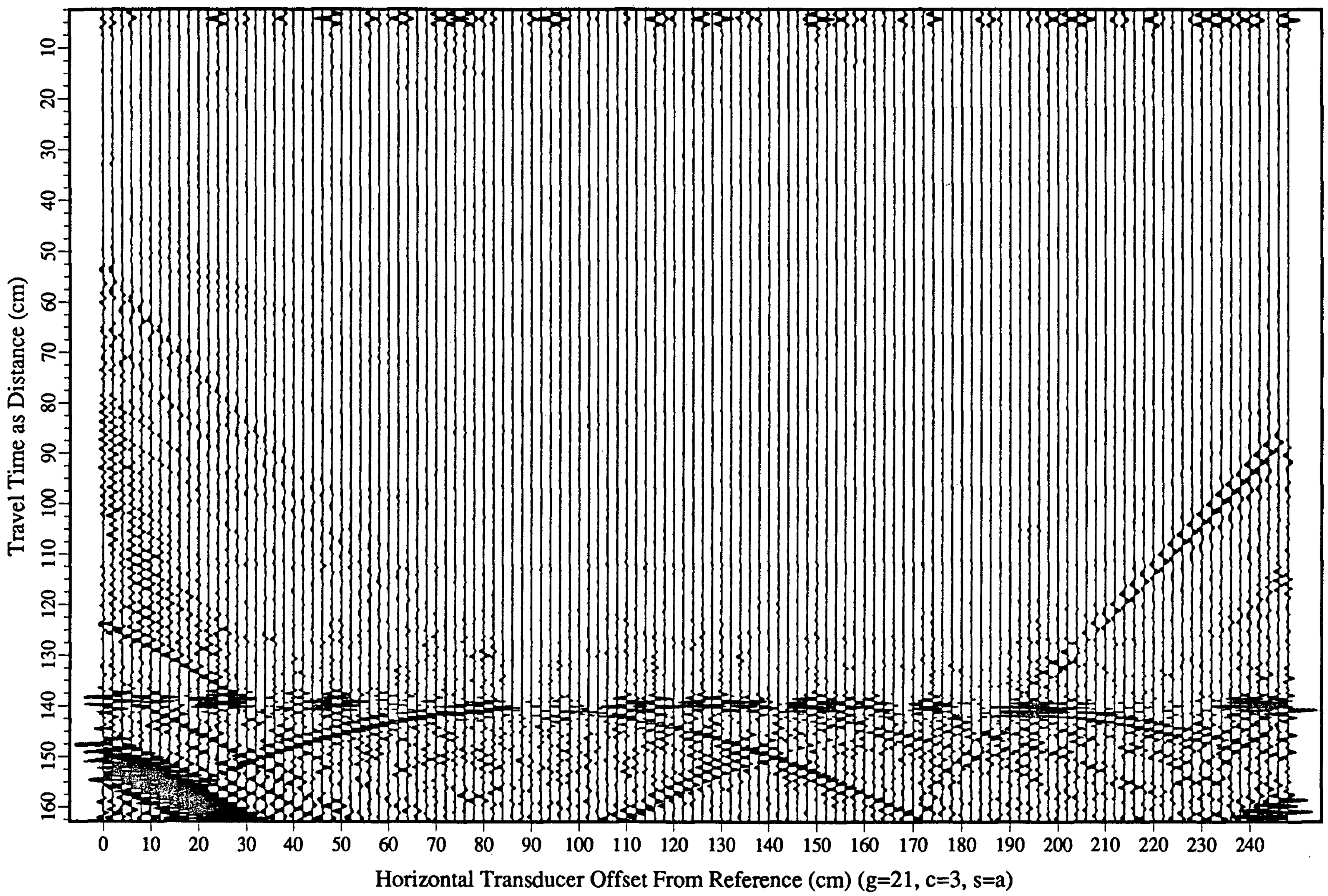

Fig. 5.5 Median Filtered Acoustic Scatter Mapping of the Vacant Target Region

locations on the wall, rather than just the one expected. The reflection from the corner of the east wall and the floor, already noticeable in the unfiltered data, became a smear of main lobes and sidelobes in the median filtered plot as the plotting routine gain was increased. Diffraction hyperbolae from the edges of the material covering the floor drain were also centered in the expected locations.

5.3 A Half Plane

The second model was constructed of a half plane, the edge of which was 59.5cm below the transducers. The half plane was constructed of a sheet of galvanized steel approximately 1.0 mm thick as shown in Fig. 5.6. The scattering solution for this model, scaled for differences in acoustic velocity and dominant frequency, was calculated by D.R. Dalton [17] and appears in Fig. 5.7. The acoustic scatter mapping of this model is shown in Fig. 5.8 and a median filtered mapping is shown in Fig. 5.9.

The relative amplitude and phase of the scattering response in the experimental case were similar to that predicted by the theoretical model. The plane reflection appeared from (0, 60) to (129, 62) in the experimental data, ending in an edge at (129, 62), the center of the diffraction hyperbola. In those locations where the experimental and theoretical data were dissimilar, limitations in the construction of the model may be sited. The slope of the plane from (0, 60) to (129, 62) was due to a slope in the floor, and the diffraction response appearing from (14, 62) to (96, 107), for example, was attributed to a joint in the sheet metal, less than 0.5 mm high.

5.4 A 13 cm Step

The third model, that of a 13 cm step, is shown in Fig. 5.10. The step was constructed of plywood with an aluminum frame. The step was chosen as a demonstrative model because it is a commonly occurring natural structure. It was also a sufficiently simple structure that its exact acoustic scattering could be calculated. The exact scattering solution, calculated by Zhang Qin

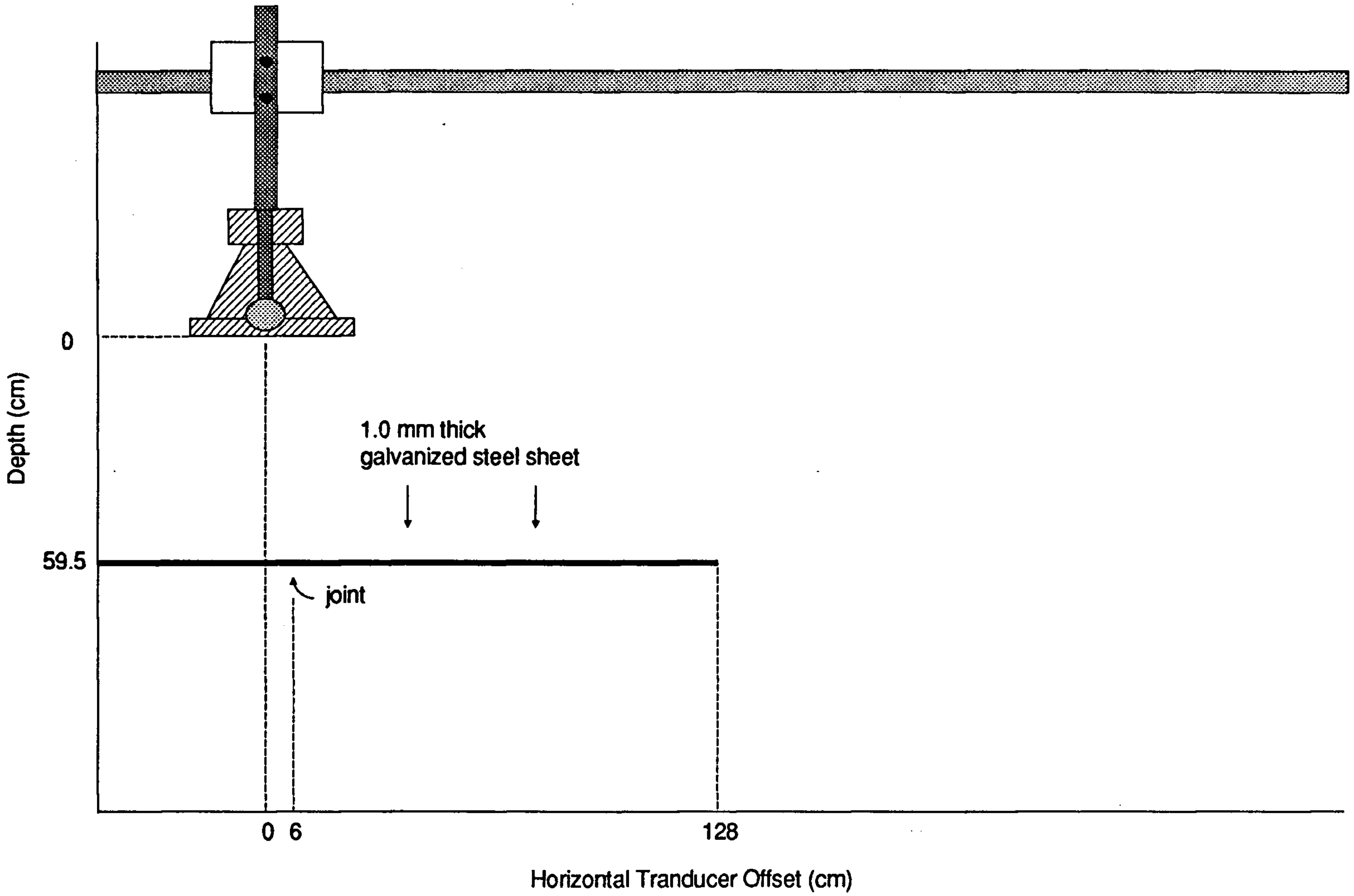

Fig. 5.6 Diagram of the Half Plane Model

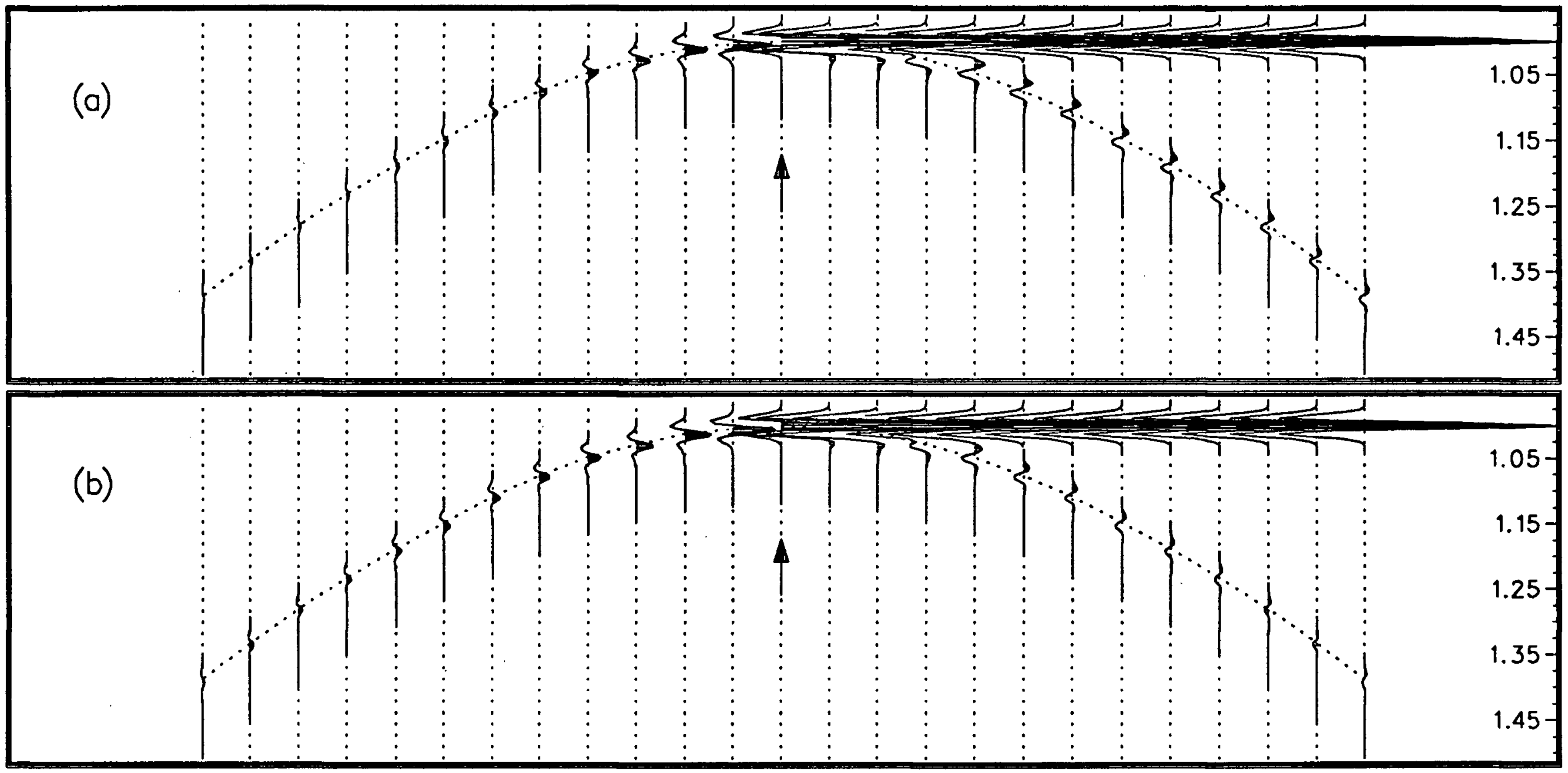

Fig. 5.7 Expected Acoustic Scatter Mapping of a Half Plane
(a) exact reflected and diffracted field, (b) Kirchoff equivalent [17]

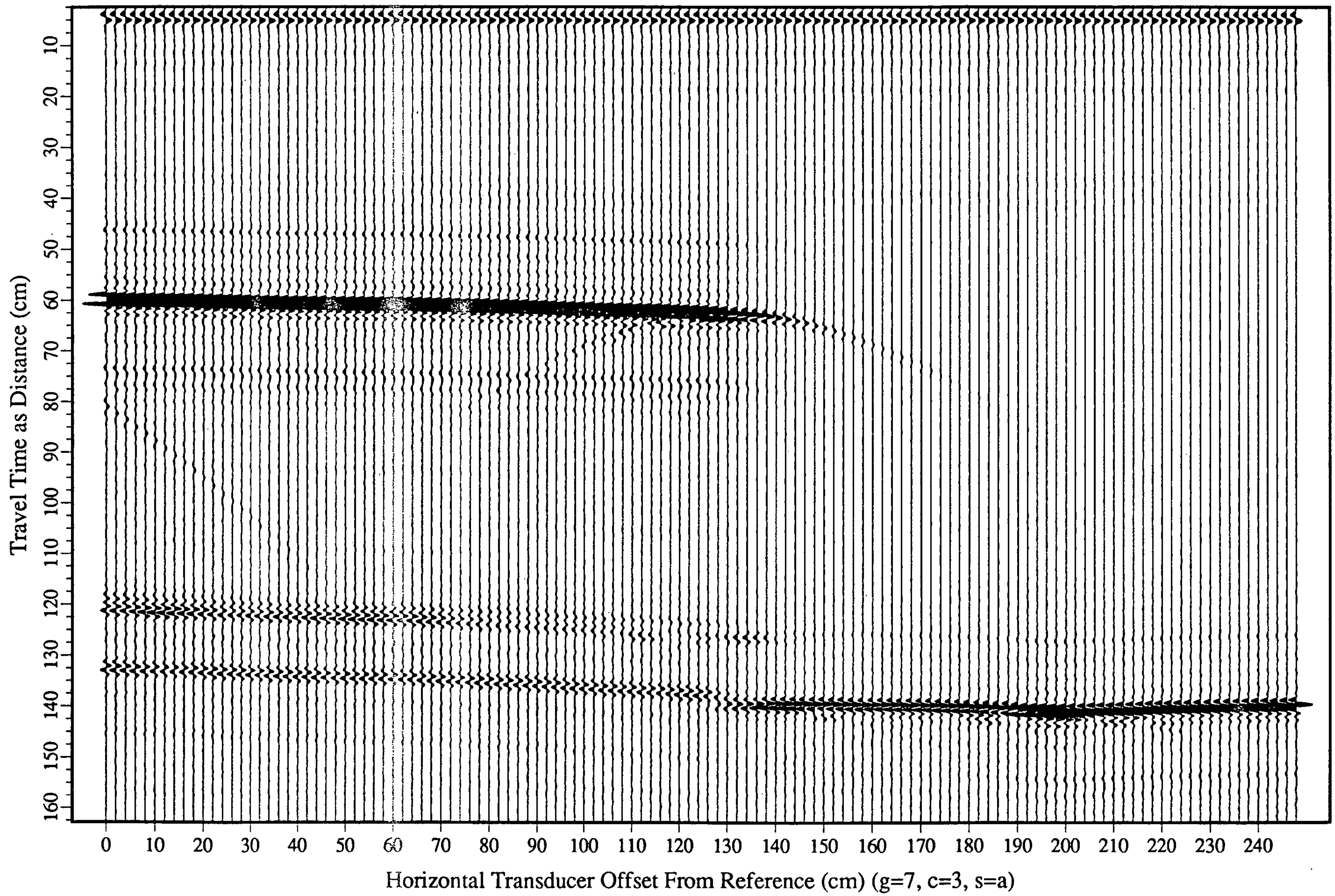

Fig. 5.8 Acoustic Scatter Mapping of the Half Plane Model

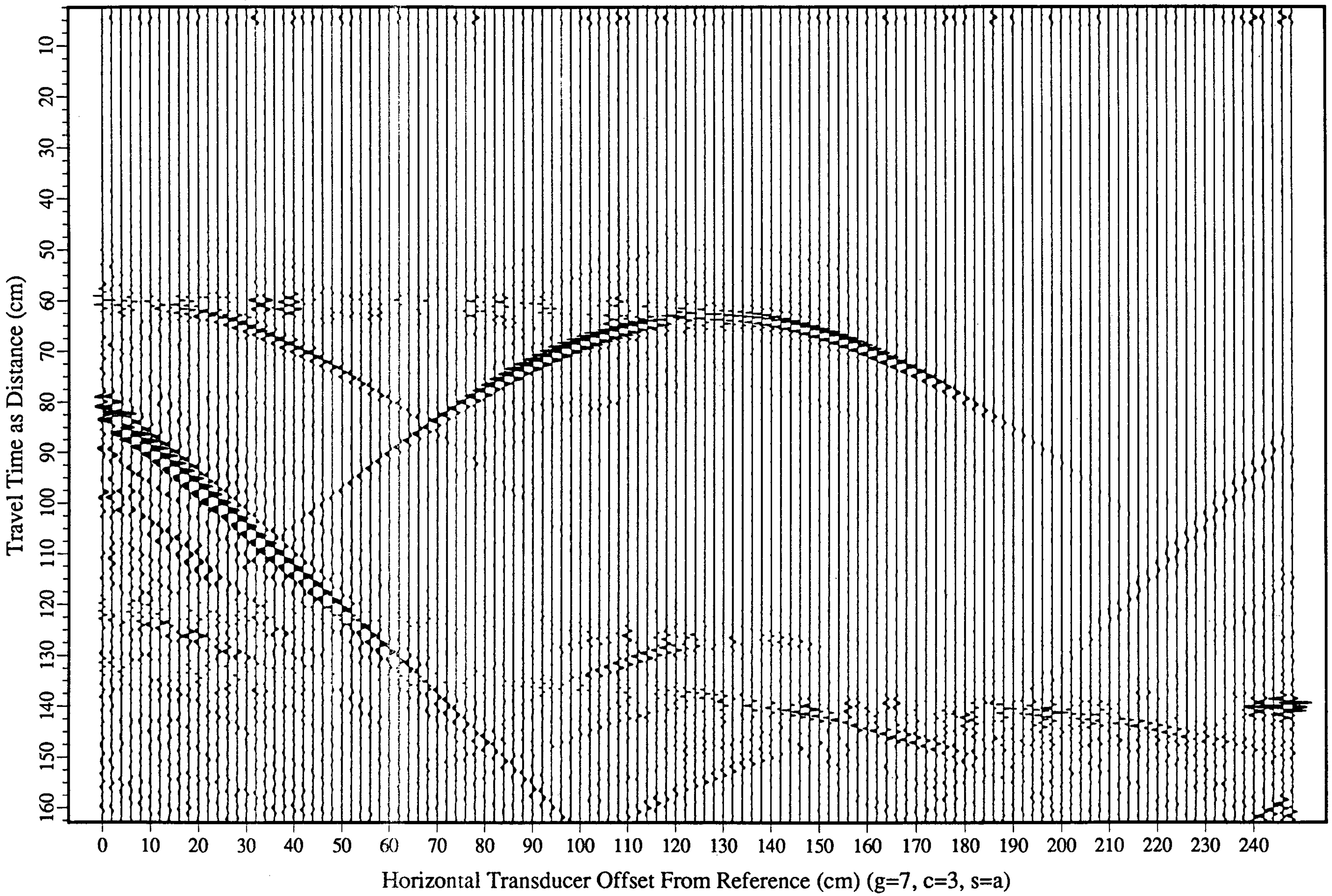

Fig. 5.9 Median Filtered Acoustic Scatter Mapping of the Half Plane Model

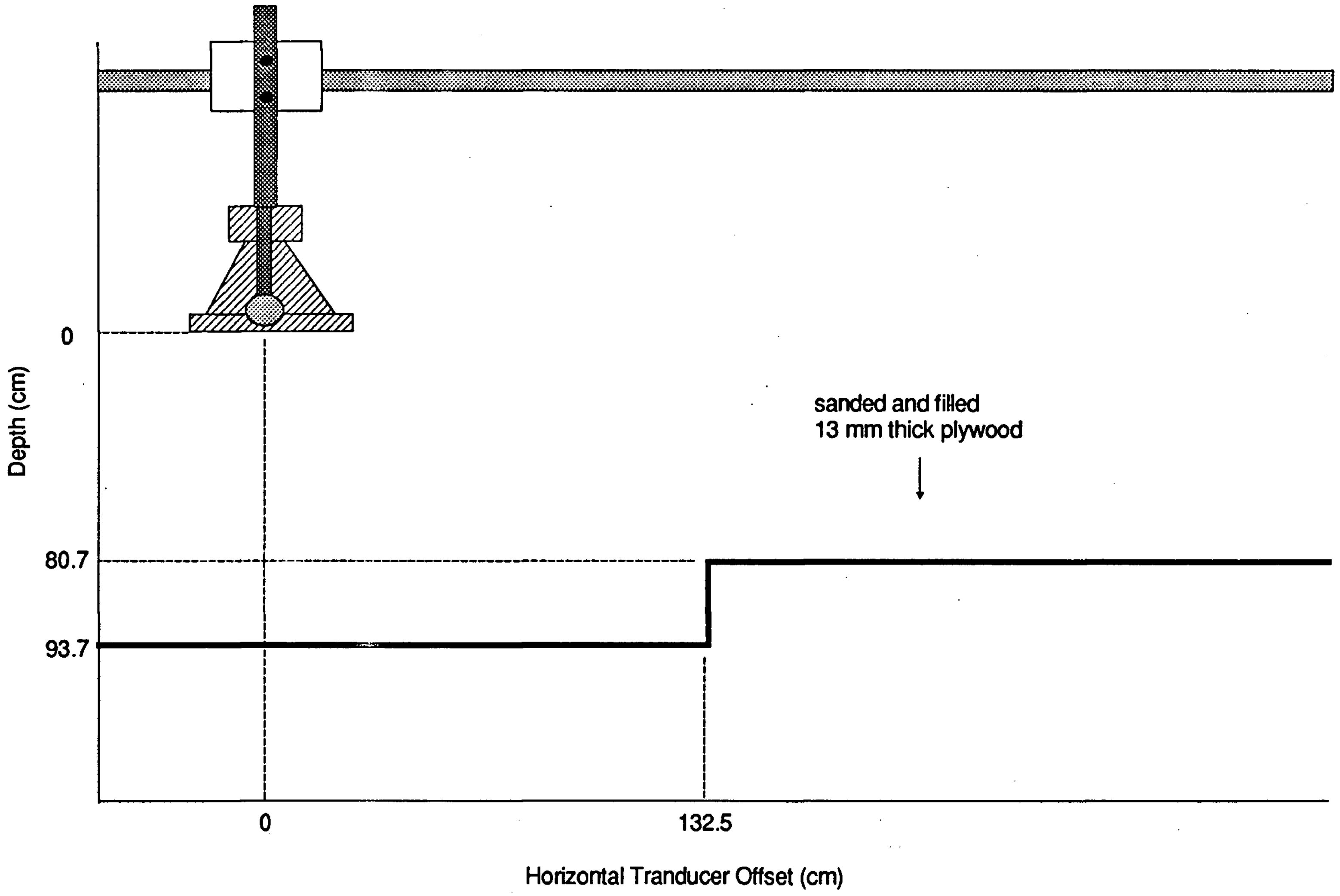

Fig. 5.10 Diagram of the 13 cm Step Model

[18], is shown in Fig. 5.11. The acoustic scatter mapping of this model is shown in Fig. 5.12 and a median filtered version of the mapping is shown in Fig. 5.13.

The relative amplitude and phase of the scattering response in the experimental case were similar to those calculated. The lower plane reflection began at (0, 94) and continued to (132, 94), where it blended into the corner reflection and diffraction centered at (132, 94). The corner reflection was the result of the source signal reflecting from face of the step and then from the lower plane. The diffraction hyperbola centered at (132, 94) was the result of a diffraction at the end of the upper plane followed by the diffracted wave reflecting from the lower plane and diffracting a second time from the end of the upper plane before being received by the microphone. One side of this diffraction hyperbola was obscured by the corner reflection. The hyperbola from a diffraction at the end of the upper plane was centered at (132, 81).

5.5 A Plane Ending in a Rounded Edge of Constant Radius

The fourth model, that of a plane ending in a curved edge of constant radius, is shown in Fig. 5.14. This model was especially interesting as the scattering solution for it, calculated by Q. Zhang [19] and appearing as Fig. 5.15, includes a creeping wave. This weak response was the predicted result of the source signal creeping along the surface of the rounded edge, diffracting on the end of the rounded edge, and creeping along the rounded surface again before arriving at the microphone. This effect results in a response from the end of the rounded edge, even when the end is not visible to the transducers.

The acoustic scatter mapping of the model is shown in Fig. 5.16. The median filtered version of the mapping, shown in Fig. 5.17, revealed a continued response beyond the 118 cm trolley position, the limit at which the end of the rounded edge is no longer visible to the transducers. The apparent duplication of the model about 59 cm below the actual model was the result of the signal returning from the surface of the model, reflecting from the loudspeaker horn, and reflecting from the surface of the model once again.

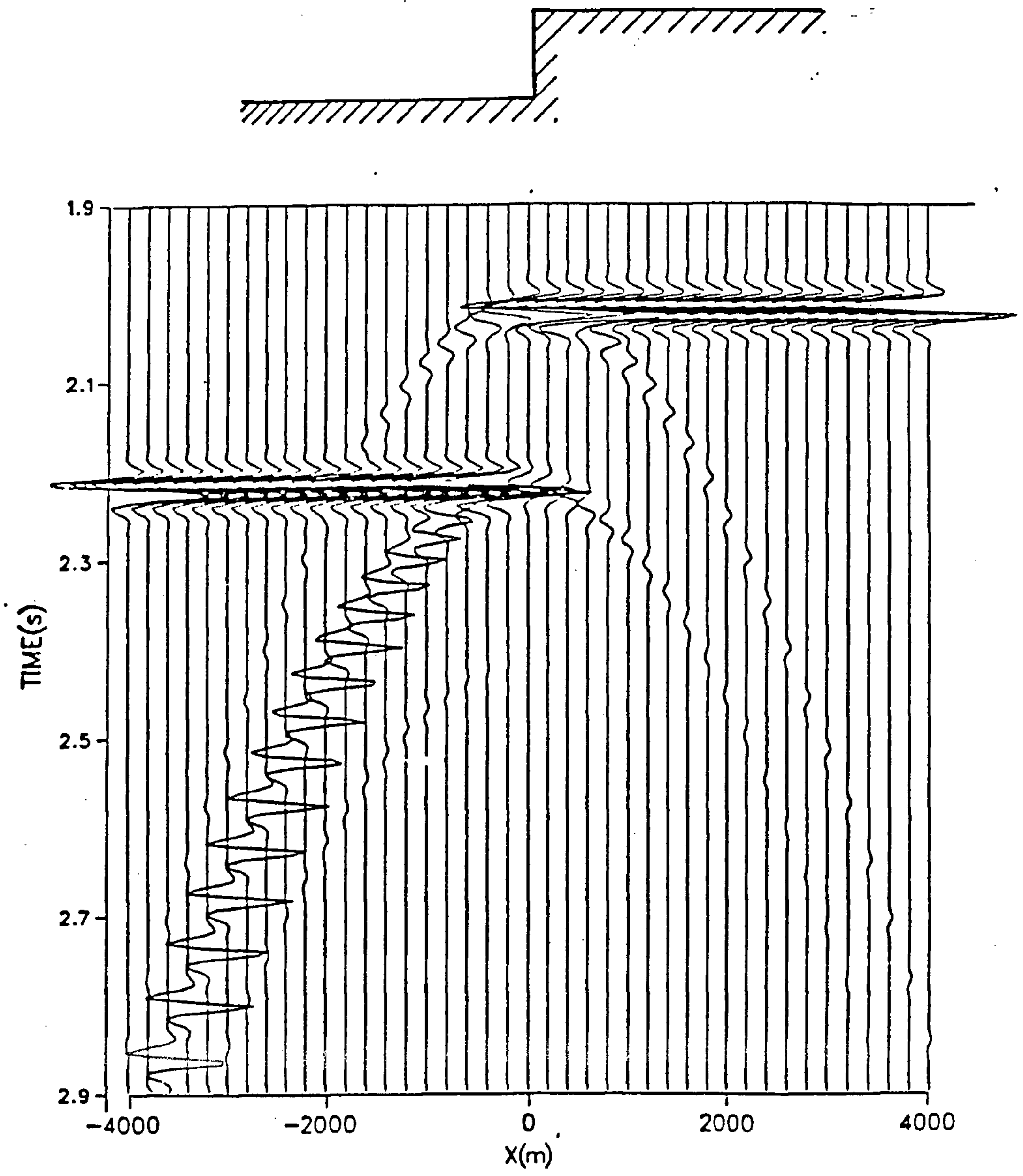

Fig. 5.11 Expected Scattering Mapping of a 13 cm Step [18]

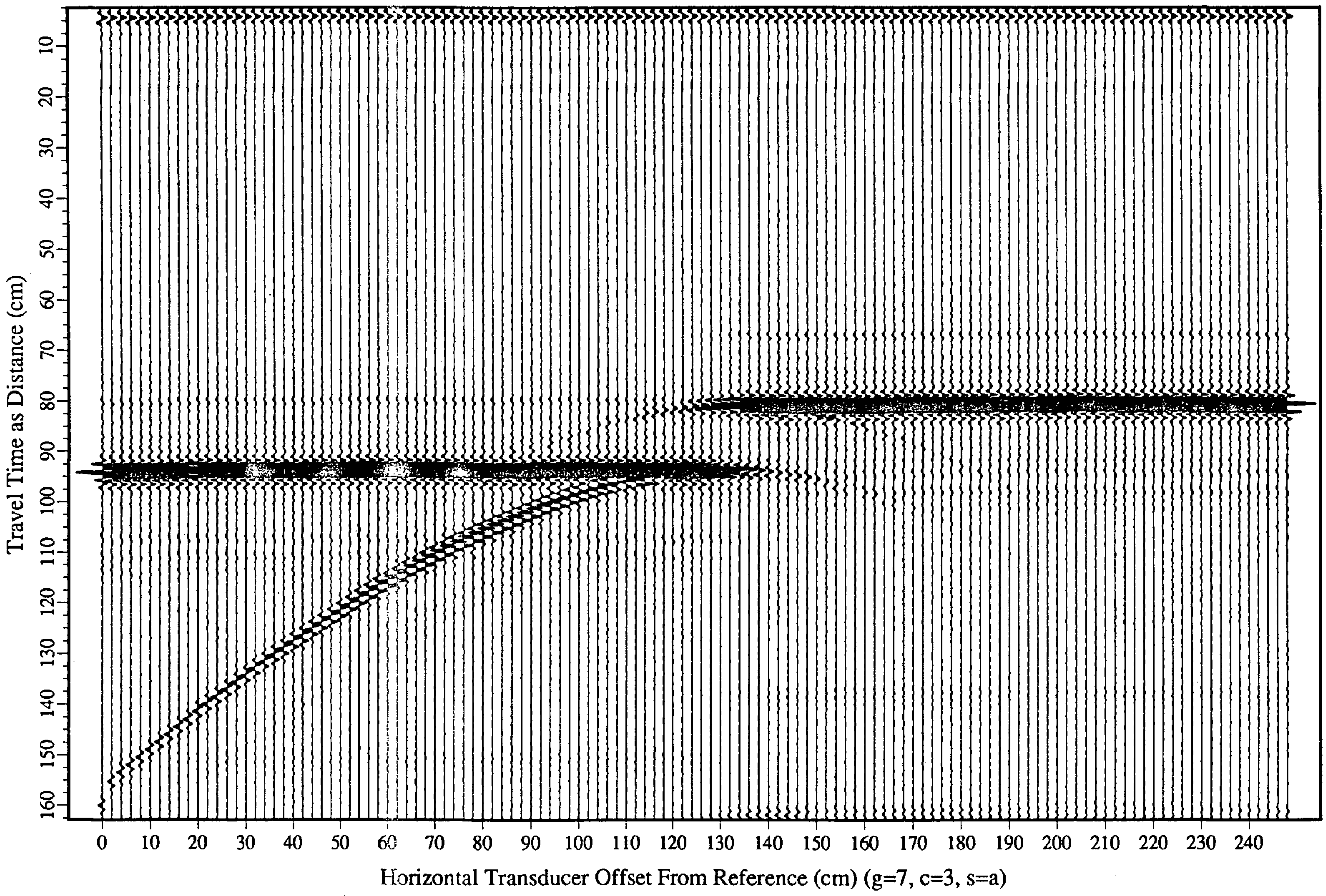

Fig. 5.12 Acoustic Scatter Mapping of the 13 cm Step

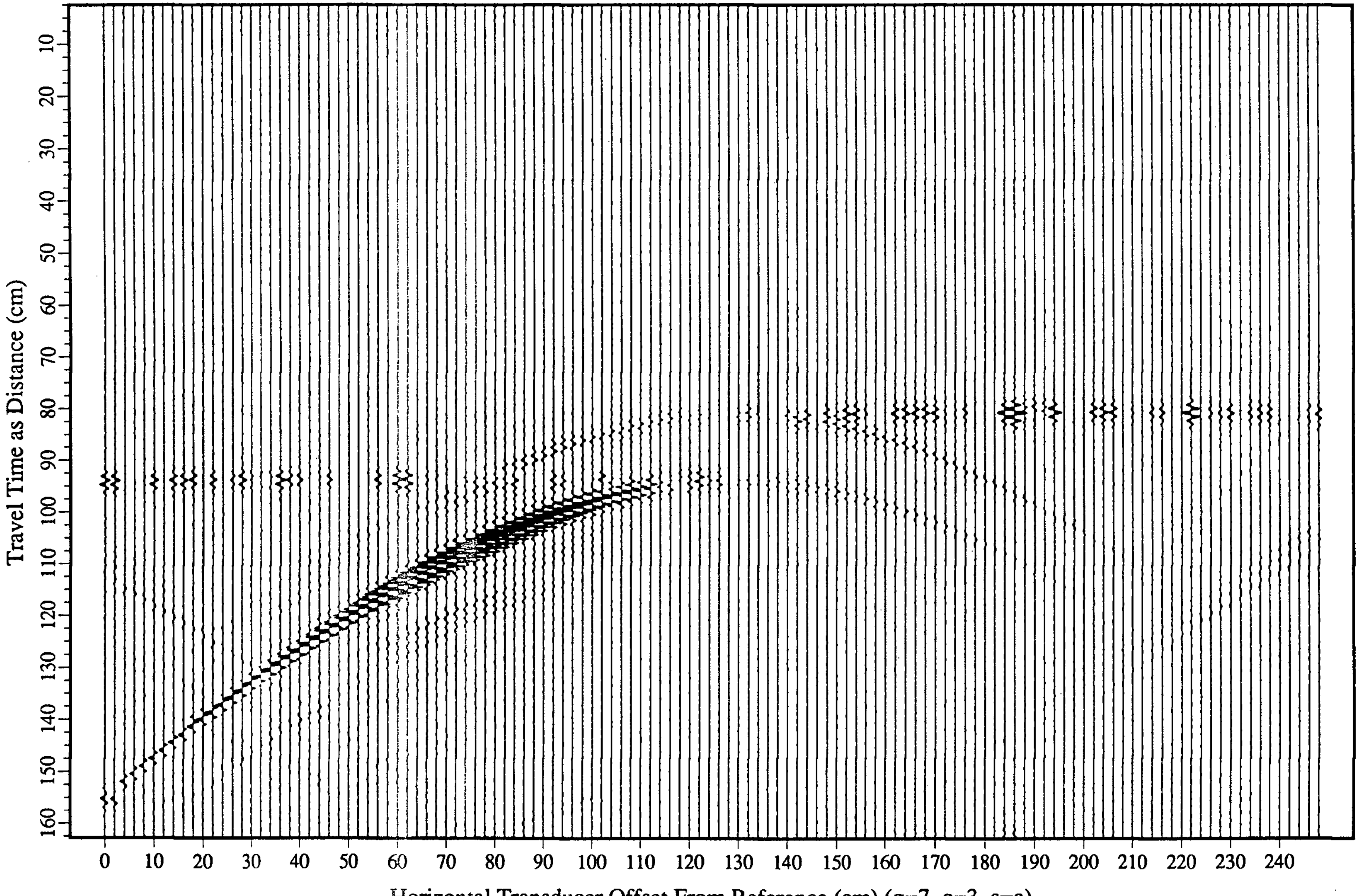

Fig. 5.13 Median Filtered Acoustic Scatter Mapping of the 13 cm Step

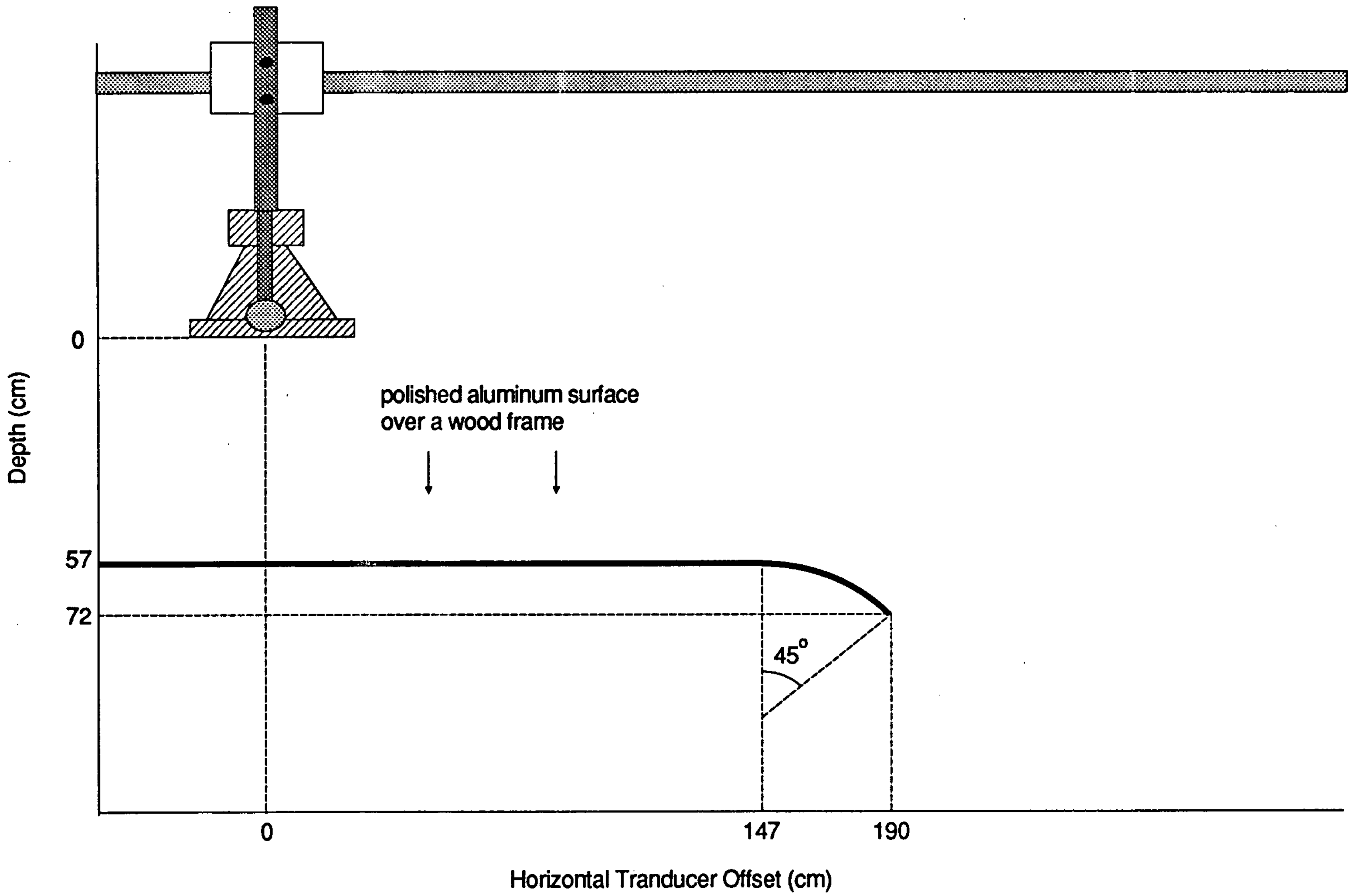

Fig. 5.14 Diagram of the Creeping Wave Test Model

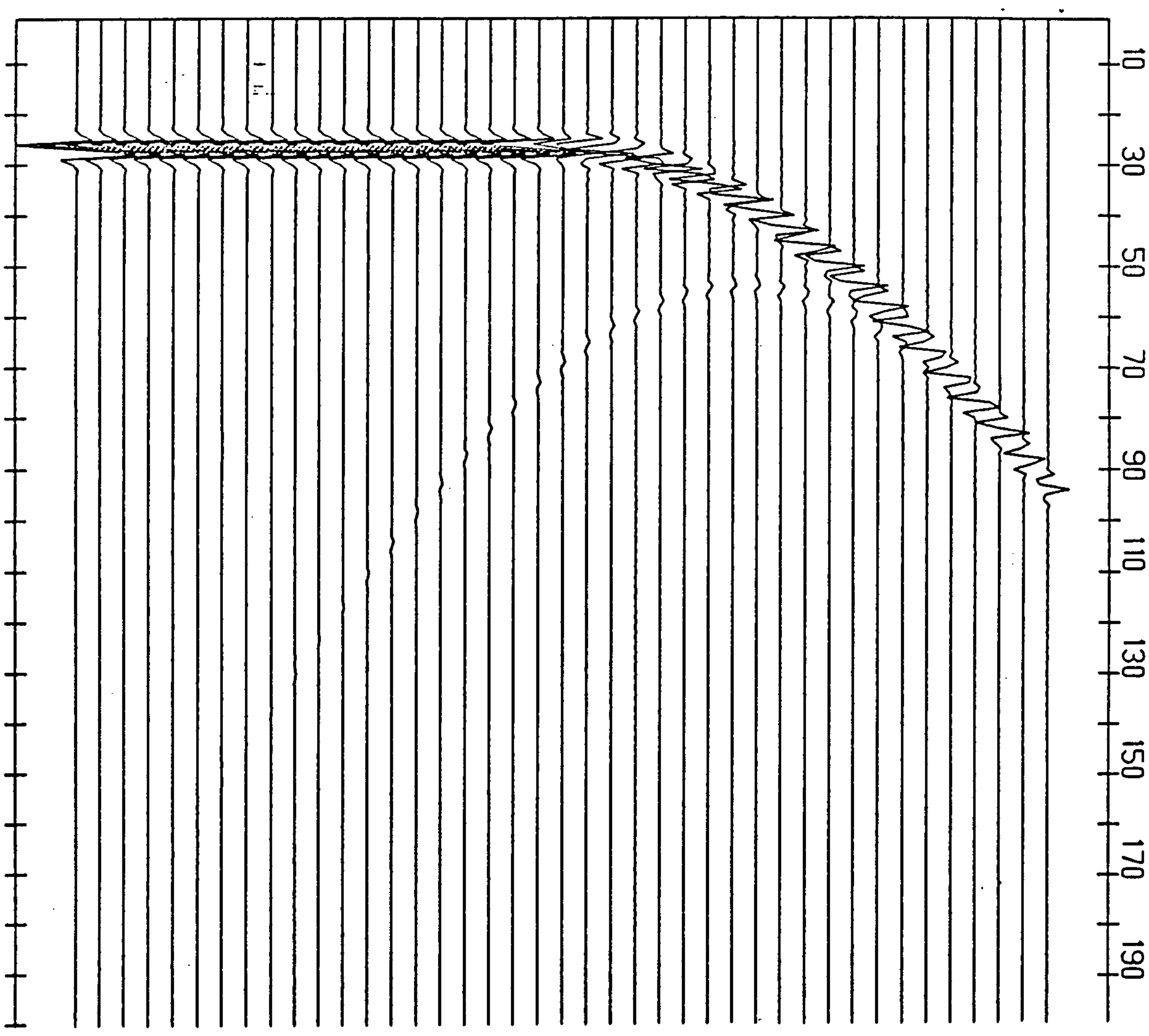

Fig. 5.15 Expected Scattering Mapping of the Creeping Wave Test Model [19]

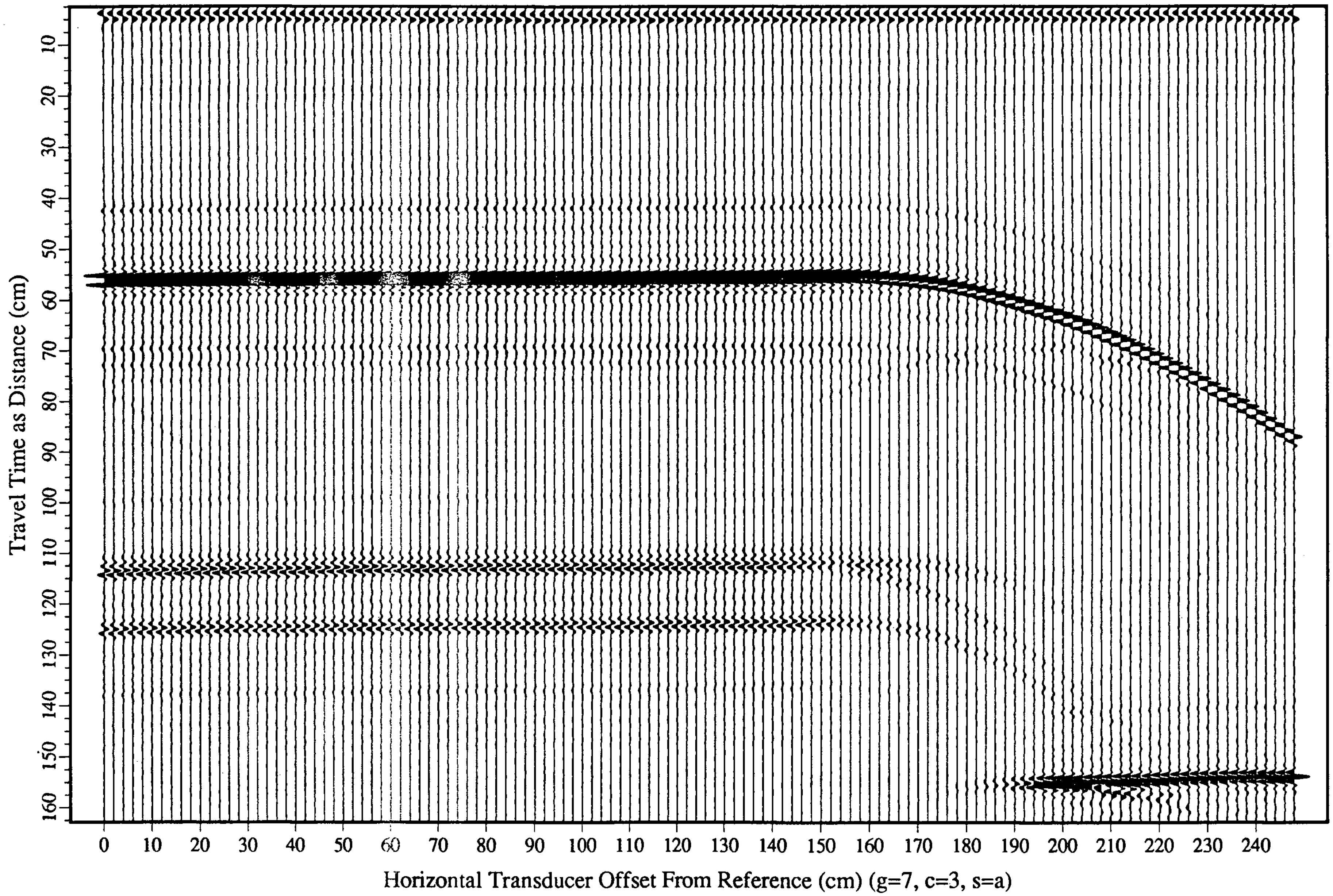

Fig. 5.16 Acoustic Scatter Mapping of the Creeping Wave Test Model

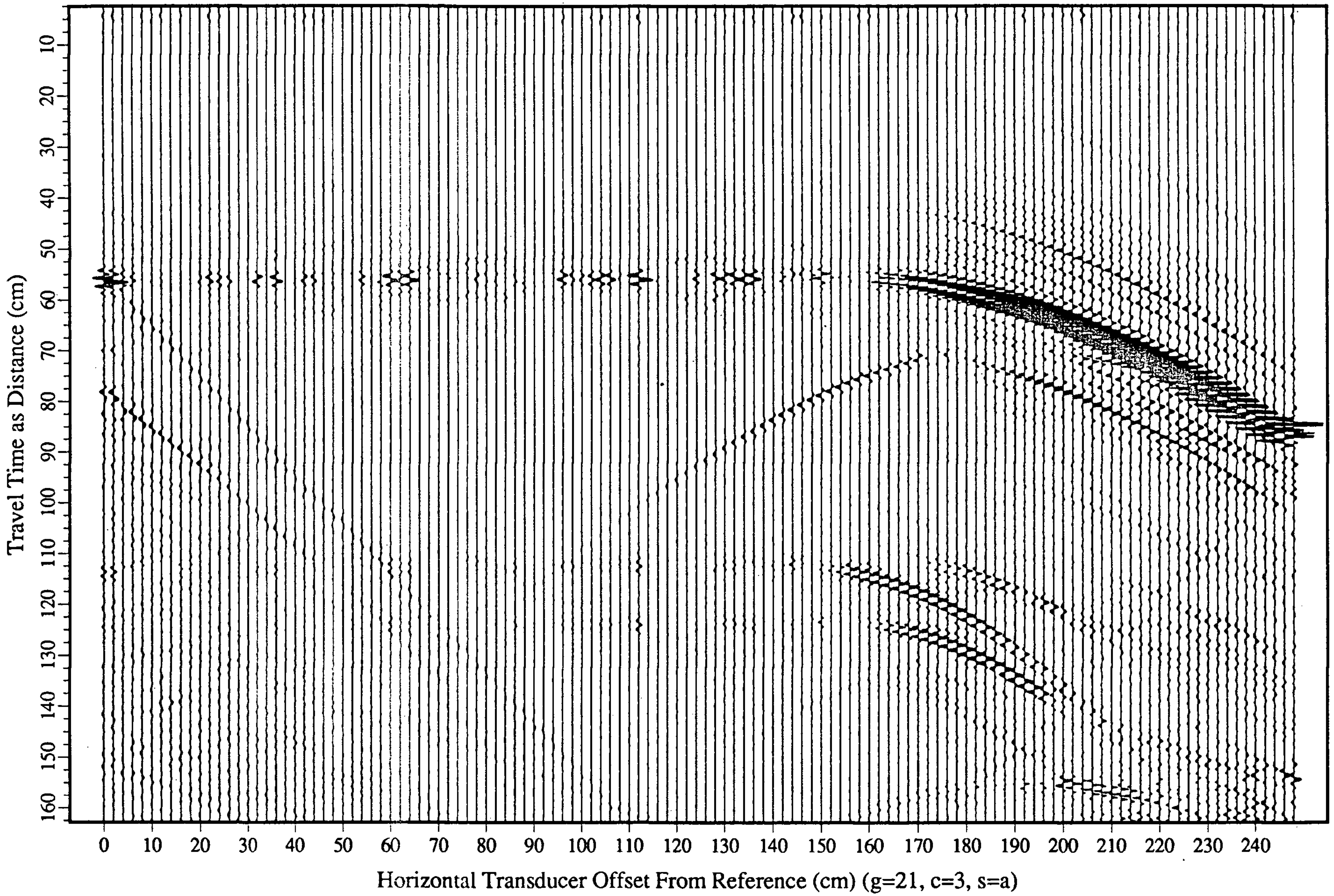

Fig. 5.17 Median Filtered Acoustic Scatter Mapping of the Creeping Wave Test Model

5.6 A Small Edge

The final model, that of single and double layers of masking tape on a smooth steel sheet is shown in Fig. 5.18. These small stacks of tape presented very small edges, a significant challenge to the resolution of the system. The single strip of tape was 0.1 mm thick and 13 mm wide and the double strip was 0.2 mm thick and 13 mm wide. Compared to the dominant wavelength of the system, 36.8 mm, these edges are 0.54% and 0.27% of a wavelength respectively. The acoustic scatter mapping of this model is shown in Fig. 5.19 and a median filtered version of the mapping is shown in Fig. 5.20.

In the unfiltered data, no effect attributable to the edges was observed. In the filtered data, however, a diffraction hyperbola centered about the location of the double layer of tape (152, 45) was plainly visible. The single layer of tape at (91, 45) did not appear to have caused any noticeable effect. Hyperbolae, attributable to diffraction by the edges of the steel sheet, were observable on both ends of the plot as were reflections due to the east and west walls.

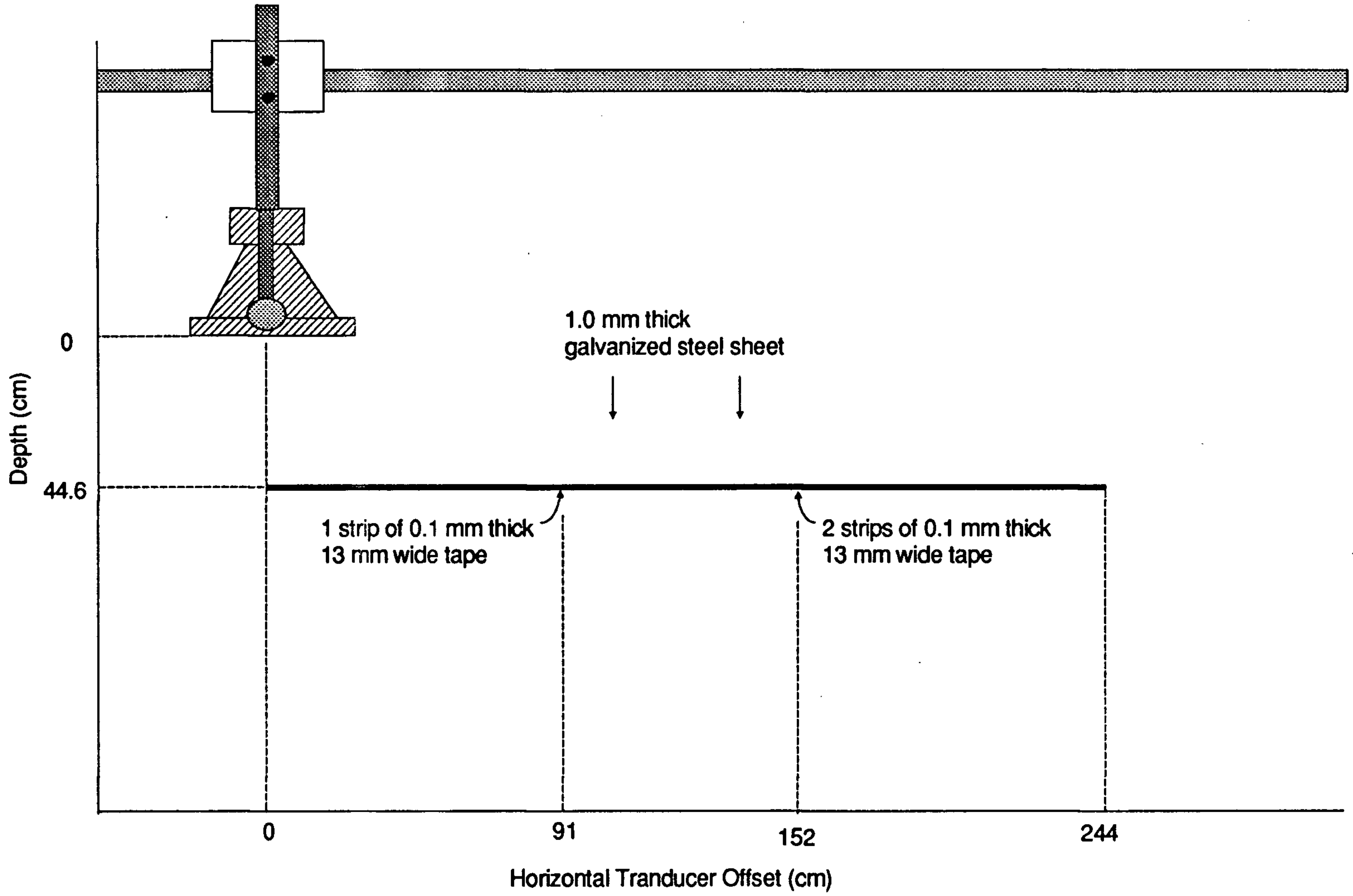

Fig. 5.18 Diagram of the Small Edge Model

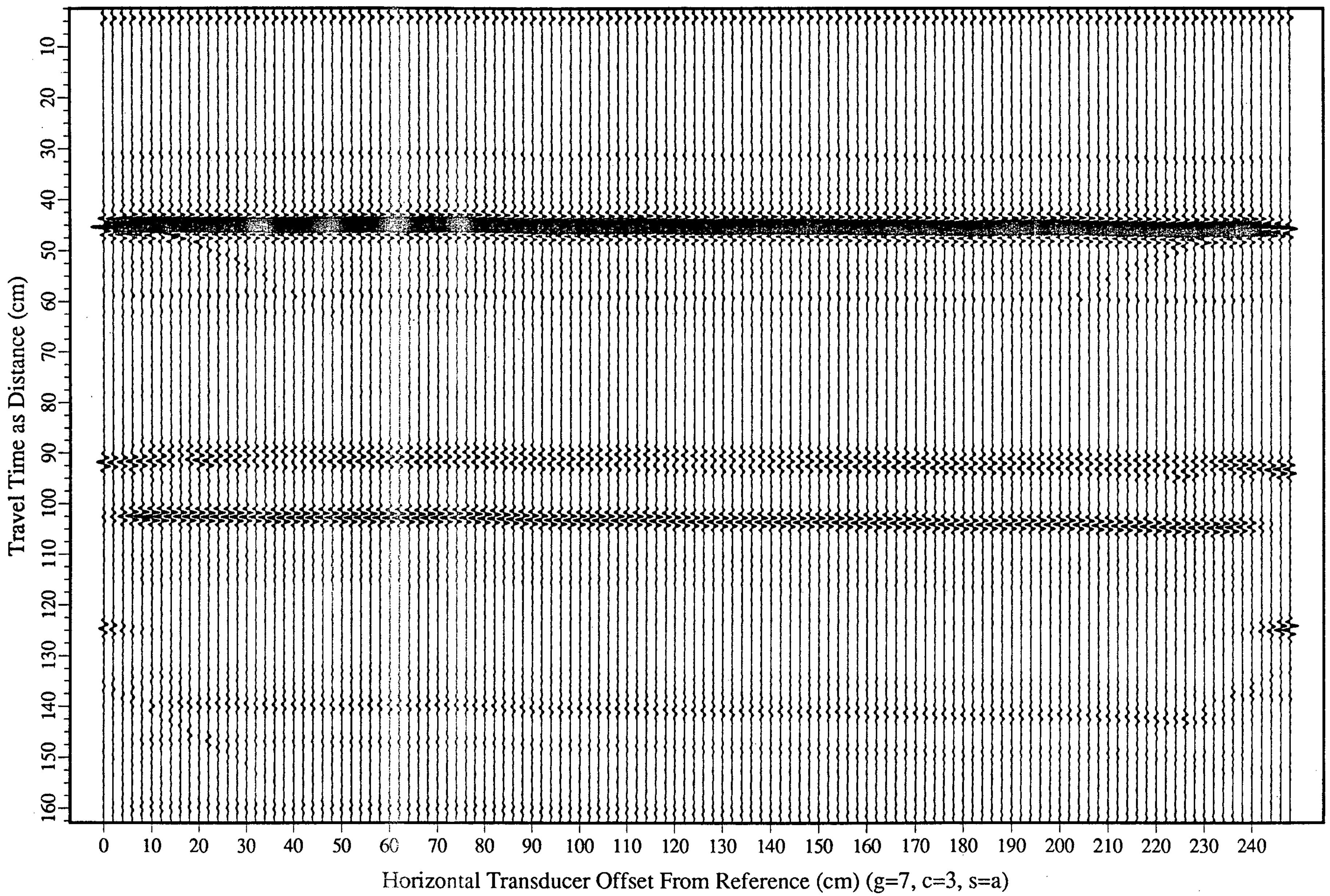

Fig. 5.19 Acoustic Scatter Mapping of the Small Edge Model

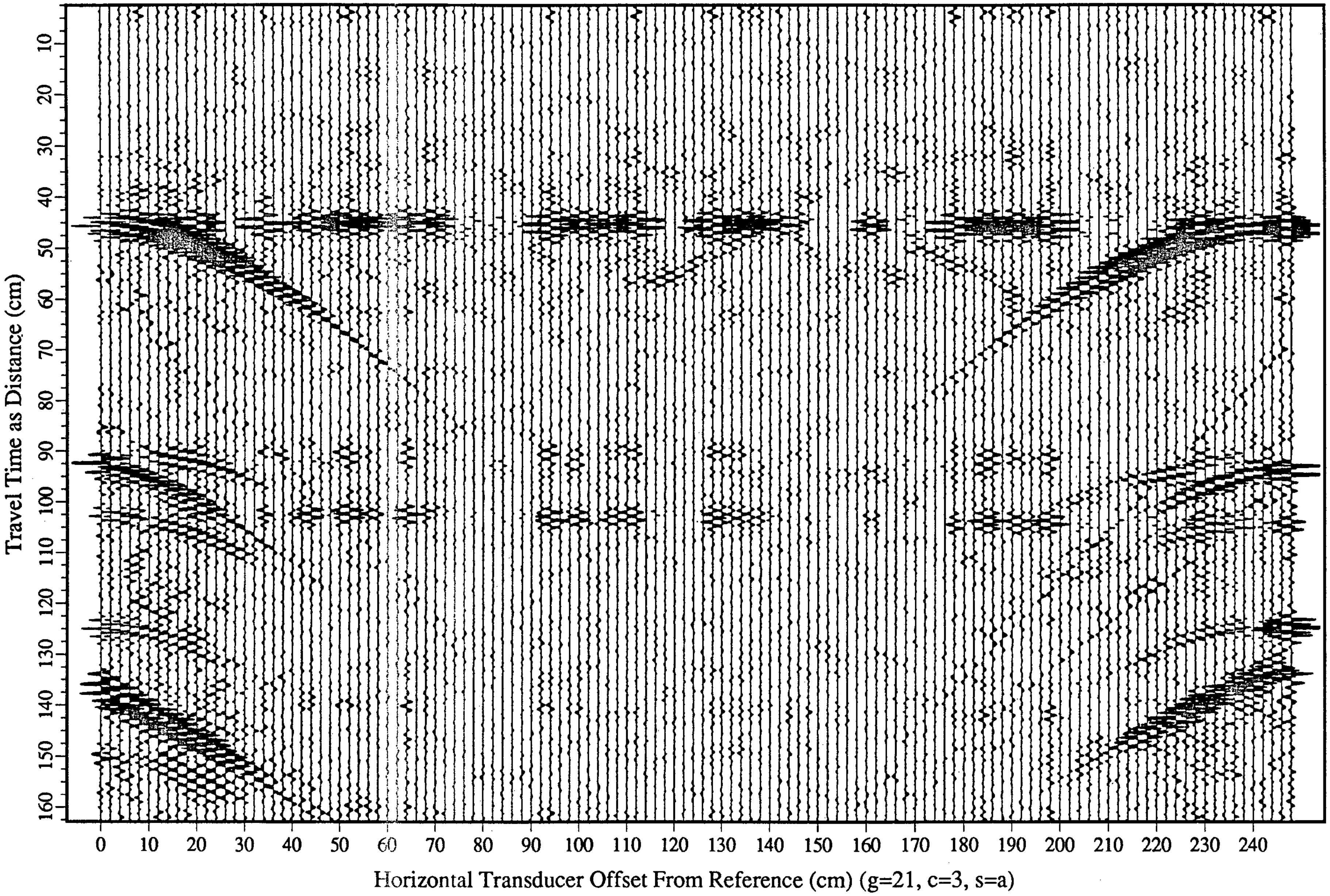

Fig. 5.20 Median Filtered Acoustic Scatter Mapping of the Small Edge Model

6 CONCLUSIONS

6.1 Summary

The described acoustic scatter-mapping system was significant for its accuracy, resolution, speed and low cost, relative to other acoustic scatter-mapping systems. The use of a cross-correlated swept-frequency source signal similar to that used in seismology and radar was novel in the area of acoustic scatter-mapping in air. The system was able to generate scatter-mappings, in less than five hours, with sufficient accuracy to distinguish between diffraction-reflection pairs less than a wavelength apart. Plotting routines for a Postscript laser printer were written as well to make a truly stand-alone system.

This acoustic scatter-mapping system, while significant in itself, presents further opportunities for a number of large and small data acquisition and image processing projects. The intent of the original system development was not the definitive acoustic scatter-mapping system, but, rather, a system which demonstrated the improvement possible using a well chosen crosscorrelated swept-frequency source signal in the particular application of acoustic scatter-mapping within the given constraints of time, cost, resolution, and flexibility.

6.2 Areas for Future Development

The images acquired to date have shown the realization of our early goals. But further progress is possible without significant redirection, and we look forward to seeing some of these improvements implemented.

The simple median filter described in Section 3.3 was useful in enhancing the visibility of diffractions by limiting the influence of reflections from horizontally continuous returns. Further processing, using other common geophysical processing techniques, greatly simplified by the availability of the finished data in a standard geophysical format, may also be applied to the acquired data, either during acquisition or post-acquisition, to further enhance the desired features of the data or test geophysical processing, interpretation, and inversion techniques.

The resolution of this system presents several opportunities for improvement. With respect to source signal calculation and generation, and received signal acquisition and processing, the system may be seen as extremely accurate, owing, predominantly, to the resolution and speed of the computer and data acquisition card. The translation of the source signal from electrical to acoustic and received signals from acoustic to electrical is also reliable. The greatest weakness of the system, with regard to accuracy, lies in its locational accuracy. That is, the positioning of the loudspeaker, microphone, and target structure with respect to the target region and to each other. This positioning problem might be reduced, but never eliminated, by using additional devices to locate each of the aforementioned objects in three dimensions. But even this would not be enough - the loudspeaker and microphone radiation patterns would have to be accurately mapped and their acoustic centers pinpointed. The trolley and overhead rail would require replacement as well. The floor on which the target structures rest is neither level nor smooth. This also should be corrected, as should the location of the walls responsible for the wall reflections seen in Figs. 5.4 and 5.5. Certainly, the possibilities are numerous. The system was sufficient to demonstrate viable techniques and useful results within the limits of their resolution. Further enhancements would improve the system but should be undertaken with an understanding of the system as a whole. The addition of an elaborate locational system would be of little benefit without an adequate control system to position the transducers and the software to operate it.

After the signal received from the microphone had been digitized, it was crosscorrelated with a reference signal to produce the finished data set. This reference signal was acquired in such a manner to expect that it was a near-perfect representation of a zero-phase signal as would be collected from a pure reflection. The indicator of a reflection in the finished data set was, therefore, a zero-phase autocorrelation of the zero-phase reference signal. The indicator of a signal returning at 45 or 225 degrees, i.e., a diffraction, was, similarly, a 45 or 225 degree autocorrelation, respectively, of the zero-phase reference signal. A question which arose concerning this point was whether one could, by crosscorrelating the finished data set with an n-phase autocorrelation of the zero-phase reference signal, automatically locate the position and amplitude of n-phase signals in the finished data set. This technique may have potential as an automated method for locating diffractions in acquired data, and, by further processing using an automated method, map some of the features responsible for the diffractions.

At the completion of this project, this acoustic scatter-mapping system was capable of acquiring useful data from models varying in the *y* and *z* dimensions so long as they were continuous in the *x* dimension. Expansion of the present trolley to reposition the transducers in both *x* and *y* dimensions would facilitate the acquisition of useful data in three dimensions. The implementation of this improvement would also require significant changes in the manner in which the acquired data would be processed and plotted.

The most gratifying improvement in this system would be the use of it to map the scattering due to specific models. This aspect has, to some extent, been taken advantage of already, but opportunity awaits to make fuller use of the system potential.

REFERENCES

[1] F.J. Hilterman, "Three-Dimensional Seismic Modeling," *Geophysics*, vol. 35, pp.1020-1037, December 1970.

[2] F.A. Angona, "Two-Dimensional Modeling and its Application to Seismic Problems," *Geophysics*, Vol. 25, pp. 468-482, 1960.

[3] F. Rieber, "Complex Reflection Patterns and Their Geologic Sources," *Geophysics*, vol. 2, pp. 132-160, 1937.

[4] S. Kamal and K.K. Sekharan, "Recent Improvements to the Physical Modeling System," *Semi-annual Progress Review: April 1983*, Houston: Seismic Acoustics Laboratory, University of Houston, April 1983.

[5] D. Khosla and G.H.F. Gardner, "Vibroseis Data Acquisition in the Modeling Tank," *Semi-annual Progress Review: April 1983*, Houston: Seismic Acoustics Laboratory, University of Houston, April 1983.

[6] B.B. Narod and M.J. Yedlin, "A Basic Diffraction Experiment for Demonstrating the Geometrical Theory of Diffraction," *American Journal of Physics*, vol. 54, pp. 1121-1126, December 1986.

[7] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing,* Englewood Cliffs, New Jersy: Prentice-Hall, 1975.

[8] M.J. Yedlin, I.F. Jones and B.B. Narod, "Application of the Karhunen-Loeve Transform to Diffraction Separation," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. Aspp-35, pp. 2-8, January 1987.

[9] D. Davis and C. Davis, *Sound System Engineering*, Indianapolis, Indiana: Howard W. Sams & Co., 1975.

[10] L.E. Kinsler et al. *Fundamentals of Acoustics*, New York: John Wiley & Sons, 1982.

[11] A.W. Trorey, "A Simple Theory for Seismic Diffractions," *Geophysics*, vol 35, pp. 762-784, 1970.

[12] J.M. Crawford, W.E.N. Doty and M.R. Lee, "Continuous Signal Seismograph," *Geophysics*, vol. 25, pp. 95-105, February 1960.

[13] J.R. Klauder, A.C. Price, S. Darlington and W.J. Albersheim, "The Theory and Design of Chirp Radars," *Bell System Technical Journal*, vol. 39, pp. 745-808.

[14] W.H. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*, New York: Cambridge University Press, 1988.

[15] O. Koefoed, "Aspects of Vertical Seismic Resolution," *Geophysical Prospecting*, vol. 29, pp. 21-30, 1981.

[16] P.L. Goupillaud, "Signal Design in the Vibroseis Technique," *Geophysics*, vol. 41, pp. 1291-1304, December 1976.

[17] D.R. Dalton and M.J. Yedlin, "Time Domain Solutions for Diffraction of Acoustic Waves by a Half Plane," Surveys in Geophysics vol. 11, (to be published).

[18] Q. Zhang and E.V. Jull, "Pulse Diffraction by a Step Discontinuity," *IEEE Antennas and Propagation Society International Symposium*, pp. 146-149, June 1988.

[18] Q. Zhang and E.V. Jull, "Acoustic Pulse Diffraction by Step Discontinuities on a Plane," *Geophysics*, June 1990.

[19] Q. Zhang and E.V. Jull, "Acoustic Pulse Diffraction by a Curved Half Plane," 1989 URSI International Symposium on Electromagnetic Theory, Stockholm, pp. 263-265, August 1989.

BIBLIOGRAPHY

A. Berry, "Controlled Experiments of the Diffraction of Sound by a Curved Surface," *Journal of the Acoustical Society of America*, vol. 83, pp. 2047-2058, 1988.

V. Cerveny and R. Ravindra, *Theory of seismic head waves*, Toronto: University of Toronto Press, 1971.

C.B. Chinoy and P.T. Lewis, "Acoustical Signal Processing: FFT Convolution and Deconvolution," Acoustics Letters, vol. 8, pp. 120-127, 1985.

J.F. Claerbout, *Fundamentals of Geophysical Data Processing: With Applications to Petroleum Prosepecting,* New York: McGraw-Hill, 1976.

W.H. Dragoset, "Marine Vibrators and the Doppler Effect," *Geophysics*, vol. 53, pp. 1388-1398, 1988.

B.M. Gurbuz, "Signal Enhancement of Vibratory Source Data in the Presence of Attenuation," *Geophysical Prospecting*, vol. 20, pp. 421-438, 1972.

F.J. Hilterman, "Amplitudes of Seismic Waves - A Quick Look," *Geophysics*, vol. 40, pp. 745-762, 1975.

F.J Hilterman, "Seismic Imaging," *Acoustic Imaging: Visualtization and Characterization: Volume 9*, New York: Plenum Press, 1980.

G.M. Hoover, "Acoustical Holography using Digital Processing," *Geophysics*, vol. 37, pp. 1-19, 1972.

F.K. Levin and H.C. Hibbard, "Three Dimensional Seismic Model Studies," *Geophysics*, vol. 20, pp. 19-32, 1955.

A.F Linville and M.A. Dablain, "A Case Study in Seismic Diffraction Near a Gulf Coast Salt Dome," *Geophysics*, vol. 50, pp. 1077-1082, 1985.

L.A. Mayer, and L.R. LeBlanc, "The Chirp Sonar: A New, Quantitative High Resolution Sub-Bottom Profiling System," Pace, N.G. Ed., *Acoustics and the Sea Bed*, Bath: Bath University Press, 1983.

D.R. Pant and S.A. Greenhalgh, "Lateral Resolution in Seismic Reflection - a Physical Model Study," *Geophysical Journal*, vol 97, pp. 187-198, 1989.

A. Paul and M. Campillo, "Diffraction and Conversion of Elastic Waves at a Corrugated Interface," *Geophysics*, vol. 53, pp. 1415-1424, 1988.

L.R. Rabiner and R.W. Schafer, *Digital Processing of Speech Signals*, Englewood Cliffs, New Jersy: Prentice-Hall, 1978.

E. Reitsch, "Computerized Analysis of Vibroseis Signal Similarity," *Geophysical Prospecting*, vol. 25, pp. 541-552, 1977.

E. Reitsch, "Vibroseis Signals With Proscribed Power Spectrum," *Geophysical Prospecting*, vol. 25, pp. 613-620, 1977.

R.A. Roberts and C.T. Mullis, *Digital Signal Processing,* Reading, Massachusetts: Addison-Wesley, 1987.

T.D. Rossing, *The Science of Sound,* Reading, Massachusetts: Addison Wesley Publishing Co., 1982.

S.G. Schock, L.R. LeBlanc and L.A. Mayer, "Chirp Subbottom profiler for Quantitative Sediment Analysis," *Geophysics,* vol. 54, pp. 445-450, 1989.

H. Taub and D.L. Schilling, *Principles of Communication Systems,* 2nd Ed., New York: McGraw-Hill Book Company, 1986.

M.J. Yedlin and D.R. Dalton, *Diffraction Coefficient Formulation of an Exact Solution for Acoustic Diffraction by a Half Plane,*

APPENDIX A Software Used by This System

The software used in operation of the system included the following programs, the source code of which follows these descriptions.

Dotplot.c - This program collects information from the operator in a question and answer format and uses this information to print the specified image on a Roland PR-1112 (72 dpi) dot matrix printer.

Ecko.asm - This subroutine expects a pointer to and the size of an array of integer data. The specified array is copied to the digital to analogue convertor on the data acquisition card.

Flotuint.c - This program will convert a designated floating point binary format file, as produced by *Multrace.c*, to a designated unsigned integer binary format file, as used by all other system software.

Lasrplot.c - This program collects information from the operator in a question and answer format and uses this information to generate a Postscript file containing a plot of the specified image. The Postscript file is normally sent, by the operator, to a (300 dpi) laser printer.

Median3.c - This program will three-point space-median filter a designated source image, string the result in the designated destination file.

Multrace.c - This is the heart of the system. It uses Ecko.asm to send out the source signal, collects the returning signal, moves the trolley, crosscorrelates the collected data with a reference signal and stores the result in the designated file.

Sourtran.c - This program will send out a source signal, collect the returning signal, and store the collected signal as the system reference signal.

Uintasci.c - This program will convert a designated unsigned integer binary format file to a designated ASCII format file.

Wavegen.c - This program will, based on the constants stored in it, calculate a source signal and store it as the system source signal file.

```
/* dotplot.c */
/****************************************************/
/*                                                  */
/*           DOT matrix PLOTting routines           */
/*           by Garfield Mellema                    */
/*           November 16, 1988                      */
/*                                                  */
/****************************************************/
/*                                                  */
/*     This program will, after asking the user a    */
/*     few relevant questions, print data read from */
/*     a specified file.  The data in the file must  */
/*     be arranged in consecutive blocks of equal,   */
/*     specified length.  The blocks are printed as  */
/*     a series of line graphs, each block a new     */
/*     line.  The line graph lengths are scaled, as  */
/*     much as possible, to fill a printer line.     */
/*     Titles may be included between the blocks     */
/*     and will be printed on the screen as the      */
/*     block is being read.  A title may be printed */
/*     at the top of the graphs.  Portions of the    */
/*     data blocks may be skipped over, as well as   */
/*     groups of data blocks.  The graphs are  */
/*     normalized either individually or globally,   */
/*     and their maximum amplitude is specified      */
/*                                                  */
/****************************************************/

#include <stdio.h>
#include <math.h>

unsigned char lines[14][16384];
unsigned int newdata[16384];
unsigned int twopow[2][9] = {{1, 3, 7, 15, 31, 63, 127, 255, 255},
                             {1, 2, 4, 8, 16, 32, 64, 128, 0}};

void main()
{
      double zero;
      float minamp, maxamp, globemin, globemax, scale;
      int i, j, k, l, linenumb[16], tracleng, tracnumb, traceamp;
      int newline[5], skipleng, endleng, skiptrac, base, bscale, escale;
      char title[100], filename[80], ifscale[5], gloscale[5];
      FILE *fi;

      printf ("\nMultiple trace plotting software");
      printf ("\nwritten by Garfield Mellema (November 16, 1988)");
      printf ("\n\nThis program will read unsigned integer data (0-65535) from ");
      printf ("a specified\nbinary file and plot the data as a series of ");
      printf ("adjacent traces on a\nRoland PR-1112 (or equivalent) printer.");
      printf ("\n\nEnter the name of the source data file\n");
      scanf ("%s", filename);
      printf ("How many points are in each trace (maximum is 16384)?\n");
      scanf ("%d", &tracleng);
      if (tracleng > 16384)
            tracleng = 16384;
      printf ("The first point read will be number 0.");
printf ("  Begin plotting at point number?\n");
scanf ("%d", &skipleng);
if (skipleng > tracleng - 1 || skipleng < 0)
      skipleng = 0;
printf ("Stop plotting at point number?\n");
scanf ("%d", &endleng);
if (endleng > tracleng - 1 || endleng < skipleng)
      endleng = tracleng - 1;
printf ("How many traces are in the file?\n");
scanf ("%d", &tracnumb);
printf ("Print every 'n'th trace.  What is 'n'?\n");
scanf ("%d", &skiptrac);
printf ("How many additional traces should the maximum positive value");
printf ("\nof each trace overlap?  The maximum number is 3.\n");
scanf ("%d", &traceamp);
if (traceamp > 3)
      traceamp = 3;
if (traceamp < 0)
      traceamp = 0;
printf ("Do you want positive shading?  (Y/N)\n");
scanf ("%s", gloscale);
if (gloscale[0] != 'Y' && gloscale[0] != 'y'){
      for (i=0; i<=8; i++)
            twopow[0][i] = twopow[1][i];
}
fputc (27, stdprn);                       /* initialize  */
fputc (64, stdprn);                       /* the printer */
fputc (27, stdprn);                       /* set it for  */
fputc (77, stdprn);                       /* elite type  */
printf ("Do you wish to print a title above the plot? (Y/N)\n");
scanf ("%s", gloscale);
if (gloscale[0] == 'Y' || gloscale[0] == 'y'){
      printf ("Enter the title as you want it printed.  To stop printing ");
      printf ("the title\nbegin a line with ^.\n");
      gets(title);
      fputc (27, stdprn);                 /* set the printer for */
      fputc (97, stdprn);                 /* centered printing   */
      fputc (1, stdprn);
      do  {
            gets (title);
            if (title[0] != '^')
                  fprintf (stdprn, "%s\n", title);
      } while (title[0] != '^');
      fputc (27, stdprn);                 /* set the printer to begin */
      fputc (97, stdprn);                 /* printing at the left margin */
      fputc (0, stdprn);
}
printf ("Do you want to print a scale above the plot? (Y/N)\n");
scanf ("%s", ifscale);
printf ("Each trace is individually normalized.  Do you prefer global ");
printf ("scaling?  (Y/N)\n");
scanf ("%s", gloscale);
globemax = -1;
globemin = 65536;
fi = fopen (filename, "rb");
if (fi == NULL){
      printf ("\n***********   file opening error   *******************");
```

```
            exit (0);
      }
      if (gloscale[0] == 'Y' || gloscale[0] == 'y'){
            printf ("Begin scaling at point number?\n");
            scanf ("%d", &bscale);
            printf ("End scaling at point number? \n");
            scanf ("%d", &escale);
            printf ("\nBegin scaling\n");
            for (i=0; i<tracnumb; i++){
                  printf ("Scaling trace number %6d", i);
                  putch (13);
                  fread (newdata, sizeof (unsigned int), tracleng, fi);
                    /* read in the data from the specified file and find its minimum
*/
                  for (j=bscale; j<=escale; j++){          /* and maximum */
                        globemax = (globemax>newdata[j]) ? globemax : newdata[j];
                        globemin = (globemin<newdata[j]) ? globemin : newdata[j];
                  }
            }
            rewind (fi);
            printf ("\nFinished scaling");
            printf ("\nglobemax is %6.0f", globemax);
            printf ("\nglobemin is %6.0f", globemin);
      }
      fprintf (stdprn, "\n");
      for (i=0; i<14; i++){                                    /* erase the trace
plotting array */
            for (j=0; j<16384; j++)
                  lines[i][j] = 0;
      }
      for (i=0; i<16; i++)
            linenumb[i] = i;
      fputc (27, stdprn);                                      /* set paper feed to
n/216 inch */
      fputc (51, stdprn);
      fputc (24, stdprn);                                      /* where n = 24 */
      newline[0] = 27;                                               /* Bit image
mode */
      newline[1] = 42;                /* calibrated for 5 usec sampling  at 331 m/s
*/
      fprintf (stdprn, "\n%41s DISTANCE (mm)\n", " ");
      if (endleng - skipleng < 464){                     /* standard density
graphics       */
            printf ("\n465 dpl graphics\n");
            if (ifscale[0] == 'Y' || ifscale[0] == 'y'){   /* print a scale above
*/
                  fprintf (stdprn,"\n   0        41        83        124       166
");
                  fprintf (stdprn,"     207       248       290       331
372\n");
            }
            newline[2] = 0;                                    /* maximum 465 dots
per line */
            newline[3]=(endleng-skipleng+1) % 256;  /* set the number of dots */
            newline[4]=(endleng-skipleng+1) / 256;         /* printed per line to
*/
      }                                          /* endleng - skipleng + 1 */
```

```
	else if (endleng-skipleng < 929){		/* double density graphics	*/
		printf ("\n930 dpl graphics\n");
		if (ifscale[0] == 'Y' || ifscale[0] == 'y'){	/* print a scale above
*/
			fprintf (stdprn,"\n	0	83	166	248	331
");
			fprintf (stdprn,"	414	497	579	662
745\n");
		}
		newline[2] = 1;							/*
maximum 930 dots per line */
		newline[3]=(endleng-skipleng+1) % 256;	/* set the number of dots */
		newline[4]=(endleng-skipleng+1) / 256;	/* printed per line to	*/
	}						/* endleng - skipleng + 1 */
	else{							/* quad density
graphics */
		printf ("\n1860 dpl graphics\n");
		if (ifscale[0] == 'Y' || ifscale[0] == 'y'){		/* print a scale
above */
			fprintf (stdprn,"\n	0	166	331	497	662
");
			fprintf (stdprn,"	828	993	1159	1324
1490\n");
	}
		newline[2] = 3;							/*
maximum 1860 dots per line */
		if (endleng - skipleng > 1859)
			endleng = skipleng + 1859;
		newline[3]=(endleng-skipleng+1) % 256;	/* set the number of dots	*/
		newline[4]=(endleng-skipleng+1) / 256;	/* printed per line to the */
	}				/* the lesser of 1872 and endleng-skipleng+1 */
	if (ifscale[0] == 'Y' || ifscale[0] == 'y'){		/* print a scale above
*/
		fputc ( 27, stdprn);
/* the plot, if desired */
		fputc ( 42, stdprn);							/* set
the printer for double	*/
		fputc (  1, stdprn);							/*
density bit-mapped graphics */
		fputc (192, stdprn);							/* 960
dots per line */
		fputc (  3, stdprn);
		for (i=0; i<29; i++)			/*	print a scale */
			fputc (0, stdprn);
		for (i=0; i<18; i++){
			fputc (255, stdprn);
			for (j=0; j<4; j++){
				for (k=0; k<4; k++)
					fputc ( 1, stdprn);
				fputc (15, stdprn);
			}
			for (k=0; k<4; k++)
				fputc ( 1, stdprn);
			fputc (63, stdprn);
			for (j=0; j<4; j++){
				for (k=0; k<4; k++)
					fputc ( 1, stdprn);
```

```
                        fputc (15, stdprn);
                    }
                    for (k=0; k<4; k++)
                        fputc ( 1, stdprn);
                }
                fputc (255, stdprn);
                for (j=0; j<4; j++){
                    for (k=0; k<4; k++)
                        fputc ( 1, stdprn);
                    fputc (15, stdprn);
                }
                for (k=0; k<4; k++)
                    fputc ( 1, stdprn);
                fputc (63, stdprn);
                for (k=0; k<4; k++)
                    fputc ( 1, stdprn);
                fputc (15, stdprn);
        }
        for (k=1; k<=tracnumb; k++){                        /* start reading
traces */
                printf ("Reading trace number %6d     ", k);   /* read in the data */
                putch (13);
                fread (newdata, sizeof(unsigned int), tracleng, fi);
                if (k % skiptrac == 0){
                    for (i=0; i<16384; i++){                        /* erase the
new bottom lines */
                        lines[linenumb[12]][i] = 0;
                        lines[linenumb[13]][i] = 0;
                    }
                    minamp = globemin;
/* reset the minimum and */
                    maxamp = globemax;
/* maximum data values     */
                    zero = 0;
                    for (j=skipleng; j<=endleng; j++){            /* find the maximum
and */
                                                                  /* minimum data
values  */
                        maxamp = (maxamp>newdata[j]) ? maxamp : newdata[j];
                        minamp = (minamp<newdata[j]) ? minamp : newdata[j];
                        zero += newdata[j];
                    }
                    zero /= endleng - skipleng;                   /* find the average
data value */
                    if (maxamp - zero > zero - minamp)
                        minamp = 2 * zero - maxamp;                          /* and
scale the data to */
                    else
                    /* center on that value */
                        maxamp = 2 * zero - minamp;
                                                                  /* scale the data to
fit the desired bounds */
                    scale = (16 + 32 * traceamp) * 0.995 / (1.0 + maxamp - minamp);
                                                                      /* center the
scaled data in the bounds */
                    base = 16 * (3 - traceamp);
                    for (i=skipleng; i<=endleng; i++){
```

```
                                                /* scale the data to
fill the available area */
                        newdata[i]=(unsigned int)((newdata[i]-minamp)*scale+base);

                }
                for (i=skipleng; i<=endleng; i++){
                        if (newdata[i] > 103){                          /*
this is the zeroeth line */
                                lines[linenumb[0]][i] |= twopow[0][(int)(newdata[i] -
104)];

                                lines[linenumb[1]][i] |= twopow[0][8];
                                lines[linenumb[2]][i] |= twopow[0][8];
                                lines[linenumb[3]][i] |= twopow[0][8];
                                lines[linenumb[4]][i] |= twopow[0][8];
                                lines[linenumb[5]][i] |= twopow[0][8];
                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
                        else if (newdata[i] > 95){                      /* this is
the first line */
                                lines[linenumb[1]][i] |= twopow[0][(int)(newdata[i] -
96)];

                                lines[linenumb[2]][i] |= twopow[0][8];
                                lines[linenumb[3]][i] |= twopow[0][8];
                                lines[linenumb[4]][i] |= twopow[0][8];
                                lines[linenumb[5]][i] |= twopow[0][8];
                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
                        else if (newdata[i] > 87){                      /* this is
the second line */
                                lines[linenumb[2]][i] |= twopow[0][(int)(newdata[i] -
88)];

                                lines[linenumb[3]][i] |= twopow[0][8];
                                lines[linenumb[4]][i] |= twopow[0][8];
                                lines[linenumb[5]][i] |= twopow[0][8];
                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
                        else if (newdata[i] > 79){                      /* this is
the third line */
                                lines[linenumb[3]][i] |= twopow[0][(int)(newdata[i] -
80)];

                                lines[linenumb[4]][i] |= twopow[0][8];
                                lines[linenumb[5]][i] |= twopow[0][8];
                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
                        else if (newdata[i] > 71){                      /* this is
the fourth line */
                                lines[linenumb[4]][i] |= twopow[0][(int)(newdata[i] -
72)];

                                lines[linenumb[5]][i] |= twopow[0][8];
                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
                        else if (newdata[i] > 63){                      /* this is
the fifth line */
                                lines[linenumb[5]][i] |= twopow[0][(int)(newdata[i] -
64)];

                                lines[linenumb[6]][i] |= twopow[0][8];
                        }
```

```
                           else if (newdata[i] > 55)                    /* this is
the sixth line */
                                  lines[linenumb[6]][i] |= twopow[0][(int)(newdata[i] -
56)];
                           else if (newdata[i] > 47)                    /* this is
the seventh line */
                                  lines[linenumb[7]][i] |= twopow[1][(int)(newdata[i] -
48)];
                           else if (newdata[i] > 39)                    /* this is
the eighth line */
                                  lines[linenumb[8]][i] |= twopow[1][(int)(newdata[i] -
40)];
                           else if (newdata[i] > 31)                    /* this is
the nineth line */
                                  lines[linenumb[9]][i] |= twopow[1][(int)(newdata[i] -
32)];
                           else if (newdata[i] > 23)                    /* this is
the tenth line */
                                  lines[linenumb[10]][i] |= twopow[1][(int)(newdata[i]
- 24)];
                           else if (newdata[i] > 15)             /* this is the
eleventh line */
                                  lines[linenumb[11]][i] |= twopow[1][(int)(newdata[i]
- 16)];
                           else if (newdata[i] > 7)              /* this is the
twelfth line */
                                  lines[linenumb[12]][i] |= twopow[1][(int)(newdata[i]
- 8)];
                           else
       /* this is the thirteenth line */
                                  lines[linenumb[13]][i] = twopow[1][(int)newdata[i]];
                    }

                    /* plot the next data trace */
                    fputc (13, stdprn);                                       /*
Carriage Return */
                    fputc (10, stdprn);                                       /*
Line Feed */
                    if (k > (3 * skiptrac))                                   /*
print the trace number */
                           fprintf (stdprn, "%3u", (k-3*skiptrac));
                    else
                           fprintf (stdprn, "   ");
                    for (j=0; j<=4; j++)
/* prepare to print the first */
                           fputc (newline[j], stdprn);                 /* half line
of the plot */
                    for (l=skipleng; l<endleng+1; l++)
                           fputc (lines[linenumb[0]][l], stdprn);
                    fputc (13, stdprn);                                       /*
Carriage Return */
                    fputc (10, stdprn);                                       /*
Line Feed */
                    fprintf (stdprn, "   ");
                    for (j=0; j<=4; j++)
/* prepare to print the second */
```

```
                                fputc (newline[j], stdprn);                    /* half line
of the plot */
                        for (l=skipleng; l<endleng+1; l++)
                                fputc (lines[linenumb[1]][l], stdprn);
                        linenumb[14] = linenumb[0];
                        linenumb[15] = linenumb[1];            /* scroll up the line
numbers */
                        for (i=0; i<14; i++)
                                linenumb[i] = linenumb[i+2];
                }
        }
        for (i=0; i<12; i+=2){
                fputc (13, stdprn);                                                  /*
Carriage Return */
                fputc (10, stdprn);                                                  /*
Line Feed */
                if (i < 6 && (2-tracnumb/skiptrac) < i/2 )
                        fprintf (stdprn, "%3u", skiptrac * (((k - 1) / skiptrac) + i/2 -
2));
                else
        /* print the trace number */
                        fprintf (stdprn, "   ");
                for (j=0; j<=4; j++)                        /* prepare to plot a line */
                        fputc (newline[j], stdprn);
                for (l=skipleng; l<endleng+1; l++)      /* plot the first half line */
                        fputc (lines[linenumb[i]][l], stdprn);
                fputc (13, stdprn);                                                  /*
Carriage Return */
                fputc (10, stdprn);                                                  /*
Line Feed */
                fprintf (stdprn, "   ");
                for (j=0; j<=4; j++)                        /* prepare to plot a line */
                        fputc (newline[j], stdprn);
                for (l=skipleng; l<endleng+1; l++)       /* plot the second half line */
                        fputc (lines[linenumb[i+1]][l], stdprn);
        }
        fputc (13, stdprn);                                                          /*
Carriage Return */
        fputc (12, stdprn);                                                          /*
Form Feed */
        fputc (27, stdprn);                                                          /*
reinitialize */
        fputc (64, stdprn);                                              /*    the printer
*/
}
```

```
;* ecko.asm *
;****************************************************
;*                                                  *
;*    ECKO waveform generation function             *
;*       by Garfield Mellema                        *
;*       September 28, 1988                         *
;*                                                  *
;****************************************************
;*                                                  *
;*    This program will read data values from a     *
;*    specified array and send those values to the  *
;*    DAC on the Computerscope card installed in    *
;*    computer.  The program will also initialize,  *
;*    trigger the aforementioned card to begin      *
;*    collecting data as specified by the calling   *
;*    program.  The calling program must call this  *
;*    function with both a far pointer to the first *
;*    integer in the array, and the number of values *
;*    to be sent sequentially from the array to the *
;*    DAC.                                          *
;*                                                  *
;****************************************************

ECKO_TEXT  segment  byte public 'CODE'
   assume  cs:ECKO_TEXT
_ecko      proc     far
   push    bp                          ;save the original location of the stack
   mov     bp, sp                         ;pointer to retrieve the calling data
   push    ds
   push    bx                          ;save the original contents of registers
   push    cx                         ;which may be altered during the function
   push    dx
   mov     ds, word ptr [bp+8]       ;set the data segment to the array segment
   mov     bx, word ptr [bp+6]          ;location of source waveform data array
   mov     cx, word ptr [bp+10]                  ;number of points in the array
   mov     dx, 781                             ;trigger data acquisition system
   out     dx, ax
   mov     dx, 769                                            ;DAC port address
   cli                                                ;disable the interrupts while
wordout:                                            ;sending out the source waveform
   mov     ax, [bx]                             ;get a number (two bytes) from
   out     dx, ax                             ;the array and send it to the DAC
   inc     bx                          ;increment the array index for each byte
   inc     bx
   dec     cx                     ;decrement the number of bytes left to send
   jnz     wordout                     ;continue sending numbers if any remain
   sti                                                ;re-enable the interrupts
   pop     dx
   pop     cx                                         ;return altered registers
   pop     bx                                         ;to their original values
   pop     ds
   pop     bp
   ret
_ecko      endp
   public     _ecko
ECKO_TEXT ends
   end
```

```
/* flotuint.c */
/*************************************************/
/*                                               */
/*    a program to convert a set of binary       */
/*    FLOaTing point data to                     */
/*    Unsigned INTeger data                      */
/*    by Garfield Mellema (November 22, 1988)    */
/*                                               */
/*************************************************/
/*                                               */
/*    This program wil read a file of floating   */
/*    point data in binary format and rewrite    */
/*    it in unsigned integer format.  The        */
/*    source file specified in the command       */
/*    line is assumed to be 125 sets of 1860     */
/*    values.  This program was written for      */
/*    use with the Syntheseis acoustic           */
/*    imaging system.                            */
/*                                               */
/*************************************************/
/*                                               */
/*    max   - minimum value of the source data   */
/*    min   - maximum value of the source data   */
/*    scale - scale factor used to fit the       */
/*            data into the range 0 to 65535     */
/*    zero  - median value of the source data    */
/*                                               */
/*************************************************/

#include <stdio.h>

float dataflot[1860];
unsigned int datauint[1860];

void main (argc, argv)
int argc;
char *argv[];
{

   double max, min, scale;
   int i, j;
   FILE *fib, *fob;

   if (argc < 3){
      printf ("\nThe command line format is:\n\n");
      printf ("flotuint  sourcefilename  destinationfilename\n\n");
      exit(0);
   }
   printf ("Source file name is %s\n", argv[1]);
   printf ("Destination file name is %s\n", argv[2]);
   fib = fopen (argv[1], "rb");
   fob = fopen (argv[2], "wb");
   if (fib == NULL || fob == NULL){
      printf ("*********** file access errror **********");
      exit (0);
   }
   max = -3.4e38;                    /* determine the range of the source data */
   min =  3.4e38;
   for (i=0; i<125; i++){
      fread (dataflot, sizeof(float), 1860, fib);
      for (j=0; j<1860; j++){
         max = (max>dataflot[j]) ? max : dataflot[j];
         min = (min<dataflot[j]) ? min : dataflot[j];
      }
      printf ("Trace number %3d tested", i);
      putch (13);
   }
   rewind (fib);                                /* rewind the source file and */
   scale = 65535 / (max - min);                 /* determine a scaling factor */
   printf ("\nFinished scaling");                 /* to fit the data in range */
   printf ("\nmax is %8g  min is %8g", max, min);
   printf ("\nscale is %8g\n", scale);
   for (i=0; i<125; i++){                              /* write a new data file */
      fread (dataflot, sizeof(float), 1860, fib);
      for (j=0; j<1860; j++){
         datauint[j] = (unsigned int)((dataflot[j] - min) * scale);
      }
      if (fwrite(datauint, sizeof(unsigned int), 1860, fob) != 1860)
      printf ("\n********** binary write error **********");
      printf ("Trace number %3d converted", i);
      putch (13);
   }
   fclose (fib);
   fclose (fob);
}
```

```
/* lasrplot.c */
/*************************************************/
/*
                    */
/*          LASeR printer PLOTting program (.uib)          */
/*          written by Garfield Mellema                                  */
/*          April 1, 1989
*/
/*
                    */
/*************************************************/
/*
                    */
/*          This program will plot Syntheseis data         */
/*          on any Postscript printer.  It asks a          */
/*          number of questions, with respect to the       */
/*          desired plot parameters, of the user and       */
/*          uses the responses to generate the                   */
/*          desired plot.  The user choose to answer       */
/*          the questions interactively, or store          */
/*          the responses, in order, in a file.                  */
/*
                    */
/****************************************************/
/*
                          */
/*          base          - data to plot offset value                    */
/*          clip      - maximum trace overlap cutoff       */
/*          endpoint - the last plotted point in a trace   */
/*          endscale - global scaling end point                        */
/*          *fid      - pointer to the source data file    */
/*          *fps      - pointer to postfile                            */
/*          gain      - maximum calculated trace overlap   */
/*          globemax - maximum source data value                 */
/*          globemin - minimum source data value                 */
/*          globzero - mean source data value                    */
/*          gloscale - use global scaling (Y/N)                        */
/*          inipoint - plotting start point                            */
/*          iniscale - global scaling start point                */
/*          maxamp - maximum trace data value                    */
/*          minamp - minimum trace data value                    */
/*          numbtrac - number of traces to be plotted            */
/*          point         - present data point (x value)         */
/*          shade  - shade technique (pos/neg/all/wig) */
/*          postfile - destination postscript filename     */
/*          scale         - data to plot scale factor                    */
/*          skiptrac - trace skip interval                             */
/*          sourfile - source data file                                      */
/*          title     - the plot title                                       */
/*          tracleng - number of points in each trace            */
/*          xaxtitle - the x - axis title                                    */
/*          xmax          - upper x limit of the plot page       */
/*          xmin          - lower x limit of the plot page       */
/*          xscale - x axis scale factor (times 1/72")     */
/*          xspace - spacing between adjacent x events */
/*          yaxtitle - the y - axis title                                    */
/*          ymax          - upper y limit of the plot page       */
/*          ymin        - lower y limit of the plot page        */
/*          yscale - y axis scale factor (times 1/72")      */
/*          yspace - spacing between adjacent y events      */
/*          zero        - mean value of the present trace */
/*
                            */
/*****************************************************/

#include <stdio.h>

unsigned int sourdata[1860];

void main()
{
        char gloscale[10], shade[10], postfile[80], sourfile[80];
        char title[200], xaxtitle[200], yaxtitle[200];
        FILE *fid, *fps;
        float base, globemax, globemin, globzero, maxamp, minamp;
        float point, scale, xscale, yscale, zero;
        int i, j, k, clip, copies, endpoint, endscale, gain, iniscale, inipoint;
        int numbtrac, skiptrac, tracleng, xmax, xmin, xspace, ymax, ymin, yspace;

        printf ("\n\nLASeR printer PLOTting program for use with the");
        printf ("\nSyntheseis acoustic imaging system");
        printf ("\nwritten by Garfield Mellema (March 31, 1989)");
        printf  ("\n\nThis program will read data from a .uib file and");
        printf ("\nprepare a Postscript file to plot it on a laser printer");
        printf ("\nEnter the name of the source data file.\n");
        scanf ("%s", sourfile);
        printf ("Enter the name of the destination postscript file.\n");
        scanf ("%s", postfile);
        printf ("How many points are in each trace (maximum 1860)?\n");
        scanf ("%d", &tracleng);
        if (tracleng > 1860 || tracleng < 1){
                tracleng = 1860;
                printf ("\nA trace length of 1860 points was assumed\n");
        }
        printf ("The first point read will be number 0\n");
        printf ("Start plotting at point number?\n");
        scanf ("%d", &inipoint);
        if (inipoint > tracleng || inipoint < 0){
                inipoint = 0;
                printf ("\nPlottting will start at point 0\n");
        }
        printf ("Plot up to, and include, point number?\n");
        scanf ("%d", &endpoint);
        if (endpoint > 1859  || endpoint < inipoint){
                endpoint = 1859;
                printf ("Plotting will end at point 1859\n");
        }
        printf ("How many traces are in the file?\n");
        scanf ("%d", &numbtrac);
        printf ("Skip every 'n'th trace.  What is 'n'?\n");
        scanf ("%d", &skiptrac);
        printf ("Would you like positive shading, negative shading,");
        printf ("\nabsolute shading or wiggle traces? (P/N/A/W)\n");
        scanf ("%s", shade);
        printf ("What is the maximum trace overlap (gain)?\n");
        scanf ("%d", &gain);
        printf ("At what overlap should the traces be clipped?\n");
        scanf ("%d", &clip);
        if (clip > gain || clip < 1)
              clip = gain;
        printf ("Enter the plot title\n");
    gets (title);
        gets (title);
        printf ("Enter the x - axis legend\n");
        gets (xaxtitle);
        printf ("Enter the y - axis legend\n");
        gets (yaxtitle);
        printf ("How many copies of this plot would you like?\n");
        scanf ("%d", &copies);
        printf ("Each trace will be individually scaled.\n");
        printf ("Would you prefer global scaling? (Y/N)\n");
        scanf ("%s", gloscale);
        globemax = 0;                                   /* set the maximum and minimum
data values */
        globemin = 65535;                                           /* just outside the
allowable range */
        fid = fopen (sourfile, "rb");
        fps = fopen (postfile, "w");
        if (fid == NULL || fps == NULL){
              printf ("\n**********   file opening error   **********");
              exit (0);
        }                                   /* if global scaling is selected, */
                                                                        /* globally
scale the source data set */
        if (gloscale[0] == 'y' || gloscale[0] == 'Y'){
              printf ("Begin scaling at, and include, point number\n");
              scanf ("%d", &iniscale);
              printf ("End scaling at, and include, point number\n");
        scanf ("%d", &endscale);
       printf ("Begin scaling the source data\n");
              k = 0;
              globzero = 0;
              for (i=1; i<=numbtrac; i++){
                    fread (sourdata, sizeof(unsigned int), tracleng, fid);
                    if (i % skiptrac == 0){
                          printf ("Scaling trace number %d", i);
                          putch (13);
             k ++;
                          for (j=inipoint; j<=endpoint; j++){
                                globemax =(globemax>sourdata[j]) ? globemax :
sourdata[j];
                                globemin =(globemin<sourdata[j]) ? globemin :
sourdata[j];
                                globzero += sourdata[j];
                          }
                    }
              }
              globzero /= k * (float)(endpoint-inipoint+1);
              rewind (fid);
              printf ("\nglobemin is %f", globemin);
              printf ("\nglobemax is %f", globemax);
```

```
            printf ("\nglobzero is %f\n", globzero);
        }                                               /* boundaries of the plotting
area in 1/72" scale */
        xmin = 90;
        xmax = 738;
        ymin = 108;
        ymax = 522;
                                                                  /* rotate the page
coordinates so the x and y */

/*      axis coincide with the plot axis */
        fprintf (fps, "\n612 0 translate 90 rotate");
        fprintf (fps, "\n1 setlinewidth");

                                /* draw the page outline */
        fprintf (fps, "\nnewpath 90 108 moveto 90 522 lineto 738 522 lineto");
        fprintf (fps, "\n738 108 lineto closepath stroke");


        /* add a text centering procedure */
        fprintf (fps, "\n/centertext {dup stringwidth ");
        fprintf (fps, "pop -2 div 0 rmoveto show} def");

                                                                       /* add the
plot title and the axis legends */
        fprintf (fps, "\n/Times-Roman findfont 16 scalefont setfont");
        fprintf (fps, "\n396 54 moveto (%s) centertext", title);
        fprintf (fps, "\n/Times-Roman findfont 12 scalefont setfont");
        fprintf (fps, "\n414 75 moveto (%s) centertext", xaxtitle);
        fprintf (fps, "\n66 315 moveto 90 rotate ");
        fprintf (fps, "(%s) centertext -90 rotate", yaxtitle);

                        /* move the origin to the lower left corner of the plotting area
*/
        fprintf (fps, "\n%d %d translate", xmin, ymin);
        xmax -= xmin;
        ymax -= ymin;
                                                    /* change the page scaling to best fit
the plot size */
        xscale = 0.01 * xmax / (numbtrac + 2 * clip);
        yscale = ymax / (1.0 * tracleng);

                                                                  /* calculate the new
plotting area boundaries */
        xmax /= xscale;
        ymax /= yscale;
        fprintf (fps, "\n%6f %6f scale", xscale, yscale);


/* add number centering procedures */
        fprintf (fps, "\n/Times-Roman findfont 10 scalefont setfont");
        fprintf (fps, "\n/xcenternumber {gsave %6f %6f ", 1/xscale, 1/yscale);
        fprintf (fps, "scale dup stringwidth pop -2 div 0 rmoveto ");
        fprintf (fps, "show grestore} def");
        fprintf (fps, "\n/ycenternumber {gsave %6f %6f ", 1/yscale, 1/xscale);
        fprintf (fps, "scale dup stringwidth pop -2 div 0 rmoveto ");
        fprintf (fps, "show grestore} def");
```

```
                                    /* add the hashmarks and position numbers on
the x axis */
     for (i = 100 * clip + 50; i < 100 * (numbtrac + clip); i += 500){
          fprintf (fps, "\n%d 0 moveto 0 %d rlineto stroke", i, (int)(-5/yscale));
          fprintf (fps, "\n%d %d moveto ", i, (int)(-15/yscale));
          fprintf (fps, "(%d) xcenternumber", (i - 100 * clip - 50) / 50);
     }
                                    /* add the hashmarks and position numbers on
the y axis */
     for (i = 1860; i > 0; i -= 29){
          if (((1889-i) % 116) != 0){
               fprintf (fps, "\n0 %d moveto %d 0 ", i, (int)(-3/xscale));
               fprintf (fps, "rlineto stroke");
          } else {
               fprintf (fps, "\n0 %d moveto %d 0 rlineto ", i, (int)(-6/xscale));
               fprintf (fps, "stroke \n%d %d moveto ", (int)(-8/xscale), i);
               fprintf (fps, "90 rotate (%d) ", (int)(10*(1889-i)/116));
               fprintf (fps, "ycenternumber -90 rotate");
        }
     }
   minamp = globemin;                             /* use the global scaling limits
*/
     maxamp = globemax;
     zero = globzero;
     if ((maxamp - zero) > (zero - minamp))
          minamp = 2 * zero - maxamp;
     else
          maxamp = 2 * zero - minamp;

/* add the positive shaping routine */
     fprintf (fps, "\n/posshade {dup 3 index gt {pop pop pop} ");
     fprintf (fps, "{1 index lineto lineto} ifelse} def ");

/* add the negative shaping routine */
     fprintf (fps, "\n/negshade {dup 3 index lt {pop pop pop} ");
     fprintf (fps, "{1 index lineto lineto} ifelse} def ");

                                    /* plot the data */
     for (i=1; i<=numbtrac; i++){
    printf ("Reading trace number %d", i);
          putch (13);
          fread (sourdata, sizeof(unsigned int), tracleng, fid);
          if (i % skiptrac == 0){
                                                  /* or determine the local
scaling limits */
               if (gloscale[0] != 'Y' && gloscale[0] != 'y'){
                    zero = 0;
                    for (j=inipoint; j<=endpoint; j++){
                         maxamp = (maxamp > sourdata[j]) ? maxamp :
sourdata[j];
                         minamp = (minamp < sourdata[j]) ? minamp :
sourdata[j];
                         zero += sourdata[j];
                    }
                    zero /= (endpoint - inipoint + 1);
                    if ((maxamp - zero) > (zero - minamp))
```

```
                                minamp = 2 * zero - maxamp;
                        else
                                maxamp = 2 * zero - minamp;
                }                                                       /* calculate
the data to plot scale factor */
        scale = gain * 100.0 / (maxamp - minamp);
        base = 100 * (i + clip) - 50;
                for (j=inipoint; j<=endpoint; j++){

/* scale each data point and ensure */

/* it is within the clipping range */
                        point = (sourdata[j] - zero) * scale;
                        point = (point < (clip * 100)) ? point : (clip * 100);
                point = (point > (-clip * 100)) ? point : (-clip * 100);
                        point += base;

/* put it on the postscript stack */
                        if ((j-inipoint) % 400 != 0 || j == inipoint){
                                if ((j-inipoint) % 10 == 0)
                                        fprintf (fps, "\n%ld", (long)point);
                else
                                        fprintf (fps, " %ld", (long)point);
                                }
                        else{
        /* plot the data on the stack */
                                fprintf (fps, "\n%ld %d moveto", (long)point, j-
inipoint);
                                fprintf (fps, "\n%d -1 %d ", j-inipoint-1, j-
inipoint-400);

                                /* positive shading */
                                if (shade[0] == 'P' || shade[0] == 'p'){
                                        fprintf (fps, "{2 copy lineto %ld ",
(long)base);
                                        fprintf (fps, "posshade} for stroke");

                                /* negative shading */
                                } else if (shade[0] == 'N' || shade[0] == 'n'){
                                        fprintf (fps, "{2 copy lineto %ld ",
(long)base);
                                        fprintf (fps, "negshade} for stroke");

                                /* absolute shading */
                                } else if (shade[0] == 'A' || shade[0] == 'a'){
                                        fprintf (fps, "{2 copy lineto %ld ",
(long)base);
                                        fprintf (fps, "1 index lineto lineto} for
stroke");
                                } else
                                /* no shading */
                                        fprintf (fps, "{lineto} for stroke");
                                fprintf (fps, "\n%ld", (long)point);
                        }
                }                                                               /*
clean up anything left on the stack */
                fprintf (fps, " %d moveto", j-inipoint-1);
                fprintf (fps, "\n%d -1 ", j-inipoint-2);
                fprintf (fps, "%d ", (j-inipoint - ((j-inipoint) % 400)));
            if (shade[0] == 'P' || shade[0] == 'p'){
                    fprintf (fps, "{2 copy lineto %ld posshade} ", (long)base);
                    fprintf (fps, "for stroke");
                } else if (shade[0] == 'N' || shade[0] == 'n'){
                    fprintf (fps, "{2 copy lineto %ld negshade} ", (long)base);
                    fprintf (fps, "for stroke");
            } else if (shade[0] == 'A' || shade[0] == 'a'){
                    fprintf (fps, "{2 copy lineto %ld 1 index ", (long)base);
                    fprintf (fps, "lineto lineto} for stroke");
                } else
                    fprintf (fps, "{lineto} for stroke");
        }
        }
/* wrap up the plot and the program */
        if (copies > 1)
            fprintf (fps, "\n/#copies %d def", copies);
        fprintf (fps, "\nshowpage\n");
        fclose (fid);
        fclose (fps);
}
```

```
/* median3.c */
/***************************************************************************/
/*                                                                         */
/*    a program to 3-point space MEDIAN filter a set of binary unsigned    */
/*    integer data                                                         */
/*                                                                         */
/*    by Garfield Mellema (January 11, 1989)                               */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*       This program will examine three traces of data at a time from     */
/*    the first file specified in the command line and determine the       */
/*    median value, at each point, of the data.  This value is then        */
/*    subtracted from the center trace in an attempt to remove             */
/*    horizontally recurring artifacts  from the image stored in the       */
/*    file.  The values subtracted from the first and last traces of       */
/*    the image will be the same as those subtracted from the second       */
/*    and second last respectively.  The results of this process will      */
/*    be stored in the second file specified in the command line.  This    */
/*    program assumes that the file being filtered is a standard size      */
/*    Syntheseis image file, that is 125 traces of 1860 points.  They      */
/*    are read one at a time and are rotated in the array datauint by      */
/*    successive shifts of the array indices stored in the array isline.   */
/*                                                                         */
/***************************************************************************/
/*                                                                         */
/*    datauint[][] - unsigned integer raw data                             */
/*    filtered[]   - filtered version of the present trace                 */
/*    isline[]     - datauint array presently acting as the present trace  */
/*    median[]     - median of present and immediately adjacent traces     */
/*    temp         - median filtered value of the present point            */
/*    value[]      - present and immediately space adjacent points         */
/*    zero         - the average value of the image being filtered         */
/*                                                                         */
/***************************************************************************/

#include <stdio.h>

unsigned int datauint[3][1860], filtered[1860], median[1860];

void main (argc, argv)
int argc;
char *argv[];
{
   unsigned int value[3], extra;
   int i, j, k, l, isline[4];
   float zero, temp;
   FILE *fib, *fob;

   if (argc < 3){
        printf ("\nThe command line format is:\n\n");
      printf ("median3  sourcefilename.uib  destinationfilename.uib\n\n");
      exit(0);
   }
   printf ("Source file name is %s\n", argv[1]);
   printf ("Destination file name is %s\n", argv[2]);
   fib = fopen (argv[1], "rb");
```

```
fob = fopen (argv[2], "wb");
if (fib == NULL || fob == NULL){
     printf ("*********** file access errror **********");
   exit (0);
}
for (i=0; i<125; i++){
   fread (datauint, sizeof(unsigned int), 1860, fib);
   for (j=0; j<1860; j++)
      zero += (float)datauint[0][j];
   printf ("Trace number %3d considered", i);
   putch (13);
}
zero /= 232500;
printf ("\nzero is %f\n", zero);
rewind (fib);
for (i=0; i<1860; i++)
   median[i] = zero;
for (i=0; i<3; i++)
   isline[i] = i;
fread (&datauint[0][0], sizeof(unsigned int), 1860, fib);
fread (&datauint[1][0], sizeof(unsigned int), 1860, fib);
for (i=0; i<125; i++){                                  /* for each trace */
   if (i==0){
      fread (&datauint[isline[2]][0], sizeof(unsigned int), 1860, fib);
      for (j=0; j<1860; j++){            /* for each point of each trace */
                                            /* load similar point values */
      value[0] = datauint[isline[0]][j];
      value[1] = datauint[isline[1]][j];
      value[2] = datauint[isline[2]][j];
      if (value [1] > value [0]){
         extra = value [1];
         value [1] = value [0];
         value [0] = extra;
      }
      if (value [2] > value [1]){
         extra = value [2];
         value [2] = value [1];
         value [1] = extra;
      }
      if (value [1] > value [0]){
         extra = value [1];
         value [1] = value [0];
         value [0] = extra;
      }
      median[j] = value[1];
      temp =(float)datauint[isline[0]][j] -(float)value[1] +(float)zero;
      if (temp < 0){
         printf ("\nfiltered value is too small %6f ", temp);
         printf ("at %d %d", i, j);
         temp = 0;
      } else if (temp > 65535){
         printf ("\nfiltered value is too large %f ", temp);
         printf ("at %d %d", i, j);
         temp = 65535;
      }
      filtered[j] = (unsigned int) temp;
   }
```

```
}
if (i==1){
   for (j=0; j<1860; j++){              /* for each point of each trace */
                                           /* load similar point values */
      temp =(float)datauint[isline[1]][j] -(float)median[j] +(float)zero;
      if (temp < 0){
         printf ("\nfiltered value is too small %f ", temp);
         printf ("at %d %d", i, j);
                                    temp = 0;
      } else if (temp > 65535){
         printf ("\nfiltered value is too large %f ", temp);
         printf ("at %d %d", i, j);
         temp = 65535;
      }
      filtered[j] = (unsigned int) temp;
   }
} else if (i==123){
   fread (&datauint[isline[2]][0], sizeof(unsigned int), 1860, fib);
   for (j=0; j<1860; j++){              /* for each point of each trace */
      value[0] = datauint[isline[0]][j];
      value[1] = datauint[isline[1]][j];
      value[2] = datauint[isline[2]][j];
      if (value [1] > value [0]){
         extra = value [1];
         value [1] = value [0];
         value [0] = extra;
      }
      if (value [2] > value [1]){
         extra = value [2];
         value [2] = value [1];
         value [1] = extra;
      }
      if (value [1] > value [0]){
         extra = value [1];
         value [1] = value [0];
         value [0] = extra;
      }
      median[j] = value[1];
      temp =(float)datauint[isline[1]][j] -(float)value[1] +(float)zero;
      if (temp < 0){
         printf ("\nfiltered value is too small %f ", temp);
         printf ("at %d %d", i, j);
         temp = 0;
      } else if (temp > 65535){
         printf ("\nfiltered value is too large %f ", temp);
         printf ("at %d %d", i, j);
         temp = 65535;
      }
      filtered[j] = (unsigned int) temp;
   }
} else if (i==124){
   for (j=0; j<1860; j++){              /* for each point of each trace */
                                           /* load similar point values */
      temp =(float)datauint[isline[2]][j] -(float)median[j] +(float)zero;
      if (temp < 0){
         printf ("\nfiltered value is too small %f ", temp);
         printf ("at %d %d", i, j);
```

```
            temp = 0;
         } else if (temp > 65535){
            printf ("\nfiltered value is too large %f ", temp);
            printf ("at %d %d", i, j);
            temp = 65535;
         }
         filtered[j] = (unsigned int) temp;
      }
   } else{
      fread (&datauint[isline[2]][0], sizeof(unsigned int), 1860, fib);
      for (j=0; j<1860; j++){               /* for each point of each trace */
         value[0] = datauint[isline[0]][j];
         value[1] = datauint[isline[1]][j];
         value[2] = datauint[isline[2]][j];
           if (value [1] > value [0]){
            extra = value [1];
            value [1] = value [0];
            value [0] = extra;
         }
         if (value [2] > value [1]){
            extra = value [2];
            value [2] = value [1];
            value [1] = extra;
         }
         if (value [1] > value [0]){
            extra = value [1];
            value [1] = value [0];
            value [0] = extra;
         }
         temp =(float)datauint[isline[1]][j] -(float)value[1] +(float)zero;
         if (temp < 0){
            printf ("\nfiltered value is too small %f ", temp);
            printf ("at %d %d", i, j);
            temp = 0;
         } else if (temp > 65535){
            printf ("\nfiltered value is too large %f ", temp);
            printf ("at %d %d", i, j);
                                                   temp = 65535;
         }
         filtered[j] = (unsigned int) temp;
      }
   }
   if (i != 0 && i != 123){
      isline [3] = isline [0];
      isline [0] = isline [1];
      isline [1] = isline [2];
      isline [2] = isline [3];
   }
   if (fwrite(filtered, sizeof(unsigned int), 1860, fob) != 1860)
      printf ("\n********** binary write error **********");
      printf ("Trace number %3d filtered", i);
      putch (13);
   }
   fclose (fib);
   fclose (fob);
}
```

```
/* multrace.c */
/*****************************************************/
/*                                                   */
/*    MULtiple TRACE data acquisition program        */
/*    by Garfield Mellema                            */
/*    February 16, 1989                              */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    This program will read data values from a      */
/*    specified file and send those values to the    */
/*    DAC on the Computerscope card installed in     */
/*    computer.  The program will also initialize,   */
/*    trigger, and collect data from the multiplexed */
/*    ADC on the aforementioned card.  The sending   */
/*    out of the file data and the collection of the */
/*    resultant data is repeated 16 times and summed */
/*    each time for the reduction of noise in the    */
/*    received data.  The 16384 points of summed     */
/*    received data are then crosscorrelated in the  */
/*    frequency domain with a previously made        */
/*    recording of the source using a Fast Fourier   */
/*    Transform taken from the book Numerical Recipes */
/*    in C by Pres et. al.  The resultant waveform is */
/*    windowed and stored in the file specified in   */
/*    the program command line.                      */
/*       This program makes use of the files         */
/*    inwave.bin and refwave.uib which store the     */
/*    source and correlation reference waveforms     */
/*    respectively.                                  */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    SAMPRATE  - ADC sampling rate in microseconds  */
/*    ARASIZE   - collected dataset size             */
/*    refwave[] - correlation reference waveform     */
/*    datasum[] - the stacked received data          */
/*    finwave[] - datasum[] centered around zero     */
/*    autox[]   - the crosscorrelated data           */
/*    xetra[]   - an extra crosscorrelation array    */
/*    dataout[] - the source signal as DAC values    */
/*    datain    - pointer to the ADC data buffer     */
/*    offset    - the average value of datasum[]     */
/*    start     - time structures to                 */
/*    stop      - control trolley movement           */
/*                                                   */
/*****************************************************/

#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <time.h>

#define SAMPRATE 5                     /* data sampling rate in microseconds */
#define ARASIZE 16384                   /* ARASIZE is the input dataset size */
                                      /* 16384 is the maximum allowable size */
```

```
float refwave[ARASIZE], finwave[ARASIZE], autox[2*ARASIZE], xetra[2*ARASIZE];
extern void ecko(int *, int);
unsigned int datasum[ARASIZE];
int dataout[23000];

void main (argc, argv)
int argc;
char *argv[];
{
   unsigned char *datain;
   unsigned int i, j, k, l;
   double offset;
   void correl(), four1();
   time_t start, stop;
   FILE *fia, *fib, *fob;

                       /* set a pointer to the start of the datacard memory */

   datain = (unsigned char *)0xA0000000;

                    /* check that the destination filename has been supplied */

   if (argc < 2){
      printf ("\nThe command line format is:\n\n");
      printf ("multrace  destinationfilename\n\n");
      exit(0);
   }
   printf ("\nThe destination filename is %s", argv[1]);

                  /* open the necessary files and read in the source wavelet */

   fia = fopen("\\synseis\\inwave.bin", "rb");
   fib = fopen("\\synseis\\refwave.flb", "rb");
   fob = fopen(argv[1], "wb");
   if (fia == NULL || fib == NULL || fob == NULL){
        printf ("\n*********** File opening error **********");
        exit(0);
   }

   printf("\nreading source wavelet from file\n");
   fread (dataout, sizeof(int), 23000, fia);
   printf ("\nFinished taking in source waveform");

      /* set up the data acquisition card to collect the returning waveform */

   outp (779, 0);                                 /* disable data acquisition */
   for (i=0; i<16; i++)                           /* take data from channel 8 */
      outp (784, 7);                           /* channel 8 is the microphone */
   outp (785, 0);                           /* use the timer on the data card */
   outp (776, 0);                      /* trigger data sampling from software */
   outp (775, 116);                      /* use mode 2 of the internal timer */
   outp (773, (SAMPRATE % 256));                     /* sample every (LSB) */
   outp (773, (SAMPRATE / 256));             /* SAMPRATE microseconds (MSB) */
   outp (775, 50);                                 /* take 1 sample at a time */
   outp (772, 1);
   outp (772, 0);
outp (775, 178);                                /* set the post-triggering */
outp (774, 0);                                     /* delay to 16384 words */
outp (774, 64);                              /* i.e. take 16384 data points */

/* read the crosscorelation reference waveform from refwave.uib, convert */
/* it to floating point, and subtract its mean to center it around zero  */

fread (datasum, sizeof(unsigned int), ARASIZE, fib);
offset = 0.0;
for (i=0; i<ARASIZE; i++)                  /* find the mean of the waveform */
   offset += (float)datasum[i];
offset /= ARASIZE;

for (i=0; i<ARASIZE; i++)                  /* subtract it from the waveform */
   refwave[i] = datasum[i] - offset;

                      /* the following is the major data collection loop */
                      /* it cycles once per trace (trolley position)      */

for (k=0; k<125; k++){
   for (j=0; j<ARASIZE; j++)               /* erase the received data array */
        datasum[j] = 0;
     printf ("\nReady to take data position %u", k);

      /* the following loop causes the source waveform to be transmitted */
   /* and collects the returning waveform from the data acquisition card */

   for (j=0; j<16; j++){
      outp (789, 0);                         /* manually select data bank B */
                                           /* i.e. store data points in buffer A */
      printf ("\nAbout to call ecko");
      outp (795, 06);                  /* set sample buffer A to 16384 words */
      ecko (dataout, 23000);                 /* send out the source waveform */
                                           /* and initiate data acquisition */
      printf ("\nReturned from ecko");
      for (i=0; i<SAMPRATE*2; i++)
         printf ("\nfinish taking data block %3d position %3d", j, k);

         outp (788, 0);              /* read the acquired data from bank A */
         printf ("\nsumming data block");
         for (i=0; i<ARASIZE; i++)
                                                /* retrieve the acquired data */
                                  /* note that it is stored in reversed byte order */
            datasum[i] += datain[2*i] + 256 * datain[2*i+1];
        }

      /* this section moves the trolley 20 mm to the next trace position */
         /* the trolley moves ,using a synchronous motor, at 2.5 sec/mm */

   for (i=0; i<25; i++)                            /* use digita 1 switch #1 */
        outp (794, 64);                           /* to move the trolley WEST */
   time (&start);
   printf ("\nMoving trolley..");
   do
      time (&stop);                                    /* 0.416 mm / second */
   while (difftime (stop, start) < 48.0);     /* 20 / 0.416 = 48 seconds */
   for (i=0; i<25; i++)
```

```
          outp (794, 00);                                            /* stop
trolley */
      printf ("\nStopping trolley....");

                  /* the mean of the stacked received data is determined and */
            /* subtracted to yield a floating point array centered around zero */

      offset = 0.0;
      for (i=0; i<ARASIZE; i++)
         offset += datasum[i];                    /* find the average value of */
      offset /= ARASIZE;                          /* the stacked received data */
      printf ("\nThe offset is %f", offset);
      for (i=0; i<ARASIZE; i++)
         finwave[i] = datasum[i] - offset;         /* attempt to zero the data */

                        /* the trolley is given time to come to a complete */
                        /* stop during the following crosscorrelation      */

         correl (refwave-1, finwave-1, (long)ARASIZE, autox-1, xetra-1);
                              /* write the useful crosscorrelated data to a file */

                   /* the relevant portions of the crosscorrelated data are */
               /* written to the file specified in the program command line */

      printf ("\nwriting crosscorrelated data to file");
      if (fwrite(autox+ARASIZE/2+6800, sizeof(float), 1392, fob) != 1392)
         printf ("\n********** binary write error **********");
      if (fwrite(autox, sizeof(float), 468, fob) != 468)
         printf ("\n********** binary write error **********");
   }
   fclose (fia);
   fclose (fib);
   fclose (fob);
}

/**************************************************************************/
/*                                                                        */
/*   The following program sections were taken from Numerical Recipes in C */
/*   in C.  No attempt is made to document them as the aforementioned     */
/*   book does a superb job of this rather complex task.                  */
/*                                                                        */
/**************************************************************************/

void correl(data1,data2,n,ans,fft)
float data1[],data2[],ans[],fft[];
long n;
{
       long no2,i;
       float dum;
       void twofft(),realft();

       printf ("\nDo a forward FFT");
       twofft(data1,data2,fft,ans,n);
       no2=n/2;
       for (i=2;i<=n+2;i+=2) {
             ans[i-1]=(fft[i-1]*(dum=ans[i-1])+fft[i]*ans[i])/no2;
             ans[i]=(fft[i]*dum-fft[i-1]*ans[i])/no2;
```

```
        }
        ans[2]=ans[n+1];
        printf ("\nDo an inverse FFT");
        realft(ans,no2,-1);
}

void twofft(data1,data2,fft1,fft2,n)
float data1[],data2[],fft1[],fft2[];
long n;
{
        long nn3,nn2,jj,j;
        float rep,rem,aip,aim;
        void four1();

        nn3=1+(nn2=2+n+n);
        for (j=1,jj=2;j<=n;j++,jj+=2) {
                fft1[jj-1]=data1[j];
                fft1[jj]=data2[j];
        }
        four1(fft1,n,1);
        fft2[1]=fft1[2];
        fft1[2]=fft2[2]=0.0;
        for (j=3;j<=n+1;j+=2) {
                rep=0.5*(fft1[j]+fft1[nn2-j]);
                rem=0.5*(fft1[j]-fft1[nn2-j]);
                aip=0.5*(fft1[j+1]+fft1[nn3-j]);
                aim=0.5*(fft1[j+1]-fft1[nn3-j]);
                fft1[j]=rep;
                fft1[j+1]=aim;
                fft1[nn2-j]=rep;
                fft1[nn3-j] = -aim;
                fft2[j]=aip;
                fft2[j+1] = -rem;
                fft2[nn2-j]=aip;
                fft2[nn3-j]=rem;
        }
}

void realft(data,n,isign)
float data[];
int isign;
long n;
{
        long i,i1,i2,i3,i4,n2p3;
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;
        void four1();

        theta=3.141592653589793/(double) n;
        if (isign == 1) {
                c2 = -0.5;
                four1(data,n,1);
        } else {
                c2=0.5;
                theta = -theta;
        }
        wtemp=sin(0.5*theta);
```

```
	wpr = -2.0*wtemp*wtemp;
	wpi=sin(theta);
	wr=1.0+wpr;
	wi=wpi;
	n2p3=2*n+3;
	for (i=2;i<=n/2;i++) {
		i4=1+(i3=n2p3-(i2=1+(i1=i+i-1)));
		h1r=c1*(data[i1]+data[i3]);
		h1i=c1*(data[i2]-data[i4]);
		h2r = -c2*(data[i2]+data[i4]);
		h2i=c2*(data[i1]-data[i3]);
		data[i1]=h1r+wr*h2r-wi*h2i;
		data[i2]=h1i+wr*h2i+wi*h2r;
		data[i3]=h1r-wr*h2r+wi*h2i;
		data[i4] = -h1i+wr*h2i+wi*h2r;
		wr=(wtemp=wr)*wpr-wi*wpi+wr;
		wi=wi*wpr+wtemp*wpi+wi;
	}
	if (isign == 1) {
		data[1] = (h1r=data[1])+data[2];
		data[2] = h1r-data[2];
	} else {
		data[1]=c1*((h1r=data[1])+data[2]);
		data[2]=c1*(h1r-data[2]);
		four1(data,n,-1);
	}
}

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void four1(data,nn,isign)
float data[];
int isign;
long nn;
{
	long n,mmax,m,j,istep,i;
	double wtemp,wr,wpr,wpi,wi,theta;
	float tempr,tempi;

	n=nn << 1;
	j=1;
	for (i=1;i<n;i+=2) {
		if (j > i) {
			SWAP(data[j],data[i]);
			SWAP(data[j+1],data[i+1]);
		}
		m=n >> 1;
		while (m >= 2 && j > m) {
			j -= m;
			m >>= 1;
		}
		j += m;
	}
	mmax=2;
	while (n > mmax) {
		istep=2*mmax;
		theta=6.28318530717959/(isign*mmax);
```

```
            wtemp=sin(0.5*theta);
            wpr = -2.0*wtemp*wtemp;
            wpi=sin(theta);
            wr=1.0;
            wi=0.0;
            printf (" %d",mmax);
            for (m=1;m<mmax;m+=2) {
                    for (i=m;i<=n;i+=istep) {
                            j=i+mmax;
                            tempr=wr*data[j]-wi*data[j+1];
                            tempi=wr*data[j+1]+wi*data[j];
                            data[j]=data[i]-tempr;
                            data[j+1]=data[i+1]-tempi;
                            data[i] += tempr;
                            data[i+1] += tempi;
                    }
                    wr=(wtemp=wr)*wpr-wi*wpi+wr;
                    wi=wi*wpr+wtemp*wpi+wi;
            }
            mmax=istep;
        }
}

#undef SWAP
```

```
/* sourtran.c */
/*****************************************************/
/*                                                   */
/*    SOURCE data acquisition program                */
/*    by Garfield Mellema                            */
/*    October 21, 1988                               */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    This program will read data values from a      */
/*    specified file and send those values to the    */
/*    DAC on the Computerscope card installed in     */
/*    computer.  The program will also initialize,   */
/*    trigger, and collect data from the multiplexed */
/*    ADC on the aforementioned card.  The sending   */
/*    out of the file data and the collection of the */
/*    resultant data is repeated 16 times and summed */
/*    each time for the reduction of noise in the    */
/*    received data.  The summed received data is    */
/*    then stored in a specified file.  A speaker    */
/*    is connected to the DAC and a microphone is    */
/*    connected to the ADC, both via amplifiers.     */
/*    The intent of this program is to aid in finding */
/*    the transform function of the speaker,         */
/*    microphone, and amplifiers setup.              */
/*    The original data file is read once.           */
/*    The function 'ecko' is written in assembly     */
/*    language as the file 'ecko.asm'.               */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    datain    - a pointer to the beginning of      */
/*                the data acquisition card memory   */
/*    samprate  - the sampling rate to be used by    */
/*                the data acquisition card          */
/*    datasum[] - stacked data sampled by the        */
/*                data acquisition card              */
/*    dataout[] - calculated source signal to be     */
/*                sent the data acquisition via Ecko */
/*                                                   */
/*****************************************************/


#include <stdio.h>
#include <math.h>
#include <dos.h>
#define MOD 0             /* use an offset of 128 for the modified datacard */
                          /* use an offset of 0 for the unmodified datacard */

extern void ecko(int *, int);
unsigned int datasum[16384];
int dataout[23000];

void main()
{
   unsigned char *datain;
   long samprate;
   unsigned int i, j, k;
   FILE *fi, *fib, *fo, *fob;


   datain = (unsigned char *)0xA0000000;
                             /* set a pointer to the start of the datacard memory */
                             /* use 0xD0000000 for the unmodified datacard */
                                      /* use 0xA0000000 for the modified datacard
*/
   samprate = 5;                        /* sampling rate in microseconds */
   fi = fopen("\\data\\inwave.prn", "w");
   fib = fopen("\\data\\inwave.bin", "rb");
   fo = fopen("\\data\\outwave.prn", "w");
   fob = fopen("\\data\\outwave.bin", "wb");
   if (fib == NULL || fo == NULL || fob == NULL){
      printf ("\n*********** File opening error **********");
        exit(0);
   }
   printf("\nreading wavelet from file\n");
   fread (dataout, sizeof(int), 23000, fib);
   printf ("\nFinished taking in source waveform");
   outp (779 + MOD, 0);                         /* disable data acquisition */
   for (i=0; i<16; i++)                         /* take data from channel 8 */
      outp (784 + MOD, 7);                   /* channel 8 is the microphone */
   outp (785 + MOD, 0);                   /* use the timer on the data card */
   outp (776 + MOD, 0);            /* trigger data sampling from software */
   outp (775 + MOD, 116);             /* use mode 2 of the internal timer */
   outp (773 + MOD, (samprate % 256));                /* sample every (LSB) */
   outp (773 + MOD, (samprate / 256));     /* samprate microseconds (MSB) */
   outp (775 + MOD, 50);                         /* take 1 sample at a time */
   outp (772 + MOD, 1);
   outp (772 + MOD, 0);
   outp (775 + MOD, 178);                        /* set the post-triggering */
   outp (774 + MOD, 0);                            /* delay to 16384 words */
   outp (774 + MOD, 64);                     /* i.e. take 16384 data points */
   for (j=0; j<8192; j++)
        datasum[j] = 0;
   printf ("\nReady to take data blocks");
   for (j=0; j<16; j++){
      outp (789 + MOD, 0);                   /* manually select data bank B */
                                       /* i.e. store data points in buffer A */
        printf ("\nAbout to call ecko");
      outp (795 + MOD, 06);          /* set sample buffer A to 16384 words */
        ecko (dataout, 23000);               /* send out the source waveform */
                                           /* and initiate data acquisition */
      printf ("\nReturned from ecko");
      for (i=0; i<2*samprate; i++)
         printf ("\nfinish taking data block %3d", j);
      outp (788 + MOD, 0);           /* read the acquired data from bank A */
      printf ("\nsumming data block");
      for (k=0; k<16384; k++)                /* retrieve the acquired data */
                           /* note that it is stored in reversed byte order */
         datasum[k] += datain[2*k] + 256 * datain[2*k+1];
   }
                        /* write the collected data to a file in the format */
                                /* time(usec)           microphone point */
   printf ("\nwriting source data to a file");
   for (i=0; i<23000; i+=4)
      fprintf (fi,"\n%10lu %6u", (i*samprate), dataout[i]);
   printf ("\nwriting summed data blocks to file");
   for (i=0; i<16384; i+=2)
   fprintf (fo,"\n%10lu %6u", (i*samprate), datasum[i]);
                                        /* write the data in binary for further processing */
   if (fwrite(datasum, sizeof(unsigned int), 16384, fob) != 16384)
        printf ("\n********** binary write error **********");
   fclose (fi);
   fclose (fib);
   fclose (fo);
   fclose (fob);
}
```

```
/* uintasci.c */
/**************************************************/
/*                                                */
/*    a program to convert a file of binary       */
/*    Unsigned INTeger  data to ASCIi format      */
/*    by Garfield Mellema (December 1, 1988)      */
/*                                                */
/**************************************************/
/*                                                */
/*    This program wil read a file of unsigned    */
/*    integer data in binary format and rewrite   */
/*    it in signed ASCII format.  The source      */
/*    file specified in the command line is       */
/*    assumed to be 125 sets of 1860 values.      */
/*    This program was written for use with       */
/*    the Syntheseis acoustic imaging system.     */
/*                                                */
/**************************************************/
/*                                                */
/*    max   - minimum value of the source data    */
/*    min   - maximum value of the source data    */
/*    zero  - median value of the source data     */
/*                                                */
/**************************************************/

#include <stdio.h>

unsigned int datauint[1860];

void main (argc, argv)
int argc;
char *argv[];
{

   unsigned int max, min;
   int i, j, k;
   float zero;
   FILE *fib, *fo;

   if (argc < 3){
      printf ("\nThe command line format is:\n\n");
      printf ("intasci  sourcefilename  destinationfilename\n\n");
      exit(0);
   }
   printf ("Source file name is %s\n", argv[1]);
   printf ("Destination file name is %s\n", argv[2]);
   fib = fopen (argv[1], "rb");
   fo = fopen (argv[2], "w");
   if (fib == NULL || fo == NULL){
      printf ("*********** file access errror **********");
      exit (0);
   }
   zero = 0;                                /* determine the parameters to */
   max = 0;                                   /* be used to scale the data */
   min = 65535;
   for (i=0; i<125; i++){
      fread (datauint, sizeof(unsigned int), 1860, fib);
      for (j=0; j<1860; j++){
         max = (max>datauint[j]) ? max : datauint[j];
         min = (min<datauint[j]) ? min : datauint[j];
         zero += datauint[j];
      }
      printf ("Trace number %3d tested", i);
      putch (13);
   }                                          /* rewind the source file and */
   rewind (fib);                                /* calculate the zero point */
   zero /= 232500;                                   /* 125 * 1860 = 232500 */
   printf ("\nzero is %8g\n", zero);
   for (i=0; i<125; i++){
      fread (datauint, sizeof(unsigned int), 1860, fib);
      for (j=1859; j>=0; j-=5){
         fprintf (fo, "\n");                          /* write the new file */
         for (k=j; k>j-5; k--)
         fprintf (fo, "%7.0f  ", (datauint[k]-zero));
      }
      printf ("Trace number %3d converted", i);
      putch (13);
   }
}
```

```
/* wavegen.c */
/*****************************************************/
/*                                                   */
/*    swept frequency WAVEform GENeration program    */
/*    by Garfield Mellema                            */
/*    November 1, 1988                               */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    This program will compute the values to be     */
/*    sent to the DAC on the RC Computerscope data   */
/*    acquisition to produce a swept frequency       */
/*    sinewave.  This waveform is normally sent to   */
/*    the speaker on the trolley to generate         */
/*    reflection and diffraction data.  Some source  */
/*    parameters are specified as constants, other   */
/*    are embedded in the program.  The instantaeous */
/*    frequency of the signal is calculated along    */
/*    with the shaping factor to be used at each     */
/*    point and they are combined to produce one     */
/*    of the stream of values of the source signal.  */
/*                                                   */
/*****************************************************/
/*                                                   */
/*    amplitud  - the amplitude factor of the        */
/*                present source signal point        */
/*    frequency - the instanteous frequency of the   */
/*                present source signal point        */
/*    angle     - the phase angle of the present     */
/*                source signal point                */
/*    starttap  - the point number at which to end   */
/*                the starting taper                 */
/*    endtaper  - the point number at which to begin */
/*                the ending taper                   */
/*    spare     - a temporary variable               */
/*                                                   */
/*****************************************************/

#define PI 3.1415926536
#define ARASIZE 23000              /* the size of the waveform file required */
                                        /* SOURTRAN.C requires 25000 points */
#define Fstart 1.0                         /* the start frequency (in KHz) of */
                                              /* the swept frequency waveform */
#define Fstaper 1.2                      /* the end frequency (in KHz) of the */
                                        /* taper at the start of the waveform */
#define Fetaper 14.8                       /* the start frequency (in KHz) of */
                                      /* the taper at the end of the waveform */
#define Fend 15.0                            /* the end frequency (in KHz) of */
                                              /* the swept frequency waveform */
#define VOLUME 1024                         /* the amplitude of the generated */

                                             /* waveform from 0 to 2 * VOLUME */
#define SAMPRATE 333                /* the number of samples per millisecond */
                                                   /* of the generated waveform */
                               /* the ECKO.ASM routine used by SOURTRAN.C */
                                          /* requires 333 points per millisecond */
#include <stdio.h>
```

```
#include <math.h>
#include <dos.h>

int point[ARASIZE];

void main()
{

   long i;
   double amplitud, frequency, angle, starttap, endtaper, spare;
   FILE *fob;

   printf ("\nswept frequency WAVEform GENeration program");
   printf ("\nWritten by Garfield Mellema (November 1, 1988)");
   fob = fopen ("d:\\data\\inwave.bin", "wb");
   printf("\n\n\n%4.2f to %4.2f KHz swept frequency sinewave\n", Fstart, Fend);
   starttap = ARASIZE * (Fstaper - Fstart) / (Fend - Fstart);
   endtaper = ARASIZE * (1 - (Fend - Fetaper) / (Fend - Fstart));
   printf ("\nstarttap is %6.0f, endtaper is %6.0f\n", starttap, endtaper);
   angle = 0.0;
   for (i=0; i<ARASIZE; i++){
                                             /* shape the waveform envelope */

/*      amplitud = 1.00 - 1.0 * fabs ((i * 1.0 / ARASIZE) - 0.75);       */
      spare = 1.0 - fabs ((i * 1.0 / ARASIZE) - 0.7);
      amplitud = spare * spare;
                      /* apply a cosine taper to the start of the waveform */
        if (i < starttap)
         amplitud *= 0.5 * (1 - cos (PI * i / starttap));
                        /* apply a cosine taper to the end of the waveform */
      else if (i>endtaper)
      amplitud *= 0.5*(1- cos (PI * (ARASIZE - i) / (ARASIZE - endtaper)));
                                          /* instantaneous frequency in kHz */
      frequency = Fstart + (Fend-Fstart) * (i * 1.0 / ARASIZE);
      if (frequency > 5.5 && frequency < 9.5)
           amplitud *= 1.0 + 0.1 * (9.5 - frequency);
      angle += frequency * PI * 2.0 / SAMPRATE;
      point[i] = (amplitud * sin (angle) + 1) * VOLUME;
   }
   printf ("\nFiling points\n");
         /* write the waveform data in binary for the waveform output routine */
   if (fwrite(point, 2, ARASIZE, fob) != ARASIZE)
   printf ("\n********** binary write error **********");

                 /* if the generated file is not at least as big as the maximum */
                    /* SOURTRAN.C resultant file, pad it with zeroes until it is */

   if (ARASIZE < 23000){
        for (i=0; i<ARASIZE; i++)
         point[i] = VOLUME;
      for (i=ARASIZE; i<23000; i+=ARASIZE)
         if (fwrite(point, 2, ARASIZE, fob) != ARASIZE)
              printf ("\n********** binary write error **********");
   }
}
```