234-8
IPOS

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 19/00

A1

(11) International Publication Number: WO 00/02155

(43) International Publication Date: 13 January 2000 (13.01.00)

(21) International Application Number: PCT/US99/15188

(22) International Filing Date: 6 July 1999 (06.07.99)

(30) Priority Data:
09/110,575 6 July 1998 (06.07.98) US

(71) Applicant: MOBIL OIL CORPORATION [US/US]; 3225 Gallows Road, Fairfax, VA 22037 (US).

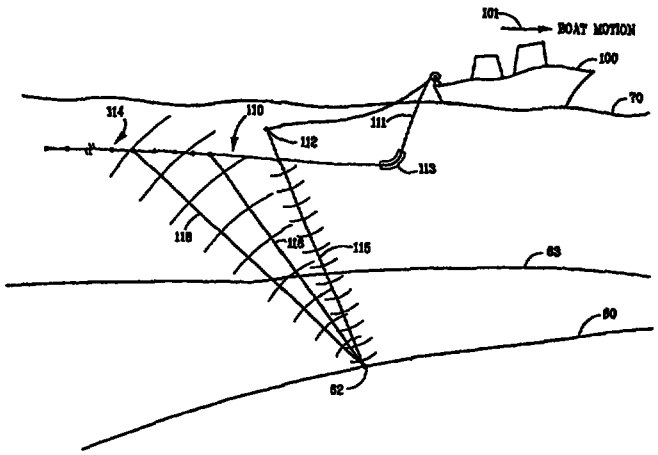
(72) Inventors: HOUCK, Richard, Thomas; 2518 Fallview, Carrollton, TX 75007 (US). JENKERSON, Michael, Richard; 1102 N. Graves, McKinney, TX 75069 (US). SEYB, Stefan, Michael; 10947 Cromwell Drive, Dallas, TX 75229 (US).

(74) Agents: FURR, Robert, B., Jr. et al.; Mobil Oil Corporation, 3225 Gallows Road, Fairfax, VA 22037 (US).

(81) Designated States: AU, BR, CA, ID, KZ, NO, SG, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: METHOD FOR REMOVING SEISMIC NOISE CAUSED BY EXTERNAL ACTIVITY



(57) Abstract

A method for analyzing a target seismic trace (100), generally comprises selecting other traces in source receiver space (114), not including traces from the same shot record that contains the target trace (122), and comparing the selected traces (118), to the target trace (115). The method generally includes setting a distance value indicative of the number of shot gathers (114), to be considered in conjunction with analyzing a target trace, specifying which traces from a plurality of shot records to compare to the target trace based on the distance value, calculating a first amplitude value associated with the amplitude of the target trace (100), calculating a second amplitude valve associated with the amplitudes of the specified comparison traces (100), and comparing the first amplitude value to the second amplitude value. The comparison and target traces can be analyzed (100), in various windows of time. The target trace is modified in a suitable manner if it is found to contain noise (100). The preferred method can be implemented in a seismic processing system (100), that is included onboard a seismic ship (100). Because the method does not require all of the seismic data to be acquired and then sorted before analyzed for noise, as is the case for many conventional trace editing techniques, the method of the preferred embodiment can be performed in near real or real time. As such, real time data quality control is possible.

METHOD FOR REMOVING SEISMIC NOISE CAUSED BY EXTERNAL ACTIVITY

The present invention generally relates to a method and apparatus for reducing the effect of unwanted noise in seismic data caused by nearby seismic or other noise-producing activity. More particularly, the present invention relates to reducing noise in marine seismic data caused by seismic activity from nearby seismic ships. Still more particularly, the present invention relates to eliminating noise from nearby seismic ships during real or near real-time data acquisition or in subsequent processing.

The field of seismology focuses on the use of artificially generated elastic waves to locate mineral deposits such as hydrocarbons, ores, water, and geothermal reservoirs. Seismology also is used for archaeological purposes and to obtain geological information for engineering. Exploration seismology provides data that, when used in conjunction with other available geophysical, borehole, and geological data, can provide information about the structure and distribution of rock types and contents.

Most oil companies rely on seismic interpretation to select sites for drilling exploratory oil wells. Despite the fact that the seismic data is used to map geological structures rather than find petroleum directly, the gathering of seismic data has become a vital part of selecting the site of exploratory and development wells. Experience has shown that the use of seismic data greatly improves the likelihood of a successful venture.

Seismic data acquisition is routinely performed both on land and at sea. At sea, a seismic ship deploys a streamer or cable behind the ship as the ship moves forward. The streamer includes multiple receivers in a configuration generally as shown in Figure 1. Streamer 110 trails behind ship 100, which moves generally in the direction of the arrow 101. The streamer includes a plurality of receivers 114. As shown, a source 112 is also towed behind ship 100. Source 112 and receivers 114 typically deploy below the surface of the ocean 70. Streamer 110 typically includes electrical or fiber-optic cabling for interconnecting receivers 114 and seismic equipment on ship 100. Streamers are usually constructed in sections 25 to 100 meters in length and include groups of up to 35 or more uniformly spaced receivers. The streamers may be several miles long and often a seismic ship trails multiple

streamers to increase the amount of seismic data collected. Seismic data acquisition involving a single streamer including a single source and multiple receivers is referred to as two-dimensional ("2D") seismic data acquisition. The use of multiple streamers and/or multiple sources is referred to as three-dimensional ("3D") seismic data acquisition. Data is digitized via electronic modules located near the receivers 114 and then the data is transmitted to the ship 100 through the cabling at rates of 7 (or more) million bits of data per second. Processing equipment aboard the ship controls operation of the trailing source and receivers and performs initial processing on the acquired data. The large volume of data acquired usually requires further processing in land-based computing centers after the seismic survey has completed.

Seismic techniques estimate the distance between the ocean surface 70 and subsurface structures, such a structure 60 which lies below the ocean floor 63. By estimating the distance to a subsurface structure, the geometry or topography of the structure can be determined. Certain topographical features are indicative of oil and/or gas reservoirs. To determine the distance to subsurface structure 60, source 112 emits seismic waves 115, which reflect off subsurface structure 60. The reflected waves are sensed by receivers 114. By determining the length of time that the seismic waves 115 took to travel from source 112 to subsurface structure 60, and to receivers 114, an estimate of the distance to subsurface structure 60 can be obtained.

The receivers used in marine seismology are commonly referred to as hydrophones, or marine pressure phones, and are usually constructed using a piezoelectric transducer. Synthetic piezoelectric materials, such as barium zirconate, barium titanate, or lead mataniobate, are generally used. A sheet of piezoelectric material develops a voltage difference between opposite faces when subjected to mechanical bending. Thin electroplating on these surfaces allows an electrical connection to be made to the device so that this voltage can be measured. The voltage is proportional to the amount of mechanical bending or pressure change experienced by the receiver resulting from seismic energy propagating through the water. Various configuration of hydrophones are used such as disk hydrophones and cylindrical hydrophones.

Two types of seismic sources are used to generate seismic waves for the seismic measurements. The first source type comprises an impulsive source which generates a high-

energy, short time duration impulse. The time between emitting the impulse from the source and detecting the reflected impulse by a receiver is used to determine the distance to the subsurface structure under investigation. A second type of source generates lower magnitude, vibratory energy. The measurement technique that uses such sources is referred to as the marine vibratory seismic ("MVS") technique. Rather than imparting a high magnitude pressure pulse into the ocean in a very short time period, vibratory sources emit lower amplitude pressure waves over a time typically between 5 and 8 seconds, although longer time periods are also possible. Further, the frequency of the vibrating source varies from 5 to 150 Hz, although the specific low and high frequencies differ from system to system. The frequency of the source may vary linearly or non-linearly with respect to time. The frequency variations are commonly called a "frequency sweep." The frequency sweep may thus be between 5 and 150 Hz and 5 to 7 seconds in duration. The magnitude of the seismic wave oscillations may vary or remain at a constant amplitude, but generally are much lower than the magnitude of impulsive sources.

The amount of data collected in a typical seismic survey can be voluminous. For example, a typical seismic survey may involve the mapping of a 1000 square mile region of the ocean by a 3D seismic ship trailing six or eight streamers. Each streamer may have 400 or 500 receivers attached to it. For each seismic measurement, 6-8 seconds of data (referred to as a "trace") is acquired and stored on magnetic tape on-board the ship. To completely map the survey area, which may require several weeks, one billion traces, or more, may be acquired and stored on tape. The traces are stored as "shot records" on the tape with a shot record representing the traces from all of the receivers from a single shot pulse from a source. This volume of data necessitates the use of thousands of magnetic tapes which are manually loaded into storage bins in the ship initially and then automatically accessed by specialized equipment on-board the ship during the survey. Because of the enormous volume of data acquired during a typical survey, improved techniques for efficiently processing the data are needed.

Some areas of the world are heavily explored so that several seismic ships, working for related or unrelated operators, may be conducting seismic surveys at the same time and in relatively close proximity to one another (within 50 miles or so). The receivers typically used in seismic streamer cables are highly sensitive as well as omnidirectional (sensitive to signals travelling from any direction). Virtually any sound that passes through the location of the

receiver is detected by the receiver. Accordingly, the receivers respond not only to an impulse or "shock pulse" generated by their own ship, but may also respond to shots generated by another ship in the vicinity.

Figure 2, for example, illustrates three seismic ships 20, 30, and 40 in the same general area of the ocean. As shown, seismic signals 31 and 41 generated by ships 30 and 40, respectively, propagate through the water to the receivers 114 on ship 20 as well as reflect off subsurface structures. To ships 30 and 40 their seismic signals 31, 41 represent desirable signals, but to ship 20 those signals represent undesirable noise. Additionally, there are other types of noise external to the seismic acquisition system of ship 20 that affect marine data acquisition. Examples of such noise include weather noise and "cultural" noise from rigs and shipping. All of these noises affect the cost and quality of seismic data by necessitating a relaxation in the specification for data quality control, by requiring that the data be re-acquired (re-shot), or requiring "time sharing" during which closely positioned seismic ships take turns acquiring seismic data to minimize noise on the seismic signals detected by each ship's seismic system. It thus is highly desirable to remove, or at least minimize, the noise present in a seismic signal that is generated from such external sources.

Several algorithms have been suggested for noise reduction. Some of these methods involve a process called "stacking" in which multiple traces are added together or otherwise combined into a single trace. In robust stacking methods high amplitude samples are discarded in the stacking process itself. Trace weighting methods are similar in approach to robust stacking, except that the traces are inversely weighted prior to stacking rather than selectively eliminated during stacking.

Adaptive-predictive methods, such as *f-x* deconvolution, reduce random noise by predicting coherent events. Since interference is random in common offset or CDP domains, random zones of high amplitude noise can be replaced on a per-sample basis with the surrounding acceptable data. Moveout filtering specifically targets coherent events. This method includes the well-known *f-k* filter and τ-*p* filter which depend on the signal and interference having different (and separable) "dips" in the common shot domain.

Trace editing techniques attempt to separate high amplitude zones from the surrounding signal. These zones then are either weighted down, blanked or replaced with neighboring data using an interpolation scheme. The present invention relates to an

improvement in trace editing methodology.

Conventional trace editing methods generally require (1) all of the seismic data to be collected, (2) sorting the traces into an appropriate "domain," and then (3) removing the noise from the sorted traces. Conventional shot domain methods, while not requiring sorting, do not respond well to inherent variations in signal levels. In a 2D acquisition system, sorting the traces necessarily involves considerable time and processing power because of the volume of data. This processing usually occurs at a processing facility after the seismic survey has been completed. The processing burden is exacerbated in a 3D acquisition system which may involve sorting a billion traces or more. One of the most costly aspects of seismic data-processing involves playing back the traces from the magnetic tapes and storing the processed information back on tape. The processing of tens of thousands of magnetic tapes requires specialized equipment operating over a relatively long period of time and is extremely costly.

Because full seismic data processing is generally considered too expensive a task to be performed on the ship at sea, the seismic operators of the ship have little assurance regarding the quality of the data that they have collected. Quality control "stacks" which are generated on board typically are insufficient to properly evaluate the data degradation. It is not until after the survey has completed and the tapes are processed that the operators will know whether the data is good or not. Often times only select portions of the data may be infected with noise or otherwise corrupted. Reshooting bad sections of a seismic line generally is not economically feasible once the ship has left the survey area. It would be highly advantageous to be able to evaluate the data at sea in real-time, or near real-time, to be able to determine the quality of the acquired data while the data is being acquired. Thus, if it is determined that a particular part of a line needs to be re-shot, those records can be re-shot while the ship is still in the general area. Such real-time data-processing capability would make it economically feasible to re-shoot records when required, thereby increasing the quality of the data.

Accordingly, an improved method of seismic data processing is needed to solve the problems noted above. Such a method preferably could be performed in near-real or real-time while the data is being acquired. Further, the processing method would preferably lower the cost required to process the data. Despite the advantages such a seismic processing system would offer, to date no such system is known to exist.

The deficiencies noted above with the prior art have been solved by a method for

analyzing a target seismic trace generally comprising selecting other traces in source-receiver space not including traces from the same shot record that contains the target trace, and comparing those traces to the target trace. The preferred method generally includes setting a distance value indicative of the number of shot gathers to be considered in conjunction with analyzing a target trace, specifying which traces from a plurality of shot records to compare to the target trace based on the distance value, calculating a first amplitude value associated with the amplitude of the target trace, calculating a second amplitude value associated with the amplitudes of the specified comparison traces, and comparing the first amplitude value to the second amplitude value.

The steps of calculating the first and second amplitudes preferably includes calculating the root mean square (RMS) averages of time windowed portions of the target and comparison traces. Further, the median is calculated for all of the RMS averages for the comparison traces for each time window. If the RMS average for the target trace for a given time window is greater than the median of the RMS averages for the comparison traces for the same time window, that portion of the target trace is considered noisy and either removed, set to zero or another predefined value, or otherwise altered to reduce the detrimental effect of the noise.

The preferred method can be implemented in seismic processing system that can be included aboard a seismic ship. Because the method does not require all of the seismic data to be acquired and then sorted, as is the case with many conventional trace editing techniques, the method of the preferred embodiment can be performed in near real or real-time. As such, data quality control is greatly enhanced.

The various characteristics described above, as well as other features and benefits, will be readily apparent to those skilled in the art upon reading the following detailed description of the preferred embodiments of the invention, and by referring to the accompanying drawings.

Figure 1 shows a ship for making seismic measurements with a towed streamer array including a seismic source and multiple receivers;

Figure 2 shows multiple seismic ships in a survey area illustrating the potential for signals originating from one ship to be detected undesirably by receivers on a nearby ship;

Figure 3 is a stacking chart showing the method of the preferred embodiment for

selecting traces to compare against a target trace;

Figure 4 shows an exemplary set of seismic traces showing how each trace can be divided into two or more time "windows;"

Figure 5 is a block diagram of the seismic processing system constructed in accordance with the preferred embodiment; and

Figure 6 shows a seismic ship configured for 3D seismic data acquisition.

The method of the preferred embodiment reduces or eliminates noise in seismic data generated from a source external to the seismic system. This method may be employed in a variety of seismic applications such as towed streamer and on-bottom cable (OBC) configurations. Generally, the source of this noise is seismic activity from other seismic ships in the same general vicinity. The method described below is also useful for eliminating or reducing noise with the following properties: high amplitude (higher than the signal), time limited (does not contaminate the whole record), and contaminates different parts of different records.

The preferred embodiment of the invention can best be understood by referring to Figure 3 which shows a "stacking chart." In the stacking chart, each circle 122 represents a trace from a receiver. Portions from four exemplary traces 122 are shown in Figure 4 and are now explained in conjunction with Figure 1. Each trace represents the seismic pressure signals detected by a receiver 114 after a source 112 has fired off a shot pulse. Each trace 122 typically represents approximately 8 seconds of data as the seismic signal propagates down through the water, reflects off the ocean floor and various structural interfaces below the ocean floor, and propagates back up to the receivers 114. The various "bumps" or peaks 123 in each trace represent increases in pressure sensed by the receiver caused by reflected signals from the various subsurface interfaces.

Referring to Figure 3, each row of traces in the stacking chart represent traces taken from a single "shot record." As noted above, a shot record (also called a "shot gather") is the set of traces acquired from all of the receivers 114 in a streamer after a source 112 has fired off a shot impulse. Each row of traces thus represents a shot record initiated at a different point in time than the other shot records in the chart as the ship 100 moves forward in the direction shown at a speed of approximately 4.5 knots. As shown, the stacking chart indicates the location of traces in "source-receiver" space. Moving along the X_s axis indicates a change

in source location and consequently a change in shot record. Moving along the in-line receiver axis, X_r, indicates a change in receiver location for a given source location or shot record. Each circle in the chart represents a seismic trace, such as that illustrated in Figure 4.

The streamers may be several miles long and may include hundreds of receivers in each streamer. Because the ship moves relatively slowly and each shot record lasts approximately 8 seconds, many of the trace recordings are taken from receivers located at the same spot in the ocean in different shot records. To aid in viewing the traces, stacking charts, such as that shown in Figure 3, include the various shot records spaced out along the vertical axis (in-line source location) even though all of the shot records were acquired along the single line of travel of the ship. The distance measured in units of receiver locations and marked with reference numeral 128 in Figure 3 represents the number of receiver locations the ship traveled from one shot record to the subsequent shot record.

The stacking chart of Figure 3 represents traces from a 2D survey (*i.e.*, one source and multiple in-line receivers). A 2D stacking chart has been shown for clarity—a 3D stacking chart is difficult to represent graphically. The principles of the present invention, however, are also applicable to 3D seismic data.

In accordance with the preferred embodiment, each trace in the survey is analyzed for the presence of noise. The trace to be analyzed is referred to as the "target" trace. Once a determination is made that noise is present in part or all of a trace, that trace, or at least the "noisy" portion, is modified to reduce the detrimental effect of the noise. In general, the preferred method identifies and removes or minimizes the effect of high amplitude noise in a shot record by comparing each trace in the survey with nearby traces in surrounding shot records. No sorting of traces is required in this method prior to noise editing in contrast to many conventional noise editing techniques. As such, noise editing can be performed in real or near-real time as the data is acquired using the techniques described herein.

The first step in the preferred method is to select the traces for comparing to the target trace to be analyzed for the presence of noise. The selected traces are referred to as "comparison" traces. Referring still to Figure 3, an exemplary target trace is shown as trace 126 identified by the square surrounding the circular trace representation. The comparison traces that have been selected for comparing to the target trace 126 are shown as traces 124 represented as blackened circles. In accordance with one embodiment of the invention, the

comparison traces do not include traces in the same shot record as the target trace. Omitting traces from the same shot record as the target trace is preferred because those traces will likely also be infected with the noise if the target trace is so infected. Accordingly, the comparison traces preferably do not include the traces along the same row of traces as target trace 126. The comparison traces, however, do include traces from shot records taken relatively close in time to the shot record that includes the target trace. Further, the comparison traces within the nearby shot records preferably only include traces taken at receiver locations near the target trace receiver location. As shown in Figure 3, the comparison traces are taken from two shot records before and after the target record (i.e., the shot record containing the target trace 126). Further, within the selected records, traces within four receiver locations in the $+X_r$ and $-X_r$ directions are selected for comparison to target trace 126. Preferably, the number of shot records to be considered in the analysis before and after the shot record including the target trace, as well as the number of traces selected from those shot records, is variable. The number of traces selected in Figure 3 represents only one possible set of traces to select.

One way in which to implement this selection step is to use equation (1) below:

$$DIST = \sqrt{(Xst - Xsc)^2 + (Xrt - Xrc)^2} \qquad (1)$$

where **DIST** is the distance (in units of receiver and source locations), **Xst** is the coordinate of the source location for the target trace 126; **Xsc** is the coordinate of the source location for a comparison trace; **Xrt** is the coordinate of the receiver location for the target trace 126; and **Xrc** is the coordinate of the receiver location for a comparison trace.

Equation (1) generally can be used by setting the variables **DIST**, **Xst**, and **Xrt** to desired values. The remaining two variables, **Xsc** and **Xrc**, then define a circle in source-receiver space, such as circle 132 shown in Figure 3 for a **DIST** value of 2. As such, any traces with coordinates **Xsc** and **Xrc** that fall within the circle 132 defined by equation (1) are used to compare to the target trace. This set of comparison traces preferably omits traces from the same shot record as the target trace, as noted above.

Equation (1) preferably is used by computing the distance, **DIST**, between a target trace with source location, **Xst**, and receiver coordinate, **Xrt**, and a comparison trace with source location, **Xsc**, and receiver location **Xrc**. The comparison trace is selected if **DIST** is less than a user-specified search distance. Because this method of selecting comparison traces

works directly on shot records without first requiring sorting (although the records could be sorted if desired), it can be run at sea in real or near real-time during data acquisition.

In accordance with the preferred embodiment, the coordinates of the source and receivers are determined using well-known navigational equipment. The coordinates of the source and receivers preferably are recorded with each shot record. If, however, navigation-based coordinates are unavailable, the operator of the system can specify nominal geometry parameters for the shot records.

A software embodiment using 2D equation (1) is included in the Microfiche Appendix (MULTIEDIT™) attached to the end of this disclosure. The MULTIEDIT™ software, as well as the companion 3D MEGAEDIT™ software also included in the appendix, preferably runs in the ProMAX seismic processing environment (Advance Geophysical), but the software can be easily adapted to other processing systems now known or later developed. This system runs under any of numerous versions of the Unix operating system, typically on "mid-range" work stations, but also on larger "main frame" machines. For example, the software can be run on an IBM RS6000 running the AIX 4.1.5 operating system (IBM). The code can also run on any of other varieties of the Unix operating systems that are supported by Advance Geophysical, such as Solaris (Sun Microsystem) and IRIX (Silicon Graphics).

Referring to Figure 5, seismic processing system 50 constructed in accordance with the preferred embodiment generally includes a data processing unit 51, a display 52, a tape drive 54, memory 56, and a fixed disk storage device 57. A plotter 58 for generating hard copies of the processed data is optional, but if included preferably comprises a large format 12, 24, or 36 inch plotter such as GS-636 manufactured by Oyo, or any other suitable printing or plotting device.

The data processing unit 51 may include a 72 MHZ, IBM RS6000 7013-590 CPU, but faster and higher performance CPU's are preferred. The display can be any suitable display such as the 6091 19 manufactured by IBM. The memory 56 includes any suitable type of memory such as dynamic random access memory (DRAM). Memory 56 preferably is implemented with at least 256 MB, but 500 MB, or more, is preferred. Less than 256 MB is also acceptable although with diminished performance. The fixed disk 57 preferably has a capacity of 20-30 GB and may comprise a conventional hard drive, CD ROM drive, or other suitable types of mass storage units. The tape drive 54 preferably comprises a 8mm high

density tape drive such as the 850S manufactured by Exabyte.

Referring again to Figure 3, other traces falling outside the circle 132 defined by equation (1) can also be used to compare to target trace 124. For example, as shown in Figure 3 traces 134 may also be included if desired. Moreover, the comparison traces determined by equation (1) are only a preferred set of traces as other traces can be included and some of the traces that would otherwise be selected using equation (1) can be omitted from the analysis as desired. Further, the noise arrival times may vary slowly between shots thereby contaminating more than one adjacent shot record with the noise at a similar trace time. To handle such cases, the MULTIEDIT™ (and MEGAEDIT™) software preferably includes an option to exclude shots from the trace selection. It is contemplated that this feature will be used whenever the noise arrival time changes by less than the length of the noise wave train from one shot to the next.

Referring now to Figure 6, a 3D seismic ship 200 is shown trailing two sources 212 and four streamers of receivers 214 so as to acquire 3D seismic data. Two axes X and Y are shown in Figure 6. The **X** axis corresponds to the in-line **X** axis of Figure 3. The horizontal **Y** axis is the "cross-line" axis and is used when referring to multiple streamers of receivers and to multiple sources. As such, each receiver 214 in Figure 6 is located by a coordinate (**Xr,Yr**) and each source 212 is located by coordinate (**Xs,Ys**).

In a 3D seismic survey, equation (1) can be modified to provide for the possibility of multiple sources and multiple streamers of receivers. For example, equation (2) below can be used to determine a set of comparison traces for comparing with the target trace in a 3D seismic survey:

$$DIST = \sqrt{(Xst - Xsc)^2 + (Yst - Ysc)^2 + (Xrt - Xrc)^2 + (Yrt - Yrc)^2} \qquad (2)$$

where **DIST**, **Xst**, **Xsc**, **Xrt**, and **Xrc** refer to the same quantities as in equation (1). Further, **Yst** is the cross-line coordinate of the source location for a target trace; **Ysc** is the cross-line coordinate of a source location pertaining to a comparison trace; **Yrt** is the cross-line coordinate of the receiver location for a target trace; and **Yrc** is the cross-line coordinate of

the receiver location for a comparison trace.

Equation (2) generally defines a hypersphere in four dimensional source-receiver space that includes traces generated by the same or different sources 212. As with the 2D case, the traces selected in the 3D case for comparison to a target trace preferably do not include traces from the same shot record which produced the target trace. This selection methodology permits traces from other streamers to be compared with the target trace. An exemplary software implementation using equation (2) is provided in the Microfiche Appendix (MEGAEDIT™).

The second step of the preferred method involves editing the target trace. In accordance with the preferred embodiment of the invention, and referring again to Figure 4, noise editing of a target trace is based on a suitable amplitude measure, such as the root mean square (RMS) amplitude, computed in time windows 130 for each trace. Other types of averages, such as average absolute amplitude, can also be used if desired. The RMS average of the target trace in each time window is compared to the median of the RMS averages of the comparison traces from the comparable time windows. Generally, at least three comparison traces are needed to compute the median value. That is, the RMS average is computed for each time window in the comparison traces and then the median is computed of the RMS averages for all of the comparison traces in each time window. If the RMS average of the target trace in a given time window exceeds the comparison traces' median value by more than a user-specified threshold factor, that window in the target trace is classified as noise and flagged for removal or other suitable processing. The threshold factor may be time-variant and user-adjustable. Further, the windows 130 in which the RMS and median values are calculated can be any desirable size. In accordance with the preferred embodiment, these windows preferably are in the range of 100 to 200 milliseconds, although other window sizes also are possible. Additionally, the windows need not be back-to-back as shown in Figure 4. Instead, each window may overlap an adjacent window. For example, a typical window length may be 100-200 milliseconds with a corresponding amplitude window increment of 50-100 milliseconds (overlapping).

Noise zones flagged for removal may be muted, replaced with a constant value, replaced with data interpolated from surrounding traces, or modified in any suitable manner so as to reduce the detrimental effect of the noise. Additionally, a suitable linear taper function

can be applied at the start and end of the noise zone so as to minimize the amount of high frequency harmonics added to the data, as would be understood by one skilled in the art.

The number of traces inside the hypersphere defined by equation (2) can be quite large (several hundred) even for a short search distance of two or three shot intervals. Computing the median of so many values may be time-consuming and, in many cases, unnecessary. Thus, the exemplary 3D software embodiment MEGAEDIT™ preferably includes an option for decreased runtime by computing medians from a subset of the full set of comparison traces. As the software listing indicates, a user-specified "maximum number of traces" value can be selected uniformly from the full set of comparison traces, so that the subset covers approximately the same region of source-receiver space as the full set. The median value computed from a few tens of traces is not likely to differ significantly from the median computed from a few hundred traces, as long as the comparison traces cover substantially the same part of the source-receiver space. However, the time required to sort a few tens of amplitudes is much smaller than the time required to sort several hundred amplitudes because most sorting algorithms have run times proportional to a quantity between N and N^2 depending on the number and initial order of the values, as would be understood by one of ordinary skill.

The MULTIEDIT™ or MEGAEDIT™ programs can be run at sea during data acquisition or on land after the survey has completed. Alternatively, part of the processing can be performed on board the ship during acquisition and part can be performed post data acquisition (on land). For example, the steps to determine which traces have been infected with noise can be performed during the data acquisition phase. Then, after the survey is over, the noisy traces can be edited in a data processing system on land.

Additionally, the computed edits can be mapped as attributes in a conventional binning system in which the trace locations can be analyzed. Common depth point gathers determined to have excessive noise using the techniques described above can be evaluated in the binning system. Upon viewing the system, a decision can be made to reshoot an area with a large congregation of edits.

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and

Trademark Office patent files or records, but otherwise reserves all copyrights.

The preferred embodiments of the invention described above provide significant advantages over conventional marine seismic acquisition and processing systems. For example, no pre-noise analysis sorting is required. This permits the noise editing of the preferred embodiments to occur in real-time, or at least in near real-time, relative to the rate at which the data is acquired. As such, the noise editing of the preferred embodiments can be performed on-board the ship while the data is being acquired. The ability to analyze the data for noise while the data is being acquired greatly enhances the quality control of the data. If, for example, portions of a seismic line are fatally defective due to the presence of high amplitude noise, the seismic operator can make that determination while the ship is still in the same general vicinity of the ocean where the noisy shot record was acquired. Accordingly, parts of a line can be reshot before the ship leaves the area if desired. Conventional seismic acquisition systems do not permit such near real-time data analysis and thus re-shooting shot records generally becomes prohibitively expensive. The quality of the data is greatly enhanced over conventional seismic systems.

```
TITLE:                  multiedt2.f

TIME PRINTED:           Thu May 07 08:04:23 1998

TIME QUEUED:            Thu May 07 08:04:22 1998

PRINTED AT:             ipo (hplj-3)  ~ d1r007

SUBMITTED BY:           mrjenker@d1r007

DELIVER TO:             mrjenker@d1r007


FLAG VALUES:
a=0, =a, f=, h=, j=S, 1=66, p=10, s=Courier, z=0, A=1, B=gn,  C=!,
H=d1r007, J=+, L=+, N=1, P=lp0:lp0, X=IBM-850, 1=!, 2=!, 3=!, 4=!,
5=!

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

C     Initialization routine for MULTIEDT2

      Edit a shot gather to remove zones with amplitudes higher
      than on adjacent shot gathers

                          R. T. Houck      May,  1994

C      1) Promax EOJ bug circumvented 11/30/94
C      2) Time dependent threshold added  12/15/94
C      3) Exclude multiple records around center shot  8/11/95

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

        SUBROUTINE INIT_MULTIEDT2 (LEN_SAV, ITOOLTYPE)

      #include 'multiedt2.inc'
      #include 'header.inc'
      #include 'mem.inc'

        PARAMETER (LEN_LNAME=6)
        CHARACTER*32 CDESC, AFILE*41, lname*(LEN_LNAME), cfile*41

C....Make sure input is shot records
      IF (IPSORTz .NE. ISINpz) THEN
      CALL EX_ERR_FATAL (`Input must be shot records.')
      END IF

C....Find header entries
       ALL HDR_NAMINFO (`CHAN `, CDESC, LEN, IFMT, IH~CHAN,
       IERR)
       IF (IERR .ME. 0) CALL EX_ERR_FATAL (`CHAN not in header.')
       CALL HDR_NAMINFO (`CDP  `, CDESC,  LEN, IFMT,  IH_CDP, IERR)
       IF (IERR .NE. 0) CALL EX_ERR_FATAL (`CDP not in header.')
       CALL HDR_NAMINFO (`SOURCE `, CDESC, LEN, IFMT,IH_SOURCE,
       IERR)
```

```
      IF (IERR .ME. 0) CALL EX_ERR_FATAL (`SOURCE not in header.')
      CALL HDR_NAMINFO (`AOFFSET `, CDESC, LEN, IFMT,IH_AOFF, IERR)
      IF (IERR .NE. 0) CALL EX_ERR_FATAL (`AOFFSET not in header.')
      CALL HDR_NAMINFO (`SOU_X   `, CDESC, LEN, IFMT, IH_SX, IERR)
      IF (IERR .ME. 0) CALL EX~ERR_FATAL (`SOU_X not in header.')
      CALL HDR_NAMINFO (`SOU_Y   `, CDESC,  LEN, IFMT, IH~SY,  IERR)
      IF (IERR .NE. 0) CALL EX_ERR_FATAL (`SOU_Y not in header.')

C....Get the file and line name for CENSUS output
      CALL EX_CGETPARM (`CFILE   `, 1, CFILE, NCHARS)
      IF ((CFILE .ME. `NONE') .AND. (CFILE .NE.  `none')) THEN
      CALL RTH_OPEN (CFILE, NCHARS, ICU)
      ELSE
      ICU = 0
      END IF
      CALL EX_CGETPARM (`LNAME   `, 1, LNAME, NCHARS)

C....Blank fill the line name, if necessary
      DO I = NCHARS+1, LEN_LNAME
      LNAME(I:I) = ' '
      END DO

C....Get window length and increment for computing amplitude traces
      CALL EX_GETFARM (`TLEN    `, 1, TLEN)
      CALL EX_GETPARM (`TINC    `, 1, TINC)

C....Convert to samples
      LWIN  NINT (TLEN / SAMPRATZ)
      INCWIN = NINT (TINC I SAMPRATz)

C....Number of amplitude values per trace
      NWTRC = 1 + (NUMSMPz-LWIN) / INCWIM

C....Get taper length for muted windows
      CALL EX_GETPARM (`TAPER     `, 1, TAPER)
      LTAPER = NINT (TAPER/SAMPRATz)

C....Allocate memory for taper and generate ramp
      CALL MEM_RESBUFF (LTAPER, IX_TPR, IERR)
      XSTEP = 1. / FLOAT(LTAPER)
      DO I = 1, LTAPER
      RSPACEz(IX_TPR+I-1)  = FLOAT(I)*XSTEP
      END DO

C....Open the CENSUS file and write header info
      IF (ICU .NE.  0) THEN
      WRITE (ICU,'(A)')  LNAME
      WRITE (ICU,*) NWTRC
      WRITE (ICU,*) SAMPRATz
      WRITE (ICU,*) LWIN,  INCWIN, LTAPER
      END IF

C....Get number of adjacent gathers to search
      CALL EX_GETPARM (`NADJG `, 1, NADJG)
```

```
C....Get number of center gathers to exclude
     CALL EX_GETPARM (`NEXCL `, 1 NEXCL)

C....Get replacement value for edited zones
     CALL EX_GETPARM (`REPVAL `, 1, REPVAL)

C....Get time - amplitude threshold pairs
     CALL EX_CGETPARM (`THRESH `, 1 CTEMPz, NCHARS)
     CALL DECODE_PREP (CTEMPz, NCHARS, NCO)
     IPAIRS = NCO/4 + 2
     CALL FLOAT_DECODE (CTEMPz, NCO, RTEMPz(IPAIRS), NVALS, .TRUE.)

C....Generate the time dependent threshold array
     CALL MEM_RESBUFF  (NWTRC, IX_THRESH, IERR)
     NPAIRS = NVALS/2
     CALL MULTIEDT2_THRESH (RTEMPz(IPAIRS), NPAIRS, SAMPRATz*
     LWIN-1),& SAMPRATz"INCWIN, NWTRC, RSPACEz(IX_THRESH))

C....Get output fold QC file name; open if requested
     CALL EX_CGETPARM (`AFILE `, 1, AFILE, NCHARS)
     IF (AFILE(1:4)  .NE.  `none') THEN
     CALL RTH_OPEN (AFILE, NCHARS, IUN)
     ELSE
     IUN = 0
     END IF

C....Allocate memory for time dependent fold QC array
     CALL MEM_RESBUFF  (NUMCDPz*NWTRC, IX_FQC, IERR)
     CALL VCLR (RSPACEz(IX_FQC), 1, NUMCDPzZ*NWTRC)

C....Initialize CDP counters
     MINCDPR = IBIG
     MAXCDPR = -IBIG

C....Start out in fill mode, with the first input shot
     uninitialized
     OUT_PTR = 0
     IN_PTR = 0
     ITRC_IN = 0

C....Counter for trailing shot records being flushed
     IFLUSH = 0

C....Initialize increment between shots, shot interval in groups
     INCSP =  0
     DGDS = 0.

C....Set flag needed to get around Promax bug (no EOJ trace if in
     pipe mode)
     STOPDMP = .FALSE.
     LEN_SAV = LENSAVED
     ITOOLTYPE = ICOMPLEXpz
     RETURN
     END
```

```
**********************************************************************
      SUBROUTINE MULTIEDT2_THRESH (PAIRS, NPAIRS, TO, TINC, NW, THR)
C....Generate an amplitude threshold for each amplitude window
C....Parameters:
      PAIRS  - array of input time-amplitude pairs
      NPAIRS - no. of input pairs
      TO     - time of first amplitude output threshold
      TINC   - time increment between output thresholds
      NW     - no. of thresholds to output
      THR    - threshold array (output)

      REAL PAIRS(NPAIRS), THR(NW)

C....Output Threshold Loop: Generate a threshold for each window
      ITIME = 1
      DO 1W = 1, NW
      TW = TO + (IW-1)*TINC

C....Extrapolate constant thresholds if beyond specified end pairs
      IF (TW .LE. PAIRS(1)) THEN
      THR(IW)  = PAIRS(2)
      ELSE IF (TW .GE. PAIRS(2*NPAIRS-1)) THEN
      THR(IW)  = PAIRS(2*NPAIRS)

C....Interpolate between bracketing pairs
      ELSE

C....Move to the next pair if necessary
      IF (TW .GE. PAIRS(ITIME+2)) THEN
      ITIME = ITIME + 2

C....Skip closely spaced points
      DO WHILE (TW .LT. PAIRS(ITIME))
      ITIME = ITIME + 2
      END DO
      END IF

C....Interpolate
      T1 = PAIRS(ITIME)
      A1 = PAIRS(ITIME+1)
      T2 = PAIRS(ITIME+2)
      A2 = PAIRS(ITIME+3)
      THR(IW)  = A1 + (TW-T1) * (A2-A1)/(T2-T1)
      END IF
      END DO
      RETURN
      END
**********************************************************************
C     Execution Routine for MULTIEDT2
```

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

      SUBROUTINE EXEC_MULTIEDT2   (TRACE, ITHDR, RTHDR)

      #include  'multiedt2.inc'
      #include  'header.inc'
      #include  'mem.inc'

   INTEGER ITHDR (NTHz)
   REAL TRACE(NUMSMPz),  RTHDR(NTHz)

   INTEGER SHOT_NLIST,   SHOT_ILIST, SHOT_TRC, SHOT_HDR,
   SHOT_APTR,  &  SHOT_PREV, SHOT_SP
   INTEGER AMP_NLIST, AMP_ILIST

C --- Output QC file if finished----------------------------------
      IF (CLEANUPz) THEN

C....Close CENSUS file
      IF  (ICU .NE. 0) THEN
      CLOSE   (UNIT=ICU)
      CALL FILE_CLOSE (ICU, IERR)
      END IF

C....Don't output if the file wasn't opened
      IF  (IUN .NE. 0) THEN
      NCDPR =  (MAXCDPR - MINCDPR) / INCCDPz + 1
      WRITE (IUN,*)  NWTRC,  SAMPRATz*INCWIN, NCDPR
      DO I = 1,  NCDPR
      IX = IX_FQC +  (MINCDPR-MINCDPz+I-1)*NWTRC
      LX = IX + NWTRC - 1
      WRITE  (IUN,200) MINCDPR+INCCDPz*(I-1),
      &  (ISPACEz(JX)  JX = IX, LX)
      END DO
      CLOSE   (UNIT=IUN)
      CALL FILE_CLOSE   (IUN, IERR)
      END IF
      RETURN
      END IF

 200  FORMAT  (7X,  'CDP', I10, /, (2513))

C....Ask for a new trace if we dumped the last trace on the
      previous call
      IF  (STOPDMP) THEN
      STOPDMP =  .FALSE.
      CALL EX_FILLMODE
      RETURN
      END IF

C....Set up to flush remaining shots if EOJ trace was just read
      IF ((IFLUSH .EQ. 0).AND. (ITHDR(IEOJz) .EQ. LASTTRpz))  THEN
      OUT_PTR = SHOT_ILIST(IN_PTR,1)
      ITRC_OUT = 1
      IFLUSH = 1
```

```
      END IF

C....Don't send out a trace if everything's been flushed
      IF  (((IFLUSH .NE. 0) .AND. (OUT_PTR .EQ. 0)) .OR.
&     (IFLUSH  .GT. NADJG)) THEN CALL EX_QUITMODE
      RETURN
      END IF

C ********* Fill mode: ****************************************

      IF  (OUT_PTR .EQ. 0)  THEN

C....Construct a new shot gather if this is the first trace
      IF (ITRC_IN .EQ. 0) THEN
      IX_PREV = IN_PTR
      CALL SHOT_NEW (ITHDR(IH_SOURCE), NUMSMPz, NTHz,  MAXDTRz,
&  IX_PREV, IN_PTR)
      ITRC_IN = 1
      END IF

C....Load this trace (and its header) into the shot gather
      structure
      CALL SHOT_LDTRC  (IN_PTR, ITRC_IN, TRACE, ITHDR)

C....Keep track of CDP's actually read
      IF (ITRACE_DEAD(RTHDR(1)) .NE.  1) THEN
      IF (ITHDR(ICDPz) .LT. MINCDPR)  MINCDPR = ITHDR(ICDPz)
      IF (ITHDR(ICDPz) .GT. MAXCDPR)  MAXCDPR = ITHDR(ICDPz)
      END IF

C....If we just loaded the last trace in the ensemble ...
      IF (ITHDR(IEND_ENSz) .EQ. LASTTRpz) THEN

C....Generate an amplitude gather for this shot gather
      IF (SHOT_PREV(IN_PTR) .NE. 0) THEN
      IX_PREV = SHOT_APTR(SHOT_PREV(IN_PTR))
      ELSE IX_PREV = 0
      END IF
      CALL AMP_NEW (IN_PTR, LWIN, INCWIN, NWTRC, IX_PREV, IX_AMP)

C....Determine sort directions if we've read the first two shots
      IF ((SHOT_NLIST(IN_PTR) .EQ. 2) .AND.  (INCSP .EQ.  0)) THEN
      CALL MULTIEDT2_SDIR (IN_PTR, IH_SOURCE,  IH_SX, IH_SY,
&  IH_AOFF, INCSP,  DGDS)
      END IF

C....Set an output pointer if we have enough shot records
      IF (SHOT_NLIST(IN_PTR) .EQ. NADJG+1) THEN
      OUT_PTR = SHOT_ILIST(IN_PTR, 1)
      ITRC_OUT = 1

C....Ask for a new shot record if we need more
      ELSE   ITRC_IN = 0
      CALL EX_FILLMODE
      END IF
```

```
C....Ask for another trace if there are more traces left to read
     ELSE
     ITRC_IN = ITRC_IN + 1
     CALL EX_FILLMODE
     END IF

C ****** Flush mode: ******************************************

C....Edit and output the next trace in this output shot gather
     IF  (OUT_PTR .NE. 0) THEN

C....Write SOURCE to the CENSUS file if this is the first trace
     in the shot
     IF ((ITRC_OUT .EQ. 1)  .AND. (ICU .NE.  0))
     & WRITE  (ICU,400) SHOT_SP(OUT_PTR)

C....Get pointer to first trace sample and to the trace header
     IPTR = SHOT_TRC (OUT_PTR,  ITRC_OUT)
     JPTR = SHOT_HDR (OUT_PTR,  ITRC_OUT)

C....Load the output TRACE and ITHDR array:
     CALL VMOV (RSPACEz(IPTR), 1, TRACE, 1, NUMSMPz)
     CALL VMOV (ISPACEz(JPTR), 1, ITHDR, 1, NTHz)

C....Edit the trace if it's live
     IF (ITRACE_DEAD(ITHDR) .NE.  1) THEN

C....CDP subscript for this trace
     ICDP = (ITHDR(IH_CDP)  - MINCDPz)  I INCCDPz + 1

C....Generate a search list
     CALL MULTIEDT2_SRCHLST (SHOT_APTR(OUT_PTR), ITRC_OUT, NADJG,
     &  NEXCL, INCSP, DGDS, NSRCH, ITEMPz)

C....Check each time window for above-threshold amplitudes
     CALL MULTIEDT2_TCHECK (NWTRC, NSRCH, ITEMPz,
     RTEMPz(NSRCH+1), & ITEMPz(2*NSRCH+1), RSPACEz(IX_THRESH),
     & ISPACEz(IX_FQC), NUMCDPz,  ICDP, NDEAD,
     & ITEMPz (3*NSRCH+1))

C....Write to the CENSUS file if anything was killed
     IF (NDEAD  .NE. 0) THEN
     INDX_CHAN = SHOT_HDR(OUT_PTR,  ITRC_OUT) + IH_CHAN - 1
     CALL MULTIEDT2_CENOUT (ICU,  ISPACEz(INDX_CHAN), NDEAD,
     &  ITEMPz(3*NSRCH+1))
     END IF

C....Kill the trace if all windows are dead
     IF (NDEAD .EQ. NWTRC) THEN
     CALL KILL_TRACE (TRACE, NUMSMPz, SAMPRATz,  ITHDR)
```

```
C....Otherwise, kill any specified time windows
     ELSE IF (NDEAD .GT.  0) THEN
     CALL MULTIEDT2_WKILL (ITEMPz(3*NSRCH+1), NDEAD, LWIN,
     &  INCWIN, NWTRC, REPVAL,  RSPACEz(IX_TPR),
     &  LTAPER, TRACE, NUMSMPz)
     END IF

C****Figure out what to do next:**********************************

C....If there are more traces in this shot
     IF (ITRC_OUT .LT. MAXDTRz) THEN

C....Increment the trace counter and keep dumping
     ITRC_OUT = ITRC_OUT + 1

C....If we've dumped the last trace for this output record ...
     ELSE

C....Set last trace flag in header
     ITHDR(IEND_ENSz) = LASTTRpz

C....Get rid of unneeded shot and amplitude gathers
     IX_PRE = OUT_PTR
     IX_AMP = SHOT_APTR(OUT_PTR)
     CALL SHOT_DELETE (IX_PREV)

C....Make sure there are enough amplitude gathers before
     deleting one
     NEEDED = 2*NADJG + 1 - IFLUSH
     IF (AMP_TLIST(IX_AMP) .EQ. NEEDED) THEN
     IX_FAMP = AMP_ILIST(IX_AMP, 1)
     CALL AMP_DELETE (IX_FAMP)
     END IF

C....Trigger input of a new shot if not flushing trailing records
     IF (IFLUSH .EQ. 0) THEN
     OUT_PTR = 0
     ITRC_IN = 0

C....Stop dumping after flushing this trace
     STOPDMP = .TRUE.

C....Flush the next shot
     ELSE
     IFLUSH = IFLUSH + 1
     OUT_PTR = SHOT_ILIST(IN_PTR,1)
     ITRC_OUT = 1
     END IF
     CALL EX_FLUSHMODE
     END IF

400  FORMAT (`SOURCE `, 16)
     RETURN
     END
```

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

C....SUBROUTINE MULTIEDT2_SDIR (IN_PTR, IH_SOURCE, IH_SX, IH_SY,
     &  IH_AOFF,  INCSP, DODS)

C....Use header entries from the first two input shot records to
     determine the sort directions of the shots and receivers
     #include "mem.inc."

      INTEGER SHOT_ILIST,  SHOT_HDR

C....Get header entries from current input shot

C....Pointer to header of first trace
      IHC1 = SHOT_HDR(IN_PTR, 1)

C....SOURCE
      ISP = ISPACEz(IHC1+IH_SOURCE-1)

C....Shot X,  Y coordinates
      CSX = RSPACEz(IHC1+IH_SX-1)
      CSY = RSPACEz(IHC1~IH_SY-1)

C....Absolute offsets for traces 1 and 2
      AOFF1 = RSPACEz(IHC1_IH_AOFF-1)
      IHC2 = SHOT_HDR(IN_PTR, 2)
      AOFF2 = RSPACEz(IHC2+IH_AOFF-1)

C....Get header entries from previous input shot
      IN_PTR_PREV = SHOT_ILIST(IN_PTR, 1)
      IHP1 = SHOT_HDR(IN_PTR_PREV, 1)
      ISP_PREV = ISPACEz(IHP1+IH_SOURCE-1)
      PSX = RSPACEz(IHP1+IH_SX-1)
      PSY   RSPACEz(IHP1+IH_SY-1)

C****Determine directions:**************************************

C....SOURCE increment between successive shots
      INCSP = ISP - ISP_PREV

C....Change in trace number (SEQNO) corresponding to the same
     surface location
      DS = SQRT ((CSY_PSY)**2 +  (CSY_PSY)**2)
      DGDS = DS I  (AOFF2-AOFF1)
      RETURN
      END

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

      SUBROUTINE MULTIEDT2_SRCHLST (IX_TEST, ITRC, NAG, NEXCL,
      INCSP, &  DGDS, NSRCH, LIST)

C....Generate an amplitude trace search list
```

```
C....Parameters:
     IX_TEST - pointer to amplitude gather being tested
     ITRC - trace no. of trace being tested
     NAG - no. of adjacent gathers to search
     NEXCL - no. of gathers  (in addition to current shot) to
     exclude
     INCSP - nominal SP increment between shots
     DGDS - SP interval in multiples of the receiver group
     interval
     NSRCH - no. of traces in generated search list
     LIST - list of NSRCH pointers to amplitude traces to use for
     comparison

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

      INTEGER IX_TEST, LIST(1)

      #include "mem. inc"
      INTEGER AMP_TRC, AMP_SP, AMP_NLIST, AMP_ILIST, AMP_NTRC
      LOGICAL AMP_LIVE

C....Target trace is always first in the list
      LIST(1) = AMP_TRC(IX_TEST, ITRC)
      NSRCH = 1

C....SP number of target trace
      ISP = AMP_SP(IX~TEST)

C....Set no. of adjacent traces to search - equalize search
      distance along shot and receiver axes
      NAT = NINT (ABS(DGDS) * NAG)

C....Gather Loop: Search each amplitude gather in the list
      DO ILIST = 1, AMP_NLIST(IX_TEST)

C....Point to the search gather and get its SP number
      IX_SRCH = AMP_ILIST(IX_TEST,ILIST)
      JSP = AMP_SP(IX_SRCH)

C....No. of shots before (-) or after (+) target shot
      NSHOTS = (JSP - ISP) / INCSP

C....Make sure it's within the search range (but exclude the target
     shot)
       IF ((IABS(NSHOTS) .LE. NAG) .AND.
      &  (IABS(NSHOTS) .GT. NEXCL)) THEN

C....Set range of trace numbers to search
      JTRC = ITRC + DGDS*NSHOTS
      NBELOW = MIN (JTRC-1, NAT)
      NABOVE = MIN (AMP_NTRC(IX_SRCH)-JTRC, NAT)

C....Force search to be symmetric about JTRC
      NABOUT = MIN (NBELOW, NABOVE)
      JTRCO = JTRC - NABOUT
```

```
      JTRCL = JTRC + NABOUT

C....Add traces to the list if they're not dead;
      Loop not executed if NABOUT<0
      DO JTRC = JTRCO, JTRCL
      IF (AMP_LIVE(IX_SRCH,JTRC))
      THEN
      NSRCH = NSRCH + 1
      LIST(NSRCH) = AMP_TRC(IX_SRCH,  JTRC)
      END IF
      END DO

C....End Search Range Check
      END IF
      END DO
      End Gather Loop
      RETURN
      END

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

      SUBROUTINE MULTIEDT2_TCHECK (NW,  NSRCH, IPTRS, RWRK, IWRK,
      THRESH, &  IFQC, NCDP, ICDP, NKILL, 1W_KILL)

C....At each time window check the amplitude of the first trace in
      the list of search pointers against the average of the rest;
      update fold counter if the window is nonzero and not killed

C....Parameters:
      NW    - no. of amplitude windows per trace
      NSRCH - no. of traces in the search list, including the one
                being checked
      IPTRS - list of pointers to the first sample in each search
                trace
      RWRK    - work array for median (NSRCH elements)
      IWRK    - work array for median (NSRCH elements)
      THRESH  - relative amplitude threshold array (NW elements)
      IFQC    - time variant fold function for each CDP (updated)
      NCDP    - no. of CDP's in IFQC  (input)
      ICDP    - CDP sequence no. corresponding to TRACE (input)
      NKILL   - no. of windows to be killed in this trace (output)
      1W_KILL - list of window numbers to be killed

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

        INTEGER IPTRS(NSRCH), IWRK(NSRCH),  IFQC(NW,NCDP),
        IW_KILL(NW), REAL RWRK(NSRCH), THRESH(NW)

        #include "mem.inc"

        NKILL = 0

C....Nothing to search
      IF (NSRCH .EQ. 1)
      RETURN
```

```
      DO IW = 1, NW

C....Get amplitude to be checked (first pointer)
      IX_AMPW = IPTRS(1) + 1W - 1
      SAMP = RSPACEz(IX_AMPW)

C....Compute median amplitude of search list traces in this window
      NMED = 0
      DO IS = 2, NSRCH
      AMP = RSPACEz(IPTRS(IS)+IW-1)

C....Exclude hard zeroes
      IF (AMP .NE. 0) THEN
      NMED = NMED + 1
      RWRK(NMED)  = AMP
      END IF
      END DO

      CALL RMEDIAN (RWRK, NMED, :WRK,  IMED)
      AMED = RWRK(IMED)

C....Check against threshold; flag for kill if above
      IF (SAMP .CT. THRESH(IW)*AMED) THEN
      NKILL = NKILL + 1
      IW_KILL(NKILL)  = 1W

C....Zero the killed amplitude
      RSPACEz(IX_AMPW) = 0.

C....Accumulate into fold count if nonzero
      ELSE     IF (SAMP .NE. C)  IFQC(IW,ICDP)  = IFQC(IW,ICDP) + 1
      END IF
      END DO
      RETURN
      END
*****************************************************************

      SUBROUTINE MULTIEDT2_CENOUT (ICU, ICHAN, NDEAD, IWDEAD)

      INTECER IWDEAD (NEAD)

      IF (ICU .EQ. 0) RETURN

      WRITE (ICU,410) ICHAN, NDEAD
      WRITE (ICU,411) (IWDEAD(I), I =  1, NDEAD)

410   FORMAT (2I5)

411   FORMAT (26I3)
      RETURN
      END
```

```
**********************************************************************
      SUBROUTINE MULTIEDT2_WKILL (IW_KILL, NKILL, LWIN, INCWIN,
      NWTRC, & REPVAL,  TAPER,  LTAPER, TRACE, NSMP)
C....Kill a specified list of windows in TRACE, updating fold
      counter

C....Parameters:
      IW_KILL - list of windows to be killed
      NKILL   - no. of windows to kill  (length of IW_KILL)
      NWTRC   - total no. of windows in the trace
      LWIN    - amplitude window length for killing bursts
                (input)
      INCWIN  - amplitude window increment (input)
      REPVAL  - replacement value for killed samples
      TAPER   - linear ramp for tapering edits
      LTAPER  - no. of elements in TAPER
      TRACE   - trace samples  (updated)
      NSMP    - no. of samples in TRACE (input)

      INTEGER IW_KILL(NKILL)
      REAL TAPER(LTAPER), TRACE(NSMP)
      LOGICAL LIVE

C....Offset to the center INCWIN samples in the window
      ICENT = (LWIN - INCWIN) / 2

C....Kill Loop: Zero each window in the list
      DO IK = 1, NKILL

C....Set the window number to be killed
      IW = IW_KILL(IK)

C....Set sample range for kill
      IKILL = (IW-1)*INCWIN + 1 + ICENT
      LKILL = IKILL + INCWIN - 1
C....Start killing the first window at sample 1
      IF (IW .EQ. 1)  IKILL = 1

C....Kill the last window to the end of the trace
      IF (IW .EQ. NWTRC)  LKILL = NSMP

C....Kill the specified range of samples
      DO I = IKILL, LKILL
      TRACE(I) = REPVAL
      END DO

C....Taper down to killed window if previous window was alive
      IF (IK  .EQ. 1) THEN
      LIVE = IW_KILL(IK) .GT. 1
      ELSE
      LIVE = IW_KILL(IK-1)  .NE.   (IW-1)
      END IF
      IF (LIVE) THEN
```

```
      NTAPER = MIN (IKILL-1. LTAPER)
      CALL VMUL (TRACE(IKILL-1), -1. TAPER, 1, TRACE(IKILL-1), -1,
      TAPER)
      END IF

C....Taper up from killed window if next window is alive
      IF  (IK .EQ. UKILL) THEN
      LIVE = IW_KILL(IK) .LT. NWTRC
      ELSE
      LIVE = IW_KILL(IK+1) .NE. (IW+1)
      END IF
      IF  (LIVE) THEN
      NTAPER = MIN (NSMP-LKILL,  LTAPER)
      CALL VMUL  (TRACE(LKILL+1),  1, TAPER, 1, TRACE(LKILL+1),
      & 1, NTAPER)
      END IF
      END DO
      RETURN
      END

*********************************************************************
TITLE:                    mu1tiedt2.inc

TIME PRINTED:             Thu May 07 08:04:30 1998

TIME QUEUED:              Thu May 07 08:04:30 1998

PRINTED AT:               ipO (hplj-3) @ dlrOO7

SUBMITTED BY:             mrjenker@dlrOO7

DELIVER TO:               mrjenker@d1r007

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

FLAG VALUES:
a=0,  d=a,  f=, h=,  j='., 1=66, p=10, s=Courier, z=01 A=11  B=gn,
0='.,H=dlrOO7,   J=+, L=+, N=1,  P=lpO:lpO, x=IBM-850,  1=!, 2=i,
3=.', 4=:., 5=.'

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

C....Include file for MULTIEDT
      #include "global.inc"
      INTEGER OUT_PTR
      LOGICAL STOPDMP

COMMON /SAVED_PARMS/ SAVE1z, ICU, IH_CHAN  IH_CDP,  IH_SOURCE,
&    IH_AOFF, IH_SX. IH~SY, LWIN, INCWIN, NWTRC,
&    LTAPER, IX_TPR. NADJG, REPVAL, NEXCL,  INCSP,
&    DGDS, IX_THRESH, IUN, IX_FQC, MINCDPR, MAXCDPR,
&    IN_PTR, OUT_PTR, ITRC_IN, ITRC_OUT, NSRCH,
&    IFLUSH, STOPDMP
```

```
      PARAMETER (IBIG=999999)
      INTEGER LENSAVED
      DATA LENSAVED /30/

C..ICU        - LUN for CENSUS file
C..IH_CHAN    - subscript of CHAN in header
C..IH_CDP     - subscript of CDP in header
C..IH_SOURCE  - subscript of SOURCE in header
C..IH_AOFFSET- subscript of AOFFSET in header
C..IM_SX      - subscript of SOU_X in header
C..IH_SY      - subscript of SOU_Y in header
C..LWIN       - amplitude window length (samples)
C..INCWIN     - amplitude window increment
C..NWTRC      - no. of amplitude windows per trace
C..LTAPER     - no. of samples in taper
C..IX_TPR     - supscript of taper up-ramp in RSPACEZ
C..REPVAL     - replacement value for edited samples
C..NADJG      - no. of adjacent shot gathers to search
C..NEXCL      - no. of off-center gathers to exclude from the search
C..INCSP      - nominal increment in SOURCE between shots
C..DGDS       - interval between shots in units of receiver groups
C..IX_THRESH  - subscript of threshold array in RSPACEz
C..IUN        - Fortran LUN for output fold QC file
C..IX_FQC     - subscript of time-variant fold array in RSPACEz
C..MINCDPR    - minimum CDP read
C..MAXCDPR    - maximum CDP read
C..IN_PTR     - pointer to shot gather currently being input
C..OUT_PTR    - pointer to shot gather currently being output
C..ITRC_IN    - trace no. currently being input
C..ITRC_OUT   - trace no. currently being output
C..NSRCH      - no. of traces in the current search list
C..IFLUSH     - counter for trailing shot records being flushed
C..STOPDMP    - TRUE if we should switch to fill mode on the next
                call; needed because you have to be in fill mode to
                get an EOJ trace

*****************************************************************
TITLE:                  amp.f

TIME PRINTED:           Thu May 07 08:16:33 1998

TIME QUEUED:            Thu May 07 08:16:32 1998

PRINTED AT:             lpO  (hplj-3) @ dlr007

SUBMI~TED BY:           mrjenker@dlrOO7

DELIVER TO:             mrjenker@dlr007

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

FLAG VALUES:
a=0,  d=a, f=, h=~  ~=!, 1=66, p=10, s=Courier,  z=0, A=1, B=gn,
0=!, H=dlrOO7,  J=+, L=+, N=1,  P=lpO:lpO,  x=IBM-850, 1=!, 2=!,
3=!,  4=!, 5=!
```

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

      SUBROUTINE AMP~NEW (IX_SHT, NPW,  INC, NW, IX_PREV, IX_THIS)

C....Construct a gather of amplitude traces from a shot gather

      #include "mem.inc"

C....Parameters:
      IX_SHT  - pointer to input shot gather object
      NPW     - no. of trace samples in an amplitude window
      INC     - amplitude window increment (trace samples)
      NW      - no. of amplitude windows per trace
      IX_PREV - pointer to previous amplitude gather
      IX_THIS - pointer to this amplitude gather (output)

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

       PARAMETER (NUMHDR=6)
       INTEGER SHOT_NSMP, SHOT_NTRC  SHOT_SP, SHOT_TRC

C....Get data from the input shot gather
      NSMP = SHOT_NSMP(IX_SHT)
      NTRC = SHOT_NTRC(IX_SHT)

C....Allocate memory and get pointer for this object
      CALL MEM_RESBUFF (NW*NTRC+NUMHDR, IX_THIS, IERR)
      IF (IERR  .NE. 0) CALL EX_ERR_FATAL (`No space left.')

C....Load header info
      ISPACEz(IX_THIS)   = IX_PREV
      ISPACEz(IX_THIS+2) = IX_SHT
      ISPACEz(IX_THIS+3) = SHOT_SP(IX_SHT)
      ISPACEz(IX_THIS+4) = NTRC
      ISPACEz(IX_THIS+5) = NW

C....Compute and load amplitudes
      DO ITRC = 1, NTRC
      ISTRT = IX_THIS + (ITRC-1)*NW + NUMHDR
      DO IW = 1, NW
      ISMP = (IW-1)*INC
      IPTR = SHOT_TRC(IX_SHT,ITRC) + ISMP
      RSPACEz(ISTRT+IW-1) = AMP_RMS  (RSPACEz(IPTR), NPW)
      END DO

C....Set the amplitude pointer in the shot gather
      CALL SHOT_SET_APTR (IX_SHT, IX_THIS)

C....If there's a previous gather, update its NEXT pointer
      IF (IX_PREV .NE. 0) ISPACEz(IX_PREV+1) = IX_THIS

C....This gather is new, so it doesn't have a NEXT pointer
      ISPACEz(IX_THIS+1) = 0
      RETURN    END
```

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

      REAL FUNCTION AMP_RMS  (X, NP)

C....Compute RMS amplitude of NP points in X

      REAL X(NP)
      SSQ = 0.
      DC I = 1,  NP
      SSQ = ~SQ + x(I)*x(I)
      END DO
      AMP_RMS = SQRT  (SSQ/NP)
      RETURN
      END

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

        INTEGER FUNCTION AMP_NLIST (IX_THIS)

C....Return the number of amplitude gathers in the list

      #include "mem.inc"

C....Return zero if the list is empty
      IF (IX_THIS .EQ. 0) THEN
      AMP_NLIST  = 0
      RETURN

C....Otherwise, count the passed object
      ELSE
      NLIST = 1
      END IF

C....Search forward
      IX_NEXT = ISPACEz(IX_THIS+1)
      DO WHILE (IX_NEXT .NE. 0)
      NLIST = NLIST + 1
      IX_NEXT =  ISPACEz(IX_NEXT+1)
      END DO

C....Search backwards
      IX_NEXT = ISPACEz(IX_THIS)
      DO WHILE (IX_NEXT  .NE. 0)
      NLIST = NLIST + 1
      IX_NEXT = ISPACEz(IX_NEXT)
      END DO
      AMP_NLIST = NLIST
      RETURN     END

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

        INTEGER FUNCTION AMP_ILIST (IX_THIS, IL)

C....Return a pointer to the ILth amplitude gather in the list
```

```
      #include "mem.inc"

C....Return zero if the list is empty
      IF (IX_THIS .EQ. 0) THEN
      AMP_ILIST = 0
      RETURN
      END IF

C....Search backwards to the start of the list
      IX_FIRST = IX_THIS
      IX_NEXT = ISPACEz(IX_THIS)
      DO WHILE (IX_NEXT  .NE. 0)
      IX_FIRST = IX_NEXT
      IX_NEXT = ISPACEz(IX_NEXT)
      END DO

C....Search forward to ILth element
      ILIST = 1
      IX_CURR = IX_FIRST
      IX_NEXT = ISPACEz(IX_FIRST+1)
      DO WHILE ((IX_NEXT  .NE. 0) .AND.  (ILIST .NE. IL))
      ILIST = ILIST + 1
      IX_CURR = IX_NEXT
      IX_NEXT = ISPACEz(IX_CURR+1)
      END DO

C....Return a zero pointer if we didn't find an ILth element
      IF (ILIST  .NE. IL) THEN
      AMP_ILIST = 0
      ELSE     AMP_ILIST = IX_CURR
      END IF
      RETURN
      END

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

       INTEGER FUNCTION AMP_TRC  (IX_THIS, ITRC)

C....Return a pointer to trace ITRC

      #include  "mem.inc"

C....PARAMETER (NUMHDR= 6)
      NTRC = ISPACEz(IX_THIS+4)
      NW = ISPACEz(IX_THIS+5)

C....Return a zero pointer if ITRC is out of range
      IF  ((ITRC .LT. 1) .OR. (ITRC .GT. NTRC)) THEN
      AMP_TRC = 0
      RETURN
      END IF

      AMP_TRC = IX_THIS + NUMHDR + (ITRC-1)*NW
      RETURN      END
```

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

      INTEGER FUNCTION AMP_SP     (IX_THIS)

C....Return the SP number

      #include    "mem.inc"

      AMP_SP = ISPACEZ(IX_THIS+3)
      RETURN
      END

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

      INTEGER FUNCTION AMP_NTRC     (IX_THIS)

C....Return the number of traces in the gather

      #include "mem.inc"

      AMP_NTRC = ISPACEz(IX_THIS+4)
      RETURN
      END

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

       LOGICAL FUNCTION AMP_LIVE (IX_THIS, ITRC)

C....Return  .TRUE.  If ITRC has any non-zero amplitudes

      #include "mem.inc"

       PARAMETER (NUMHDR=6)

       NW = ISPACEz(IX_THIS+5)
       IX_TRC = IX_THIS + NUMHDR + (ITRC-1)*NW
       1W = 1
       DO WHILE ((RSPACEz(IX_TRC+IW-1) .EQ. 0) .AND. (1W .LT. NW))
       1W = 1W + 1
        END DO
       AMP_LIVE = RSPACEz(IX_TRC+IW-1)   .NE.   0
       RETURN
       END

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

       SUBROUTINE AMP_SET_SPTR (IX_THIS, IX_SHOT)

C....Set the pointer to the associated shot gather

      #include "mem.inc"
```

```
      ISPACEz(IX_THIS+2) = IX_SHOT
      RETURN
      END
*********************************************************************

       SUBROUTINE AMP_DELETE (IX_THIS)

C....Delete this amplitude gather

      #include "mem.inc"

      PARAMETER (NUMHDR=6)

      integer shot_sp

C....Zero the amplitude pointer in the associated shot, if it
      still exists
      IX_SHOT = ISPACEz(IX_THIS+2)
      IF (IX_SHOT .NE. 0)  CALL SHOT_SET_APTR (IX_SHOT, 0)

C....Reset pointers in adjacent amplitude objects
      IX_PREV = ISPACEz(IX_THIS)
      IX_NEXT = ISPACEZ(IX_THIS+1)
      IF (IX_PREV  .NE. 0)  ISPACEz(IX_PREV+1) = IX_NEXT
      IF (IX_NEXT  .NE. 0)  ISPACEz(IX_NEXT) = IX_PREV

C....Free memory
      NTRC = ISPACEz(IX_THIS+4)
      NW = ISPACEz(IX_THIS+S)
      NEUF = NUMHDR + NTRC*NW
      CALL MEM_FREEBUFF (NBUF, IX_THIS, IERR)
      RETURN
      END
*********************************************************************

   SUBROUTINE SHOT_NEW (ISP,  NSMP, NTH, NTRC, IX_PREV, IX_THIS)

C....Construct a shot gather object

      #include "mem.inc"

C....Parameters:
      ISP       - SP number (SOURCE from trace header)
      NSMP      - no. of samples per trace
      NTH       - length of trace header
      NTRC      - no. of traces in this shot
      IX_PREV   - pointer to previous shot gather
      IX_THIS   - pointer to this shot gather (output)

      PARAMETER (NUMHDR=7)

C....Allocate memory and get pointer for this object
      CALL MEM_RESBUFF ((NTH+NSMP)*NTRC+NUMHDR, IXTHIS, IERR)
      IF (IERR .NE. 0) CALL EX_ERR_FATAL (`No space left.')
```

```
C....Load header info
     ISPACEz(IX-THIS)    = IX_PREV
     ISPACEz(IX_THIS+3) = ISP
     ISPACEz(IX_THIS+4) = NTRC
     ISPACEz(IX_THIS+5) = NTH
     ISPACEz(IX_THIS+6) =  NSMP

C....No amplitude gather yet
     ISPACEz(IX_THIS+2) = 0

C....If there's a previous gather, update its NEXT pointer
     IF (IX_PREV .NE. 0)  ISPACEz(IX_PREV+1) = IX_THIS

C....This gather is new, so it doesn't have a NEXT pointer
     ISPACEz(IX_THIS+1) = 0
     RETURN
     END

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

      SUBROUTINE SHOT_LDTRC (IX_THIS, ITRC, TRACE, HDR)

C....Load trace ITRC (and its header) into THIS shot

      #include "mem.inc"

      INTEGER SHOT_TRC, SHOT_HDR

C....Load the samples
     IX_TRC = SHOT_TRC (IX_THIS, ITRC)
     NSMP   ISPACEz(IX_THIS+6)
     CALL VMOV (TRACE,  1, RESPACEz(IX_TRC), 1, NSMP)

C....Load the header
     IX_HDR = SHOT_HDR (IX_THIS, ITRC)
     NHDR = ISPACEz(IX_THIS+5)
     CALL VMOV (HDR, 1, RSPACEz(IX_HDR), 1, NHDR)
     RETURN
     END

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

      INTEGER FUNCTION SHOT_NLIST (IX_THIS)

C....Return the number of shot gathers in the list

      #include  mem. inc

C....Return zero if the list is empty
     IF (IX_THIS  .EQ. 0) THEN
     SHOT_NLIST = 0
     RETURN
```

```
C....Otherwise, count the passed object
      ELSE
      NLIST = 1
      END IF

C....Search forward
      IX_NEXT = ISPACEz(IX_THIS+1)
      DO WHILE (IX_NEXT .NE. 0)
      NLIST = NLIST + 1
      IX_NEXT = ISPACEz(IX_NEXT+1)
      END DO

C....Search backwards
      IX_NEXT = ISPACEz(IX_THIS)
      DO WHILE (IX_NEXT .NE. 0)
      NLIST = NLIST + 1
      IX_NEXT = ISPACEz(IX_NEXT)
      END DO

      SHOT_NLIST = NLIST
      RETURN
      END

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

        INTEGER FUNCTION SHOT_ILIST (IX_THIS, IL)

C....Return a pointer to the ILth shot gather in the list

      #include "mem.inc"

C....Return zero if the list is empty
      IF (IX_THIS .EQ.  0) THEN
      SHOT_ILIST = 0
      RETURN      END IF

 C....Search backwards to the start of the list
       IX_FIRST = IX_THIS
       IX_NEXT = ISPACEz(IX_THIS)
       DO WHILE (IX_NEXT  .NE. 0)
       IX_FIRST = IX_NEXT
       IX_NEXT = ISPACEz(IX_NEXT)
       END DO

  C....Search forward to ILth element
       ILIST = 1
       IX_CURR = IX_FIRST
       IX_NEXT = ISPACEz(IX_FIRST+1)
       DO WHILE ((IX_NEXT .NE. 0) .AND.  (ILIST .NE. IL))
       ILIST = ILIST + 1
       IX_CURR = IX_NEXT
       IX_NEXT = ISPACEz(IX_CURR+1)
       END DO
```

```
C....Return a zero pointer if we didn't find an ILth element
      IF (ILIST  .NE. IL) THEN
      SHOT_ILIST = 0
      ELSE
      SHOT_ILIST = IX_CURR
      END IF
      RETURN
      END

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

       INTEGER FUNCTION SHOT_TRC (IX_THIS, ITRC)

C....Return a pointer to trace ITRC

      #include  "mem.inc"

      PARAMETER (NUMHDR=7)

      NTRC   ISPACEz(IX_THIS+4)
      NTH =  ISPACEz(IX_THIS+S)
      NSMP = ISPACEz(IX_THIS+6)

C....Return a zero pointer if ITRC is out of range
      IF ((ITRC .LT. 1)  .OR. (ITRC .GT. NTRC)) THEN
      SHOT_TRC = 0
      RETURN
      END IF

      SHOT_TRC = IX_THIS + NUMHDR + NTH*NTRC + (ITRC-1)*NSMP
      RETURN
      END

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

      INTEGER FUNCTION SHOT_HDR (IX_THIS, ITRC)

C....Return a pointer to the header for trace ITRC

      #include  "mem.inc"

      PARAMETER (NUMHDR=7)

      NTRC = ISPACEz(IX_THIS+4)
      NTH = ISPACEz(IX_THIS+5)

C....Return a zero pointer if ITRC is out of range
      IF ((ITRC .LT. 1)  .OR. (ITRC .GT. NTRC)) THEN
      SHOT_HDR = 0
      RETURN
      END IF

      SHOT_HDR = IX_THIS + NUMHDR + (ITRC_1)*NTH
      RETURN
      END
```

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

      INTEGER FUNCTION SHOT_PREV (IX_THIS)

C....Return pointer to the previous shot in the list

      #include "mem.inc"

      SHOT_PREV = ISPACEz(IX_THIS)
      RETURN
      END

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

      INTEGER FUNCTION SHOT_NEXT (IX_THIS)

C....Return pointer to the next shot in the list

      #include "mem.inc"

      SHOT_NEXT = ISPACEz(IX_THIS+1)
      RETURN
      END

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

      INTEGER FUNCTION SHOT_APTR (IX_THIS)

C....Return the pointer to the associated amplitude gather
      #include "mem.inc"
      SHOT_APTR = ISPACEZ(IX_THIS+2)
      RETURN    END

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

      SUBROUTINE SHOT_SET_APTR   (IX_THIS, IX_AMP)

C....Set the pointer to the associated amplitude gather

      #include "mem.inc"

      ISPACEz(IX_THIS+2)  = IX_AMP
      RETURN       END

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

       INTEGER FUNCTION SHOT_SP (IX_THIS)

C....Return the SP number

      #include "mem.inc"

      SHOT_SP = ISPACEz(IX_THIS+3)
      RETURN     END
```

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

        INTEGER FUNCTION SHOT_NTRC (IX_THIS)

 C....Return the number of traces in the gather

      #include "mem.inc"

      SHOT_NTRC = ISPACEz(IX_THIS+4)
      RETURN
      END

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

        INTEGER FUNCTION SHOT_NSMP (IX_THIS)

C....Return the number of samples per trace

      #include "mem.inc"

      SHOT_NSMP = ISPACEz(IX_THIS+6)
      RETURN
      END

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

        SUBROUTINE SHOT_DELETE (IX_THIS)

C....Delete this shot gather

      #include "mem.inc."

        PARAMETER (NUMHDR=7)

C....Zero the pointer to the associated amplitude gather, if any
      IX_AMP = ISPACEz(IX_THIS+2)
      IF (IX_AMP .NE. 0)  CALL AMP_SET_SPTR (IX_AMP, 0)

C....Reset pointers in adjacent items
      IX_PREV = ISPACEz(IX_THIS)
      IX_NEXT   ISPACEz(IX_THIS+1)
      IF (IX_PREV .NE. 0)  ISPACEz(IX_PREV+1) = IX_NEXT
      IF (IX_NEXT .NE. 0)  ISPACEz(IX_NEXT) = IX_PREV

C....Free memory
      NTRC = ISPACEz(IX_THIS+4)
      NHDR = ISPACEz(IX_THIS+5)
      NSMP = ISPACEz(IX_THIS+6)
      NEUF = NUMHDR + NTRC*(NHDR+NSMP)
      CALL MEN_FREEBUFF (NBUF, IX_THIS, IERR)
      RETURN
      END
```

```
**********************************************************************
      SUBROUTINE AMP_SRCHLST (IX_THIS, ITRC, NAG, NAT, NSRCH, LIST)
C....Generate an amplitude trace search list
C....Parameters:
      IX_THIS - pointer to amplitude gather being tested
      ITRO - trace no. of trace being tested
      NAG - no. of adjacent gathers to search
      NAT - no. of adjacent tracess to search
      MXTRC - max number of traces in a gather
      NSRCH - no. of traces in generated search list
      LIST - list of NSRCH pointers to amplitude traces to use for
            comparison
      INTEGER LIST(2*NAG*(2*NAT+1)+1)
      #include "mem.inc"
      INTEGER AMP_TRC, AMP_SP, AMP_INCSP, AMP_NLIST, AMP_ILIST,
      AMP_NTRC,  LOGICAL AMP_LIVE
C....Target trace is always first in the list
      LIST(1) = AMP_TRC(IX_THIS, ITRC)
      NSRCH = 1
C....SP number of target trace
      ISP = AMP_SP(IX_THIS)
C....Maximum SP distance to search
      MXDSP = NAG * AMP_INCSP(IX_THIS)
C....Gather Loop: Search each amplitude gather in the list
      DO ILIST = 1, AMP_NLIST(IX_THIS)
C....Point to the search gather and get its SP number
      IX_SRCH = AMP_ILIST(IX_THIS,ILIST)
      JSP = AMP_SP(IX_SRCH)
      NDSP = ISP - JSP
C....Make sure it's within the search range (but exclude the
      target shot)
      IF ((IABS(NDSP) .LE. MXDSP) .AND. (IABS(NDSP) .NE. 0)) THEN
C....Set range of trace numbers to search
      JTRCO = iMAX (1, ITRC+DRDS*NDSP_NAT)
      JTRCL = iMIN (ITRC+DRDS*NDSP+NAT, AMP_NTRC(IX_SRCH))
C....Add traces to the list if they're not dead
      DO JTRC = JTRCO, JTRCL
      IF (AMP_LIVE(IX_SRCH,JTRC)) THEN
      NSRCH = NSRCH + 1
      LIST(NSRCH) = AMP_TRC(IX_SRCH, JTRC)
```

```
      END IF
      END DO

   C....End Search Range Check
      END IF
      END DO
       End Gather Loop
      RETURN
      END

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

                        MEGA EDIT SOFTWARE

#ifndef    _MEGAEDT_H
#define    _MEGAEDT_H

#include   <iostream.h>
#include   <strstream.h>
#include   <fstream.h>

#include   <agC/Arr.h>
#include   <agc/Float2.h>

#include   "../include/EnsSocket.h"
#include   "ShotGather.h"

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

                    Control Class for MegaEdt
          Derived from Ensemole Socket Tool class EnsSocket
                    R.T. Houck   November 1995

              Improved Performance Version  June 1997

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

class MegaEdt  : public EnsSocket
(
public:

//    Constructor
      MegaEdt (int AC, char** AV);

//    Destructor
      -MegaEdt ();

//    Process shot gathers from Promax flow
      virtual int ProcessEns (int nStored, float** traces,
      int** iheads, float** rheads);

//    Determine the next shot to be output
      (return -1 if none are ready)
      int findNextOut();
```

```
    //    Edit shot iShot
          int shotEdit  (int iShot);

    //    Delete any unneeded shot gathers; return the number deleted
          int deleteGathers ();

    //    Output a fold file if one was generated
          void writeFoldFile ();

    private:

    int nadjg;       // No. of adjacent shot gathers to search
    int nexcl;       // No. of near shot gathers to exclude
    int maxcheck;    // Max number of traces to check
    float searchDist; // Search distance in shot-receiver space
    int maxSearch;     // Max number of traces per streamer to search
    int chans_per_streamer;   // Number of channels per streamer
    int nStreamers;
    AGArr<ShotGather*> Shots; // Array of shot gathers
    int lwin, incwin;   // Amplitude window length, increment (samples)
    int nwtrc;          // No. of amplitude windows per trace
    AGFloatl Thresh;    // Time-dependent amplitude threshold
    int ltaper;         // Blanking taper length (samples)
    AGFloatl Taper;     // Taper ramp coefficents
    int censusFile;     // Flag for writing a CENSUS file
    ofstream Census;    // CENSUS file stream
    char* line;         // Name of sail line
    int foldFile;       // Flag for writing a fold file
    ofstream fold_strm;// Fold file stream
    int numcdp;         // Number of CDP's
    int mincdp, maxcdp;// Initial CDP # range from GlobalRuntime
    int inccdp;         // Initial CDP # increment from GlobalRuntime
    int mincdp_r, maxcdp_r;  //CDP # range actually processed
    int ihCDP;          // Header index for CDP
    AGFloat2 fold_qc;  // Time-dependent fold array (nwtrc X numcdp)
    int ihSOURCE, ihCHAN; // Header indices for SOURCE and CHAN
    int headerGeom;     // Flag for geometry in trace headers
    int ihSOU_X, ihSOU_Y; // Header indices for SOU_X and SOU_Y
    int ihREC_X, I_REC_Y; // Header indices for REC_X and REC_Y
    int startArray;     // Array no. fired on first shot
    int nArrays;        // No. of source arrays
    float XArray;       // Crossline (X) distance between arrays
    float ShotInt;      // Inline (Y) distance betwen shots
    int startSrc;       // SOURCE of first shot
    int nextOut;        // Subscript of next shot to output
    int moreShots;      // True if there are more shots expected

    //    Generate the time-dependent threshold from input time-
          amplitude pairs
          AGFloatl tThresh (const AGFloatl& taPairs);

    //    Find traces inside the search distance and update the
          amplitude table
          int findTraces (int iEdit, int itrc, AmpTable & ampTbl);
```

```
#endif          //    _MECAEDT_H

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

                                MegaEdt
          Amplitude-based surgical blanking for multi-source,
                          multi-streamer data
                      R.T. Houck    November 1995

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

//    include ProMAX prototypes and globals
      #include "cpromax.h"
      #include "cglobal.h"
      #include "cSocketTool.h"
      #include "MegaEdt.h"

//    Initialize static members
      int ShotGeom :: nspread = 0;
      float * ShotGeom :: recSprx = NULL;
      float * ShotGeom :: recspry = NULL;
      const int KillFlags :: nLine = 80;

      #include <unistd.h>

      #include <new.h>
      void noMemory()
(
      cerr « "MegaEdt couldn't find enough memory." « endl;
      exit(-1);
)
      void main (int argc, char** argv)
(
      set_new_handler (&noMemory);

//    Start the MegaEdt run by getting the Promax input parameters
      MegaEdt MegaEdtRun (argc, argv);

//    Process the trace data and close the Promax socket
      int ProcessError = MegaEdtRun.ProcessLoop();

//    Write out a fold file if one was requested
      MegaEdtRun.writeFoldFile();

      stCloseSocketLink();
)
```

```
//************************************************************************
//
//      MegaEdt: Multi-source, multi-streamer amplitude editing
//
//                      R.T. Houck     November 199S
//
//                Improved Performance Version  June 1997
//
//************************************************************************

//    Include Promax prototypes and globals
      #include "cpromax.h"
      #include "cglobal.h"
      #include "cSocketTool.h"

//    Advance's classes
      #include <agC/String.h>
      #include <agC/Float2.h>

//    Homemade classes
      #include "MegaEdt.h"
      #include "KillFlags.h"

//    Menu decoding function
      AGFloatl decodePairs   (char* cbuf);

//    Manual instantiation of ShotGather array template
      #if defined (RS6000)
      #include <agc/Ar.c>
      #include <a~gC/Arr .
      #pragma define (AGArr<ShotGather*>)
      #endif

//    Standard C++ classes
      #include <unistd.h>
      #include <stdio.h>
      #include <iostream.h>
      #include <iomanip.h>

//************************************************************************
//
//      Constructor: Read the Promax menu and get header indices
//
//************************************************************************

MegaEdt  :: MegaEdt  (int AC, char** AV)
            EnsSocket (AC,AV), searchDist(0), nextout(-1),
            moreShots(1)
(
   stSetToolName ( "MegaEdt");

// Read the menu and initialize stuff
```

```
//    Set up string stream for error message formatting
      const int msgLen=81;     char mess (msgLen]
      ostrstream msg (mess, msgLen)

//    Get the no. of adjacent gathers
      exParGetInt  ("NADJG", &nadjg);

//    No. of near-shot gathers to exclude
      exParGetint  ("NEXCL", &nexcl);

//    Max no of traces for amp comparison
      exParGetInt  ("MAXCHECK", &maxcheck);

//    Get running amplitude window length, increment
      float wlen,  winc;
      exParGetFloat     ("WLEN", &wlen);
      exParGetFloat     ("WINC", &winc);

//    Compute lengths in samples
      float dt = gr->samprat;
      lwin = NINT (wlen/dt);
      incwin = NINT (winc/dt);

//    No. of amplitude windows per trace
      nwtrc = 1 + (nsmp-lwin) / incwin;

//    Get taper length for muted windows and generate taper
      coefficents
      float taper, taper0;
      exParGetFloat     ("TAPER", &taper);
      itaper = NINT (taper/dt); taper0 = 1./float(ltaper);
      Taper = AGFloatl (taper0,  taper0, ltaper);

//    Get time - amplitude threshold pairs and generate the
      threshold function
      char* cbuf, cbuf2;
      int nchars, ic, nDecoded, nvals, nPairs, gatesFlag;
      exParGetstring ("THRESH",  &cbuf);
      Thresh = tThresh (decodePairs     (cbuf));
           if (Thresh.n()==0) stErrFatal  ("Couldn't generate
      threshold function.");

//    Get the name of the CENSUS file and open a stream if
      requested
      exParGetString ("CFILE", &cbuf);
      AGstring cstr(cbuf);
      censusFile =  (toupper(cstr) ! = "NONE");
      if  (censusFile)
      Census.open (cbuf,  ios::out);
      exParGetstring ("LNAME",  &line);
      Census « line « '\n'
             « nwtrc « '\n'
             « dt « '\n'
            « lwin « ' '  « incwin « ' '  « ltaper « endl;
  )
```

```
    //    Find out where the geometry is supposed to come from
          exParGetInt    ("HDRGEOM", &headerGeom);
          eaderGeom = headerGeom % 2;

    //    Read geometry parameters if geometry doesn't come from the
          headers
          if  (!headerGeom)  (

          exParGetInt     ("NARRAYS", &nArrays);
          exParGetInt     ("STARTARY", &startArray);
          if (startArray > nArrays)
          msg « "Start array is invalid; must be .le." « nArrays,
          stErrFatal  (mess);
    )
          exParGetFloat    ("XARRAY", &XArray);
          exParGetFloat    ("SHOTINT", &ShotInt);

          exParGetInt     ("NSTREAM", &nStreamers);
          xParGetlnt     ("SCHANS", &chans_per_streamer);
          int gchans = nStreamers * chans_per_streamer;
          if (gchans    != maxtr)
          msg « "Nominal geometry defines " « gchans « " CHANs, but data
          has « maxtr « " CHANs.";
          stErrFatal  (mess);
    )
          float dxs,   dys, Grouplnt. YOs;
          exParGetFloat    ("DXS", &dxs);
          exParGetFloat    ("DYS", &dys);
          exParGetFloat    ("GROUPINT", &GroupInt);
          exParGetFloat    ("YOS", &YOs);

    //    Set the nominal receiver spread
          int iErr = ShotGeom :: setReceiverSpread (YOs, nStreamers,
          chans_per_streamer, dxs, dys, GroupInt);
          if (iErr)   stErrFatal  ("Couldn't generate nominal
          spread.");

    //    Don't need negative group intervals anymore
          GroupInt = abs(GroupInt);

    //    Set the max number of traces per streamer for distance search
          maxSearch = nadig * int(ShotInt/GroupInt + 0.5) +
          int(dYs/Grouplnt + 0.5); cout « "Max traces per streamer to
          check = " « 2*maxSearch « endl;
    )
    //    Get header indices

    //    SOURCE and CHAN are always needed
          ihSOURCE = stHdrIndex("SOURCE");
          if  (!ihSOURCE) stErrFatal ("SOURCE isn't in the header !");
          ihCHAN = stHdrIndex("CHAN");
          if  (!ihCHAN)  stErrFatal ("CHAN isn't in the header !");
```

```
//    Coordinates are needed if we're using header geometry
      if  (headerGeom)
      ihSOU_X = stHdrIndex("SOU_X");
      if (!ihSOU_X) stErrFatal ("SOU_X isn't in the header !");
      ihSOU_Y = stHdrIndex("SOUY");
      if (!ihSOUY)  stErrFatal ("SOU_Y isn't in the header !");
      ihREC_X = stHdrIndex("REC_X");
      if (!ihREC_X) stErrFatal ("REC_X isn't in the header !");
      ihREC_Y = stHdrIndex("REC_Y");
      if (!ihREC_Y) stErrFatal ("REC_Y isn't in the header !");
)
//    Set up to generate a fold QC array

      exParGetstring ("AFILE", &cbuf);
      cstr = Acstring(cbuf);
      foldFile = (toupper(cstr)  != "NONE");

      if  (foldFile)
      fold_strm.open (cbuf,  ios:: out);
      numcdp = globalceom->numcdp;   inccdp = globalGeom->inccdp;
      mincdp = globaloeom->mincdp;   maxcdp = globalGeom->maxcdp;
      mincdp~r = maxcdp;    maxcdp~r = mincdp;
      ihCDP = stHdrIndex("CDP");
      if (!ihCDP)   stErrFatal  ("CDP isn't in the header ");

//    Create a zero-filled fold array
      fold_qc = AGFloat2   (nwtrc, numcdp);
      AGFloatl zero(nwtrc);    zero.fill(0);
      fold_qc.fill(zero);


//    End the Promax initialization phase
      stEndInitialization();

//    Destructor - delete any remaining ShotGathers

MegaEdt :: -MegaEdt ()
(
      for (int i=0;  i<Shots.n(); i++)
      delete Shots[i];

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

//    Generate the time-dependent threshold function

AGFloatl MegaEdt :: tThresh (const AGFloatl& taPairs)
{
AGFloatl retval(nwtrc);

//    Return a null result if there's an extra value in taPairs
      if (tapairs.n()%2)
      cerr « "odd number of values in time-amp pairs: « taPairs.n()
      « endl;
      return retval;
)
```

```
int nPairs     tapairs.n()/2;
int iPair = 0;
float tO = (lwin-l)    * gr->samprat;
float tstep = incwin * gr->samprat;
float tW;

cout « "\MegaEdt Threshold Function:" « endl;

//    Generate a threshold for each amplitude window
      for (int iw=O; iw<nwtrc; iw++)
      tw = tO + iw*tstep;

//    Extrapolate a constant threshold if beyond either end time
      if (tw <= tapairs [O])
      retval[iw]   = taPairsfl];
      else if (tw > = taPairs(2*(nPairs-l)])
      retval[iw] = taPairs(2*nPairs--~l);

//    Interpolate between bracketing pairs
      else

//    Move to the next pair if necesarv
      if (tw >= taPairs[2*ipair))
      iPair++;

//    Skip closely spaced times if not aleady at the end of taPairs
      while ((iPair<nPairs-l) && (tw<taPairs(2*ipair]))  iPair++;

//    Interpolate
      int it = 2*iPair;
      float t2 = taPairs[itl, a2 = taPairs(it+l];
      float tl = taPairsfit-2], al = taPairs(it-l];
      retval(iw] = al + (tw-tl) * (a2-al)/(t2-tl);
      cout « "Time: « tw « " window no.:  " « iw « ". Threshold ="
      « retval[iw]   « endl;
)
      out « endl;
      return retVal;

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

//         Edit a shot gather

int MegaEdt  :: ProcessEns  (int nStored, float** traces, int**
iheads, float** rheads)
(

//    Set up string stream for error message formatting
      const int msgLen=81;    char mess(msgLen];
      ostrstream msg (mess, msgLen)

//    New shot gather expected ...

      if (iRead)
```

```
    //    Construct a ShotGather if the input buffer isn't empty
          if (nStored)
          int iSource = iheads(O)[ihSOURCE];

    //    If this is the first shot, set first SOURCE and flag BOL
          int term = 0;
          if (!Shots.n())   (startSrc = iSource;     term = 1;)

    //    Extract or compute geometry
          ShotGeom sgm;
          if (headerGeom)
          sgm = ShotGeom (iSource, nStored, rheads, ihSOU~X, ihSOcl_Y,
          ihREC_X,  ihREC_Y);
          else
          sgm = ShotGeom (iSource, startSrc, ShotInt, startArray,
          nArrays, XArray, nStored);

    //    Compute S-R space search distance when you have 2 shots
          if ((searchDist == 0) &&  (Shots.n() == 2))

          float dsx = ghots[0) ->SX() -Shots(11 ->SX();
          float dsy = Shots[0]->SY()-Shots(11->SY();
          searchDist = nadig * sqrt     (dsx*dsx + dsy*dsy);

    //    Add an extra group interval to searchDist so you get some
          traces from the first and last shots in the Shots array
          float drx = Shots[0]->RX(1)    - Shots(0]->RX(2);
          float dry = Shots[0]->RY(1)    - Shots(0]->RY(2);
          float group~int = sqrt     (drx*drx + dry*dry);
          searchDist += group_int;

    //    Construct a shot gather and add it to the Shots array
          Shots.append (new ShotGather (sgm, nsmp, nStored, traces, nth,
          iheads, iwin, nwtrc, incwin, term));

    //    Empty input buffer - must have reached EOL
          else
          Shots(Shots.n()-1)->setTerminus(1);
          moreShots = 0;

    //    Decide whether we can output a gather
          nextout = findnextOut ();
          iWrite = (nextOut >= 0);

    //    Edit and output a shot gather
          if  (iwrite)

    //    Edit the next shot gather
          int iError = shotEdit    (nextOut);

    //    Copy the gather for output
          if  (!iError)
          const float *edited_trace;
```

```
        const int *edited_hdr;
        const int ntrc = Shots(nextout)->nTraces();
        for (int itrc=0; itrc<ntrc; itrc++)
        edited_trace = Shots (nextout] ->trcptr(itrc+1);
        for (int is=0; is<nsmp; is++)   traces(itrc](isl =
        edited_trace(is]; edited_hdr = Shots(nextOut]-
        >hdrptr(itrc+1); for (int ih=0; ih<nth; ih++)
        iheads(itrc](ih] = edited_hdr[ih];

    //  Don't need the trace samples in the gather that was just
        edited
        Shots (nextOut) ->deleteTraces ();

    //  Bomb if there was a problem editing
        else
        msg « `Error editing shot * « Shots(nextOut]->Source();
        stErrFatal (mess);

    //  Get rid of any unneeded shot gathers
        deleteGathers ();

    //  Decide if we need a new shot
        nextOut = findNextOut ();

    //  Try to read a new shot if we need one
        iRead = (nextOut < 0) && moreShots;
        return 0;

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

    //  Find the next shot gather to be edited

        int MegaEdt    :: findnextOut ()

    //  Not enough gathers present - go back and get some more
        int nShots = Shots.n();
        if (nShots<=1)  return -1;

    //  Find the first gather that has trace samples in it
        int iShot  = 0, lShot  nShots - 1; while ((iShot<nShots) &&
        (!Shots(iShot] ->tracesPresent())) iShot++;

    //  No gathers with samples - nothing to edit
        if (iShot==nShots)   return -1;

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

    //  Found the first unedited gather -
        Decide if the required range of gathers is present
        int iSource = Shots[ishot]->Source();

    //  Search forward (trailing shots)
        int fShot = ishot;
        int fDiff = 0,    EOL = 0;
        while  (fDiff < nadjg)
```

```
    //    Quit when you reach the last shot in the Shots array
          if  (fShot == iShot)

    //    But first check to see if youre at the end of the line
          EOL = Shots(fshot]->Terminus();
          break;
    )
          fShot++;     fDiff = abs (iSource - Shots(fShot]->Source());

    //    Not enough trailing shots    ...
          if  ((fDiff < nadjg)  && !EOL) return -1;

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

    //    Search backward (preceeding shots)
          int bShot = iShot;
          int bDiff = 0,     BOL = 0;
          while  (bDiff < nadjg)

    //    Quit when you reach the first shot
          if  (bShot == 0)

    //    But first check to see if you're at the start of the line
          BOL = Shots[bshot]->Terminus();
          break;
    )
          bShot--;     bDiff = abs (iSource - Shots(bShot]->Source());

    //    Not enough preceeding shots    -
          if  ((bDiff < nadjg)  && !BOL) return -1;

          return iShot;

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

          int MegaEdt ::  shotEdit  (int iEdit)
    (
          int ntrc = Shots(iEdit]->nTraces();

    //    Number of traces to be edited

          AmpTable ampTbl(nwtrc);
          KillFlags kFlags(nwtrc);
          int all_live = 1;

    //    Becomes 0 when something's killed
          int line_count;

    //    Edit each trace in shot at iEdit
          for (int itrc=1; itrc<=ntrc; itrc++)

    //    Skip this trace if it's dead
          if (Shots(iEdit]->hdrptr(itrc) [STDHDR(itrc_type)] == IDEAD)
          continue;
```

```
    //    Look for traces inside the search distance
          int nfound = findTraces   (iEdit, itrc, ampTbl);
          edit the next trace if none are found
          if  (infound) continue;

    //    Shrink ampTbl to maxcheck traces if nonzero
          ampTbl.Shrink(maxcheck);

    //    Perform trace blanking on itrc
          kFlags = Shots[iEdit]->blank (itrc, Thresh, ampTbl, Taper);

    //    Update the fold array if we're keeping track
          if  (foldFile)
          int cdp = Shots(iEdit]   >hdrptr(itrc) (ihCDP);
          if  (cdp < mincdp_r)  mincdp_r = cdp;
          if  (cdp > maxcdp_r)  maxcdp_r = cdp;
          int icdp = (cdp - mincdp)   I inccdp + 1;
          for (int iw=0; iw<nwtrc;   iw++)
          if (kFlags(iw)   fold_qc(iw,icdp) += 1.0;

    //    Make an entry in the Census file if any windows were killed
          if  (censusFile && kFlags.nk())
          Census « line « Shots(iEditl->Source()
          « (Shots[iEdit]~>hdrptr(itrc))[ihCHAN] « kFlags « endl;

    //    Write the SOURCE line if this is the first kill
          if (all_live)
          Census « "SOURCE"  « Shots[iEdit]->Source()  « endl;
          all_live = 0;  Census « (Shots[iEdit]->hdrptr(itrc))(ihCHAN1
          « kFlags.nk() « endl; line_count = 0; for (int ik=0;
          ik<kFlags.n();  ik++)  if (.`kFlags(ik))
          Census « setw(3)   « ik+1;  line_count++;
          if (!(line_count%26))
          Census « '\n;
          if (line_count%26)
          Census « endl;
          return 0;

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

    //    Find traces inside the search distance and update the
          AmpTable

    int
    MegaEdt  :: findTraces (int iEdit, int itrc, AmpTable& ampTbl)

          int nTrcShot;
          int nShots = Shots.n();

    //    Number of shots in the array
          const float *trcAmps=NULL;
          int nfound = 0;
          ampTbl.clear();
```

```
    //    Set the search range for each streamer if using nominal
          geometry
          int itrc_streamer,  start_streamer, end_streamer;
          if  (!headerGeom) (
          itrc_streamer = (itrc-1)  % chans_er_streamer + 1;
          start_streamer = itrc_streamer - maxSearch;
          if  (start_streamer<1) start_streamer = 1;
          end_streamer = itrc_streamer + maxsearch;
          if  (end_streamer > chans~er~streamer) end_streamer =
          chans_per_streamer;

    //    Check each shot gather
          for (int jShot=0; jShot<nShots; jShot++)
          if  (abs(jShot-iEdit) > nexcl)
          nTrcShot = Shots(jShot]->nTraces();

    //    If using header geometry, check every trace
          if  (headerGeom)
          for (int jtrc=0; jtrc<nTrcShot; jtrc++)
          trcAmps = Shots(iEdit]->nearAmps (itrc, searchDist,
          Shots[jShot],  jtrc)
          if (trcAmps.'=NULL)   nfound++;  ampTbl update (trcAmps);

    //    If using nominal geometry, check only maxSearch traces on
          each streamer --
          else   int first, last;

    //    Check each streamer
          for (int iStreamer=0;   istreamer<nStreamers; iStreamer++)
          first = iStreamer    chans_oer_streamer + start_streamer;
          last = iStreamer * chans_oer_streamer + end_streamer;
          if (last > nTrcShot)  last = nTrcShot;

    //    Check maxSearch traces cenetered on i~rc_streamer
          for (int jtrc=first;  jtrc<=last; jtrc++)

    //    Return amplitudes for any trace that's within the search
          distance
          trcAmps = Shots(iEdit]->nearAmps  (itrc, searc~ist,
          Shots [jShot],  jtrc);

    //    Update the &rnplitude table if anything was found
          if (trcAmps'.=NULL)
          nfound++;    ampTbl update (trcAmps);

    //    End of excluded shot block
          End of shot gather loop
          return nfound;

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

    //    Delete any unneded gathers; return the number deleted
          int MegaEdt ::  deleteGathers ()
          int iSource,  j,  near;
          int nDel=O,  ng=Shots.n();
```

```
//    Check all gathers in the array (but not the last one)
      for (int i=0;  i<ng-1; i++)
      iSource = Shots(i)->Source();

//    Look for an unedited gather with "source" within nadjg of
      this shot
      j = 0;  near = 0;
      while (!near && (j<ng))
      near = ( (abs(iSource-Shots[j]->Source()) <= nadjg)
      && (Shots(j)->tracesPresent();
      j ++;

//    Remove this gather from the array if no nearby shots were
      found
      if (!near)
      delete Shots(i);
      Shots.remove(i);
      nDel++;
      ng--;  i--;
      return nDel;

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

//    Write the fold file
      void
      MegaEdt ::  WriteFoldFile()
      if  (!foldFile)   return;

      int ncdp_r =    (maxcdp_r - mincdp_r) I inccdp + 1;
      fold_strm « nwtrc « ' ' « gr->samprat*incwin « ' ' « ncdp_r
      « endl;

      const int n_per_line = 25;
      const int field_width = 3;
      for (int icdp=1; icdp<=ncdp_r;  icdp++)
      fold_strm « *  CDP   « mincdp_r + inccdp*(iCdp_1) « endl;
      for (int iw=0; iw<nwtrc,  iw++)
      fold_strm « setw(field_width) « int(foldqc(iw,icdp)+0.5);
      if  (!((iw+1) % n_per_line))  fold_strm « endl;
      if  (iw % n_per_line)   fold_stfln « endl;

      #ifndef _SHOTGEOM_H
      #define _SHOTGEOM_H

      #include <stddef.h>
      #include <math.h>
      #include <iostream.h>

//    These are needed for header and error functions.

      #include "cglobal.h"
      #include "cpromax.h"
      #include "oSocketTool.h"
```

```
**********************************************************************
                    MegaEdt Shot Geometry Class

                      R.T. Houck  June 1995
**********************************************************************

class ShotGeom

public:

//    Default constructor
      ShotGeom()   : ntrcs(0), recY(NULL), recX(NULL) ()

//    Constructor, from a shot gather header
      ShotGeom (int Source,  int Ntrcs, float**
      rthdrs, int iSouX, int iSouy,
      int iRecX,  int iRecY);

//    Constructor, from "nominal" geometry parameters
      ShotGeom (int Source,  int startSrc,  float ShotInt,
      int startArray,
      int Narrays,  float dXA,  int Ntrcs);

//    Copy constructor
      ShotGeom (const ShotGeom& sg);

//    Assignment operator
      ShotGeom& operator   (const ShotGeom& sg);

//    Set receiver spread arrays (for nominal geometry)
      static int setReceiverSpread (float YOs,  int Nstreamers,
      int chans_per_streamer,  float &Xs, float dys, float
      GroupInt);

//    Return source
      int Source()  const   (return source;)

//    Return shot and receiver coordinates
      float SX()  const (return soux;)
      float SY()  const (return souy;)
      float RX (int itrc)   const;
      float RY (int itrc)   const;

//    Distance between two traces in Source-Receiver space
      float distSR (int itrc, int jtrc}  const;
      float distSR (int itrc, const ShotGeom& sg, int jtrc) const;

//    Destructor
      ShotGeom   (delete  ( recX;  delete () recY;)

      private:
      int source;           //    SOURCE from Promax header
      float souX, souY;     //    Coordinates of the shot
      int ntrcs;           //    Number of traces
```

```
        float *recX, *recY;    //     Arrays of receiver locations
        static int nspread;    //    No. of traces in nominal spread
        static float *recgprX,
           *recSprY;           //    Nominal receiver spread

        #endjf                 //    SHOTGEOM_H

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

                        MegaEdt Shot Geometry Class

                          R.T. Houck     July 1995

      1) modifed to observe array and streamer numbering conventions
                                    6/10/97

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

    #include "ShotGeom.h"

    #include <iomanip.h>

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

    //    Constructors ...
          from a shot gather header

    ShotGeom :: ShotGeom (int Source, int Ntrcs, float** rthdrs,
    int iSouX, int iSouy, int iRecX,  int iRecY) : source(Source),
    ntrcs(Ntrcs)

    //    Make sure there's a header array
          if (rthdrs == NULL) stErrFatal ("ShotGeom: Trace header array
          not specified");

    //    Source coordinates
          souX = rthdrs[0][iSouX];
          souY = rthdrs(0][iSouY];

    //    Receiver coordinates
          recX = new float [ntrcs];  recY = new float [ntrcs];
          for (int i = 0; i<ntrcs; i++)
          recX(i) = rthdrs(i] [iRecx);
          recy(i) = rthdrs(i][iRecYJ;

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

    //    Constructor from a set of    nominal" geometry parameters

          ShotGeom :: ShotGeom (int Source, int startSrc,float ShotInt,
          int startArray,  int Narrays,  float dXA, int Ntrcs)
          source(Source) , ntrcs(Ntrcs)
```

```
//    Make sure receiver spread arrays have been loaded
      if ((recSprx==NULL) !! (recSpry==NULL)
      cerr « "ShotGeom: Can't load geometry until nominal spread is
      generated"
      « endl;
      abort ();

 //   And traces are consistent
      if (ntrcs 1= nspread) (
      cerr « "ShotGeom:  Input no. of traces  (" « ntrcs
      « ")  is inconsistent with size of nominal spread (" « nSpread
      « *)` « endl;
      abort ();

//    Source coordinates
      float XA1 = O.S * (Narrays-1)  * dXA;
      X-coord of Array 1
      int iArray = startArray +
      (source - startSrc) % Narrays;
      Array # of source
      souX = XA1 - dXA * (iArray-1);
      souY   ShotInt * (source - startSrc);

 //   Receiver coordinates (from receiver spread arrays)
      recX = new float (ntrcs];   recY = new float [ntrcs];
      for (int i = 0; i<ntrcs;  i++)
      recx(i) = recSprx(i);
      recy(i) = souY + recSpryfil;

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

//    Set nominal receiver spread geometry
      int ShotGeom    setReceiverSpread (float YOs, int
      Nstreamers, int chans_per_streamer, float dxs, float dYs,
      float GroupInt)

//    Allocate spread arrays; bomb if they already exist
      if ((recsprx == NULL) && (recSprx == NULL))
      nspread = Nstreamers * chans_per_streamer;
      recSprx = new float (nspread];
      recspry = new float (nspread];)
      else
      cerr « ShotGeom:  Attempted to reset receiver spread
      « endl;
      return 1;

//    Set the inline offset of lowest Chan on center streamer
      float yStart;
      int reverse    (GroupInt<0);
      if  (reverse)
      yStart = -YOs + GroupInt*(chans_per_streamer-1);}
      else
      yStart   -YOs;
```

```
    //    Generate the center streamer coords, if there is one
          int iCenter = Nstreamers/2;
    //    Center streamer no. if  (Nstreamers%2) int startChan =
          chans_per_streamer * icenter;

    //    First chan in center int endchan = startchan +
          chans_per_streamer - 1; for (int ichan=startchan;
          iChan<=endChan; ichan++)   recSprxf ichan] = 0.;
          recspry(iChan] = yStart - GroupInt*(iChan-startChan);

    //    Generate off-center streamer coords
          int noff = Nstreamers/2;   int hiStr,  lostr, hichanO,
          lochanO, hichan, lochan; float yO, hiX, loX;

    //    X-coord of first streamer; first streamer is rightmost
          if dxs is positive
          float xO = 0.5 *  (Nstreamers-1)*dxs;

    //    Loop over matching high-low pairs of off-center streamers
          for (int iaff=noff;   ioff>O; ioff--)

    //    High streamer no.,   crossline coord, first chan
          hiStr = Nstreamers - joff;
          hiX = x0 - histr    dxs;
          hichanO = histr*chans_per_streamer;

    //    Low streamer no.,  crossline coord, first chan
          loStr = ioff-1;
          loX = xO - loStr * dxs;
          loChanO = loStr*chans_per_streamer;

    //    Inline coord of heads at streamer pair
          yO = yStart - (iCenter-iOff) * dYs;

    //    Generate receiver coords for the streamer pair
          for (int iChan=O; iChan<chans_per_streamer;  iChan++)
          hiChan   hichanO + iChan; recsprx(hichan] = hiX;
          recspry(hichan] yO - GroupInt*ichan; lochan = loChanO +
          ichan; recsprx[lochan] = loX; recspry(lochan] = yO -
          GroupInt*iChan;

          cout « *\nNominal receiver spread from
          ShotGeom::setReceiverSpread\n" « endl;
          for (int i=O;   i<Nstreamers; i++)
          cout « *Trace    Xr     Yr    *;
          cout « endl;
          cout.setf (ios::fixed);
          cout.precision (1);
          int itrcO,  itrc;
          for (int iChan=O;   iChan<chans_per_streamer;
          ichan++) for (int iStr=O; iStr<Nstreamers; iStr++)
          itrc = iStr*chans_per_streamer + iChan;
          cout « setw(5) « itrc+1
          « setw(7)  « recsprXfitrc]  « setw(8) « recSpry[itrc];
```

```
      cout « endl;
      return 0;
*************************************************************************

//   Copy Constructor

      ShotGeom :  ShotGeom (const ShotGeom& sgm)
      source(sgrn.source), ntrcs (sgrn.ntrcs), soux(sgrn.soux),
      souY(sgm.souy)

      recX = new float    [ntrcs];  recY = new float  (ntrcs];
      for (int i = 0;  i<ntrcs;   i++)
      recxEi]    sgm.recX(i];
      recy(iJ  = sgrn.recy(i];

//    Assignment operator

      ShotGeom& ShotGeom  :: operator  (const ShotGeom& sgm)

//    Check for self assignment
      if  (this == &sgm)  return *this;

//    Get rid of the current contents of this ShotGeom
      delete  El recX;  delete  El recY;

//    Assign data members
      source = sgm.source;
      ntrcs = sgm.ntrcs;
      souX = sgm.souX;
      souY = sgm.souY;

//    Allocate and copy receiver coordinates
      recX = new float    Entrcs);   recY = new float (ntrcsl;
      for  (int i=0;    i<ntrcs;  i++)
      recx[i]  = sgm.recx(i];
      recY[i)  = sgm.recY(i);

      return *this;

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

//    Receiver Coordinate Functions

      float ShotGeom :: RX (int itrc)     const
      if (itrc<1   II itrc>ntrcs)
      cerr « "ShotGeorn:  no trace     « itrc « endi;
      stErrFatal  (*Aborting…*);
      return recY[itrc-l);

      float ShotGeom :: RY  (int itrc)    const
      if (itrc<l
      itrc>ntrcs)
      cerr « ~ShotGeom: no trace * « itrc « endl;
      stErrFatal  (*Aborting...*);
      return recy(itrc-l];
```

```
//    Distance between two traces in Source-Receiver space
      float ShotGeom :: distSR (int itrc, int jtrc)  const
      int iO=itrc-l,   jO=jtrc-l;
      float xdiff-recx(iOJ -recxfjO), ydiff--recY(iO] -recy(jOl;
      return sqrt  (Xdiff*Xdiff + ydiff*ydiff);

      float ShotGeom :: distSR (int itrc,  const ShotGeom& sg, int
      jtrc)  const
      int iO=itrc-l,   jO=jtrc-l;
      float dxr=recx[iO)-sg.recx[jOJ,    dyr=recy[iOJ-sg.recy[jO];
      float dxs=soux-sg.souX,  dYys=souY-sg.souY;
      return sqrt  (dxr*dXr + dyr*dyr + dXs*dXs + dYs*dYs);

#ifndef _SHOTGATHER_H
#define _SHOTGATHER_H

#include <stddef.h>
#include <math.h>

#include <agc/Floatl.h>

#include "ShotGeom.h"
#include "KillFlags.h"
#include *AmpTable.h"

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

   MegaEdt Shot Gather class, including an amplitude gather and
                              geometry

                      R.T. Houck  November 1995

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

class ShotGather

public:

//    Constructor, from a shot gather and Geometry
      ShotGather (ShotGeom& sgm, int Nsmp,  int nTraces,
      float** traces, int Nth,  int** iheads, int Npw,
      int Nwtrc, int incw, int term=O)

//    Default constructor
      ShotGather() : ntrcs(O), traces(NULL), amps(NULL) ()

//    Copy constructor
      ShotGather (const ShotGather& sg);

//    Assignment operator
      ShotGather& operator= (const ShotGather& sg);
```

```
    //    Return SOURCE value, source X, Y coordinates
          int Source() const (return source;)
          float SX() const  (return sgm.SX();)
          float SY() const  (return sgm.SY();)

    //    Return the number of traces
          int nTraces() const   (return ntrcs;)

    //    Return reciever coordinates for trace itrc
          float RX (int itrc)   (return sgm.RX(itrc);
          float RY (int itrc)   (return sgm.RY(itrc);

    //    Set and return the line terminus flag
          void setTerminus (int term)   (terminus-term;)
          int Terminuso   const  (return terminus;)

    //    Return a pointer to the trace amplitude array if trace
          jtrc in sg is within Search of trace itrc;
          otherwise, return NULL const float' nearAmps
          (int itrc, float Search, const ShotGather *sgp,
          int jtrc)  const;

    //    Blank trace samples based on thresholded amplitude comparison
          with ampTbl KijiFlags blank (int itrc, const AGFloatl&
          thresh, const AmpTable& ampTbl,const AGFloatl& taper);

    //    Return pointers to headers and samples for trace itrc
          int hdrptr() (return iheads;)
          float  trcPtr() (return traces;)
          const int* hdrptr(int itrc) const (return iheads(itrc-l];)
          const float* trcptr(int itrc) const (return traces(itrc-l];)

    //    Return true if trace samples are present
          int tracesPresent() const  (return (traces != NULL);)

    //    Delete the trace samples
          void deleteTraces ();

    //    Destructor
          ShotGather();

          private:
          int source;    SOURCE from Promax header
          int terminus; TRUE if this shot is the start or end of line
          int ntrcs;     Number of traces
          ShotGeom sgm; Coordinates of the shot and receivers
          int nth;       No. of header entries
          int**iheads;  Trace headers
          int nsmp;       Samples per trace
          float**traces 2-D array of trace samples
          int npw, incWin; No. of samples per amp window, window inc
          int nwtrc;        No. of amp windows per trace
          float** amps;     2-D array of trace amplitudes

    #endif
```

```
//    SHOTGATHER_H
**********************************************************************

                        MeqaEdt Shot Gather Class

                        R.T. Houck  November 199S

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

#include "ShotGather.h

//    Constructors
      from a shot gather and Geometry
      ShotGather :: ShotGather (ShotGeom& Sgm, int Nsmp, int
      nTraces,  float** Traces, int Nth, int** iHeads, int Npw,
      int Nwtrc,  int incw, int term)

      source(Sgm.Source()), terminus(term), ntrcs(nTraces),
      sgm(Sgm), nth(Nth), nsmp(Nsmp), npw(Npw); incWin (incw),
      nwtrc(Nwtrc)

//    Save traces and headers
      traces = new float*    (ntrcs];  iheads = new int* (ntrcs];
      for (int i=O; i<ntrcs; i++) traces(i]  = new float fnsmp];
      for (int is=O; is<nsmp; is++)  traces(ilEisl = Tracesfi)(isl;
      iheads(i] = new int (nth]; for (int ihO; ih<nth; ih++)
      iheads[i](ih] = iHeads(i][ih];

//    Compute amplitudes
      int istw,   jend;
      double ssq,  scal=1./float(npw);
      amps = new float* (ntrcs];
      for (i=O;  i<ntrcs; i++)
      amps[i]  = new float  [nwtrc];
      for (int iw=O; iw<nwtrc;    iw++)
      jstw = iw * incWin; jend  istw + npw - 1;  ssq = 0;
      for (int is=istw;  is<=iend; is++)  ssq += traces(i] [is] *
      traces[i] [is]; amps fi] [iw] = sqrt  (55q*scal)

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

//    Copy Constructor

      ShotGather :: ShotGather (const ShotGather& sg)
      source(sg.source),   ~erminus(sg.terminus).  ntrcs
      (sg.ntrcs), sgm(sg.sgm),
      nth(sg.nth), nsmp(sg.nsmp),   npw(sg.npw),
      jncWin(sg.incWin), nwtrc(sg.nwtrc)

//    Find out if there are samples to copy
      int samps = (sg.traces    != NULL);
```

```
    //    Allocate and copy trace, header, and amplitude arrays
          if (samps) traces = new float*  [ntrcsl;  iheads = new lnt*
          (ntrcs];)
          else
          traces = NULL;
          iheads = NULL;

          amps = new float* [ntrcs];

          for (int i=0; i<ntrcs; i++)

          if (samps)
          tracesfi)   = new float (nsmp];
          for (int is=0;    is<nsrnp; is++)  traces(il [is] =
          sg.traces[i] [is];

          iheads(i)  = new int   [nth];
          for (int ih=0;    ih<nth; ih++)   iheads(i)[ih]  =
          sg.iheadsfi](ih];

          amps[i]   = new float  [nwtrc];
          for (int iw=0; iw<nwtrc;    iw++)   amps(i)[iw] =
          sg.amps(i)[iw];

    //    Assignment operator

          ShotGather& ShotGather operator= (const ShotGather& sg)

    //    Self assignment ...
          if  (this == &sg)   return *this;

    //    Delete current memory      for (int i=0;  i<ntrcs;  i++)
          if  (traces i= NULL)  (delete [] traces(I); delete []
          iheads(i];  delete  (1 amps[i]; if  (traces  ! = NULL)
          delete  []  traces; delete []  iheads; delete  [] amps;

    //    Assign data members
          source =  sg.source;  terminus = sg.terminus;
          ntrcs = sg.ntrcs;   sgm = sg.sgm; nth = sg.nth;  nsmp
          sg.nsmp; npw = sg.npw;  incWin = sg.incWin; nwtrc = sg.nwtrc;

    //    Allocate and copy arrays
          if  (sg.traces NULL) traces = new float*   (ntrcsl;  iheads =
          new lnt* [ntrcs];
          else
          traces  NULL;  iheads  NULL;

          amps = new float*  (ntrcs]; for  (i=0;  i<ntrcs; i++) if
          (traces   NULL) traces(i)  = new float  (nsmpj;  for (int
          is=0; is<nsmp; is++) traces(i) (is) = sg.traces(i] (is);
          iheads(i]  = new int [nth]; for (int ih=0;   ih<nth; ih++)
          iheads[i] (ihi = sg.iheads(i) (ihi  amps[i] = new float
          (nwtrcl; for (int iw=0; iw<nwtrc;   iw++)  amps(i][iw] =
          sg.amps(i](iw];
          return *this;
```

```
//    Return an array of trace amplitudes if trace jtrc in sg is
      within Search of trace itrc; otherwise, return an empty array

const float*
ShotGather :: nearAmps (int itrc, float Search, const ShotGather
*sgp,  int jtrc)  const


if (sgm.distSR (itrc,  sgp->sgrn, jtrc) < Search)  return sgp-
>amps(jtrc-1];

return NULL;


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

//    Blank trace amplitudes based on thresholded amplitude
      comparison with ampTbl
      KillFlags ShotGather blank (int itrc,  const AGFloat1&
      thresh, const AmpTable& &rnpTbl, const AGFloat1& taper)

//    Don't kill anything if the amplitude table is empty
      KillFlags iwLive(nwtrc);  if  (iampTbl.ncol()) return iwLive;
      int fKill,  lKill,  live; int icent=(npw-incwin)/2;

//    Offset to center incWin samples

//    Identify windows to be blanked
      AGFloat1 med = ampTbl.median();
      float* medPtr = (float*)med;
      int itro_1 = itrc - 1;
      for (int iw=0; iw<nwtrc;      iw++)
      if (amps[itrc~1) (iw]  > thresh(iw]*med[iw])
      iwLive.kill(iw);
      int nKilled = iwLivenko;

//    Kill the trace if all windows are dead
      if (nKilled == nwtrc)
      iheads (itrc_1] [gTDHDR(itrc_type)]   = IDEAD;
      for (int i=0; i<nsmp;    i++)    traces(itrc_1) (i) = 0;

//    Kill the flagged windows
      else
      int lTaper=taper.n();
      for (iw=0; iw<nwtrc;   iw++)

//    Don't do anything unless this window has been killed
      if (iwLive(iw])    continue;

//    Kill incWin samples about the window center
      fKill = iw*incwin + iCent;
      lKill = fKill + incWin - 1;
```

```
//    Kill to start or end of trace if this is the first or last
      window
      if (iw == 0)    fKill = 0;
      if (iw == nwtrc-1)     lKill = nsrnp;

//    Kill the specified range of samples
      for (int is=fKill;  is<=lKill;  is++) traces(itrc~l] (is] =0;

//    Taper down to the killed window if the previous window was
      alive
      if (iw>0 &&  iwLive(iw-l])
      int last = MAX (0,      fKill-lTaper)
      for (is=fKill-l;    is>=last;  is--)
      traces(itrc_l] (is] = taper[fKill-l-is] * traces[itrc_1] [is];

//    Taper up from the killed window if the next window is alive
      if (iw<nwtrc-l && iwLive(iw+l])
      int last = MIN (nsmp,      lKill+lTaper-l)
      for (is=lKill+l;    is<=last;  is++)
      traces[itrc_l] [is] = taper[is-lKill-l] * traces(itrc_1] [is];

      return iwLive;

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

//    Delete the trace samples and headers
      void ShotGather   deleteTraceso
      if (traces    `.= NULL)
      for (int i-O; i<ntrcs;    i++)
      delete (]  traces(iJ;  delete [1 iheads(iJ;)
      delete  El traces;  delete [1  iheads;
      traces = NULL;    iheads = NULL;

//    Destructor

      ShotGather ::    ShotGather()
      deleteTraces ();
      if (amps      NULL) (
      for (int i=O; i<ntrcs;    i++)    delete El amps(i];
      delete   (1 amps;
      amps = NULL;

#ifndef    AMPTABLE_H
#define    AMPTABLE_H

//    Manual instantiation of float* array template
      #if defined (RS6OOO)
      #include <agc/Arr. C>
      #pragma define (AGArr<const float*>)
      #endif
      #include <agC/Floatl.h>
```

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

                    MegaEdt Amplitude Table Class
        Uses an array of float*'s instead of an AGFloat2 object
                        R.T. Houck   May 1997

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

class AmpTable

public:

//    Constructor (traces added by updating)
      AmpTable (int N)   : nwtrc(N)   ()

//    Append a column (trace) to the end of the table
      void update (const float *col);

//    Shrink the table to max pointers
      void Shrink (int max);

//    Clear the array of amp ptrs
      void clear()    ( m_Amps.clear();

//    Return the number of columns in the table (traces pointed to)
      int ncol()  const   ( return m_Amps.n();

//    Return an array of median amplitude for each time
      AGFloat1 mediano const;

//    Return an array of average amplitude for each time, excluding
      the nExcl
      highest values

//    AGFloat1 avg (int nExcl=0) const;

private:

    AGArr<const float*> m_Amps;
    int nwtrc;

     #endif

     _AMPTABLE_H

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

                    MegaEdt Amplitude Table Class
                (just an AGFloat2 with row-wise sorting)

                        R.T. Houck    July 1995

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

```
//    modified to do an explicit shell sort on a float array
      instead of using Arra<float>::sort     6/4/97

      #include <iostream.h>
      #include *AmpTable.h

//    Append a trace pointer to the table
      void AmpTable  update (const float *col)
      m_Amps append (col);


//    Shrink the AmpTable to contain max pointers uniformly
      selected from the existing set
      void AmpTable  Shrink (int max)
      if  (max<=0 I  max>--m_Amps.n())
      return;
      float step = float(m~Amps.n())    / float(max);
      for (int indx=0; indx<max; indx++)
      m_Amps(indx]   m_Amps(indx*step];
      m_Amps .setLength(max);

//    Find the median at each row (time window)
      void ssort (int n,  float* a);
      AGFloatl AmpTable :: median()  const

      int nrows = nwtrc;
      int ncols = m_Amps.n();
      int nLive, middle;
      float temp;

//    AGFloatl medians(nrows), arow(ncols);

      AGFloatl medians (nrows);
      float+ arow = new float [ncols];

//    Loop over table rows ...
      for (int ir=0; ir<nrows; ir++)

//    Sort the row, discarding zeroes
      nLive = 0;//     arow.clear();
      for (int ic=0; ic<ncols; ic++)
      temp = m_Amps[ic] [ir];
      if (temp) arow(nLive] = temp;
      nLive++; ssort (nLive, arow);//  arow.sort();

//    Take the center value or the avg of the two center values
      nLive = arow.n();
      middle = nLive / 2;
      if (nLive%2)
      medians[ir] = arow[middle);
      else
      medians(ir)   0.5 +  (arow[middle] + arow[middle-1]);

      delete [] arow;
      return medians;
```

```
//    Sort an array of floats into increasing order
      void ssort (int n,  float* a)
      int i,  j;
      float temp;
      for (i=1;  i<n;  i++)
      temp =
      for (j=i-1;  j>=0;  j--)
      if  (temp <= a[j])   break;
      a(j+1)   =

      if  ((j==0) &&  (temp>a(0)))
      a(0]  = temp;
      elSe
      a(j+1)   = temp;

#ifndef  _KILLFLAGS_H
#define  _KILLFLAGS_H

#include <agc/Intl.h>
#include <iostream.h>

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

                MegaEdt class for window blanking flags
                        R.T. Houck    July 1995

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

//    1) operator « added 12/13/95   RTH

class KillFlags : public AGIntl

public:

//    Constructor (n windows, all live)
      KillFlags  (int n)  : AGIntl (1, 0, n), nKilled(0) ()

//    Set all windows to *live
      void setLive()   (fill(1); nKilled=0;)

//    Kill window iw
      void kill  (int iw)  ((*this)(iw]=0; nKilled++;)

//    Return the number of killed windows
      int nk()    (return nKilled;)

//    Return a list of window numbers of killed windows
      AGIntl iKilled() const;

private:
      int nKilled;                // number of windows killed
      static const int nLine;   // number of chars per output line
```

```
    friend ostream& operator« (ostream& Os, KillFlags& kf);

#endif          //  KILLFLAGS_H

#include *KillFlags.h"

//    Return a list of killed window numbers
      AGIntl
      KillFlags :: iKilledo    const
      AGIntl kill_list(nKilled);
      int ik    0;
      for (int i=0; i<n(); i++)
      if ((*this) [ii)   kill_list(ik++)   i+l;
      return kill_list;


//    Ouptut a KillFlags as a string on unseparated 0's and 1's on
      nLine-char lines

      ostream& operator« (ostream& Os, KillFlags& kf)

      int last;

//    Print parameters
      Os « `\n5 « kf.n()-kf.nKilled « S   « kf.nKilled;

//    Print each line of the flags string
      for (int i=0; i<kf.n(); i+=kf.nLine)
     (i+kf.nLine > kf.n()) ? last-kf.n()-l last-i+kf.nLine-l;
     Os «  for (int ii=i; ii<=last; ii++)     Os « kf(ii);

      Os « endl;
      return Os;

      #ifndef _FORTRAN_H
      #define _FORTRAN_H

//    Fortran functions

      #undef MAX

      template<class T>
      inline T& MAX (T& a, T& b)
      (return a > b ? a : b;)

      *undef MIN

      template<class T>
      inline T& MIN (T& a, T& b)
      (return a < b ? a  b;)

      #undef NINT

      template<class T>
      inline int NTNT (const T& a)
```

```
(return a > 0 ? int(a+0.5) int(a-0.5);)
#endif
_FORTRAN_H
```

CLAIMS:

1. A method for analyzing a target seismic trace, comprising;

setting a distance value indicative of the number of shot gathers to be considered in conjunction with analyzing a target trace;

specifying which traces from a plurality of shot records to compare to a target trace based on the distance value;

calculating a first amplitude value associated with the amplitude of the target trace;

calculating a second amplitude value associated with the amplitudes of the specified comparison traces;

comparing the first amplitude value to the second amplitude value.

2. The method of claim 1 wherein the first amplitude value comprises calculating an absolute amplitude measure.

3. The method of claim 1 wherein the first amplitude value comprises calculating an RMS value of data points in the target trace.

4. The method of claim 1 wherein the calculating steps are performed on a portion of the data points in the target and comparison traces.

5. The method of claim 1 further including dividing the target traces into at least two time windows and calculating an RMS value for the data points in each window.

6. The method of claim 5 further including dividing the comparison traces into at least two time windows for each comparison trace and calculating an RMS value for the data points in each comparison trace time window.

7. The method of claim 6 further including calculating a median value for at least three of the comparison trace RMS values associated with a given time window.



8. The method of claim 7 wherein the comparing step comprises comparing an RMS value from the target trace to the median value.

9. The method of claim 6 further including calculating a median value for all of the comparison trace RMS values associated with a given time window.

10. The method of claim 9 wherein the comparing step comprises comparing an RMS value from the target trace to the median value.

11. A method for selecting seismic traces for analyzing the noise content of a target trace comprising:

selecting traces from shot records located within a predetermined distance of the shot record containing the target trace;

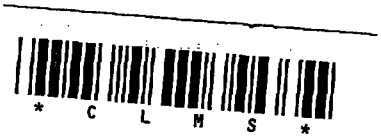
omitting during the selecting step traces from the shot record that contains the target trace;

comparing the relative amplitude of the selected traces to the target trace.

12. The method of claim 11 wherein the selecting step also includes, for a given selected shot record selecting traces from that shot record that are located within another predetermined distance from the target trace.

13. The method of claim 12 wherein comparing step includes calculating and comparing an RMS average associated with the target trace to a median of RMS averages associated with the selected traces.

14. The method of claim 11 wherein the predetermined distance defines a circle in shot-receiver space for a 2-D seismic survey.



15. The method of claim 11 wherein the predetermined distance defines a hypersphere in four dimensional shot-receiver space for a 3-D seismic survey.

16. A computer readable storage medium for storing an executable set of software instructions which, when inserted into a host computer system, is capable of controlling the operation of the host computer, the software instructions being operable to process traces from a target shot record, comprising:

a means for selecting traces from at least one nearby shot record;

a means for comparing the amplitude of target traces in the target shot record with the amplitudes of traces from the nearby shot records.

17. The invention of claim 16 wherein the means for selecting traces includes omitting traces from the target shot record.

18. The invention of claim 17 wherein the means for selecting traces includes selecting traces within a predetermined distance in the source-receiver domain from the target trace.

19. A seismic processing system, comprising:

a computer; and

a tape drive coupled to the computer;

wherein the computer retrieves seismic traces data from the tape drive and selects traces from shot records that do not contain a target trace to be analyzed for the presence of noise.

20. The seismic processing system of claim 19 wherein the computer selects traces from shot records that are located within a predetermined distance, measured in source-receiver space, from the shot record that contains the target trace.



21. The seismic processing system of claim 20 wherein the computer further compares an RMS average associated with the target trace with a median of RMS averages associated with the selected traces.

22. The seismic processing system of claim 20 wherein the computer further calculates an RMS average for portions of the target trace, calculates RMS averages for comparable portions of the selected traces, calculates median values of the selected traces RMS averages and compares the median values to the target trace RMS averages.

23. The seismic processing system of claim 22 wherein the computer alters a portion of the target trace if the RMS average for that portion is substantially greater than a median value of the RMS averages of comparable portions of the selected traces.



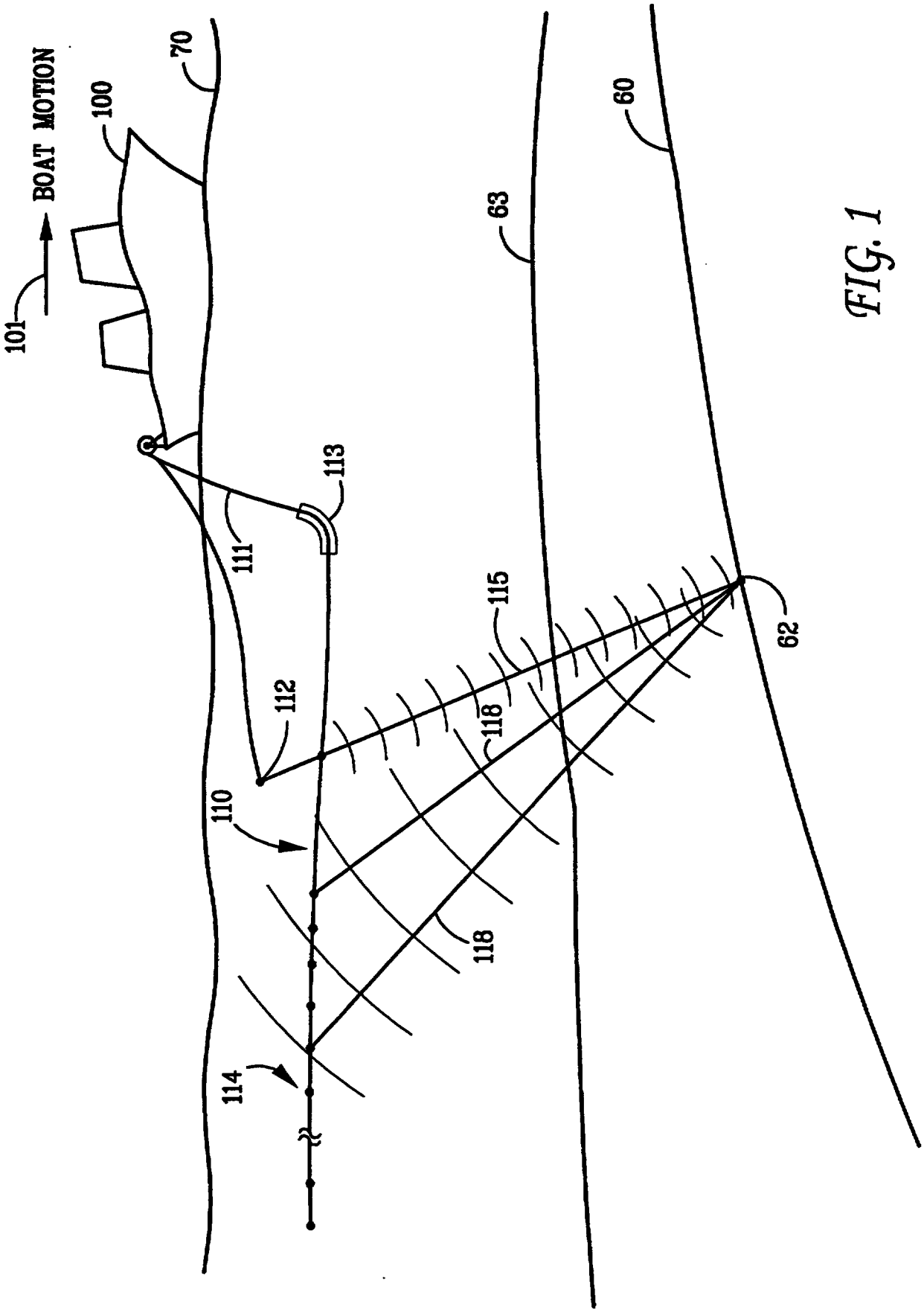
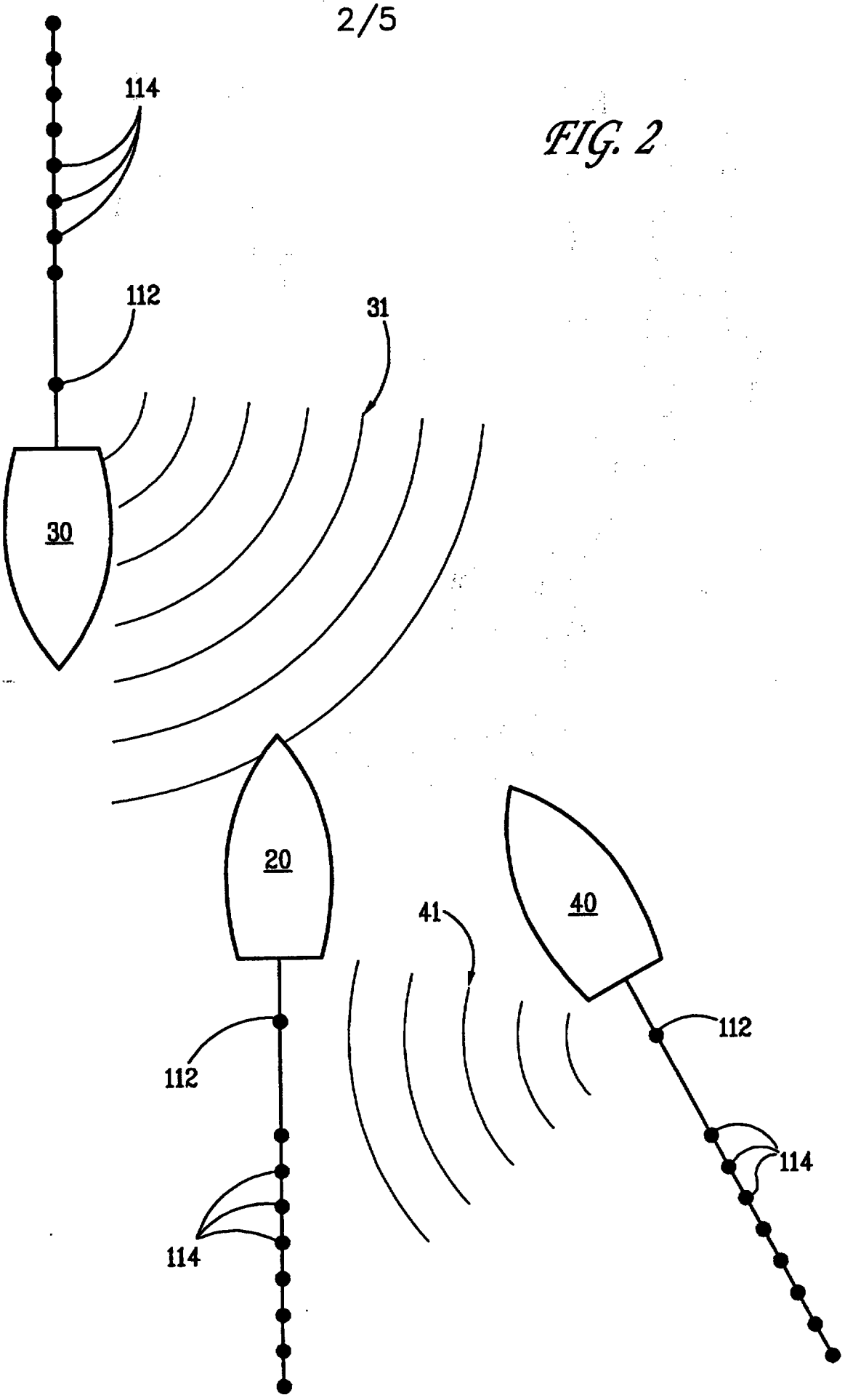


FIG. 1

FIG. 2



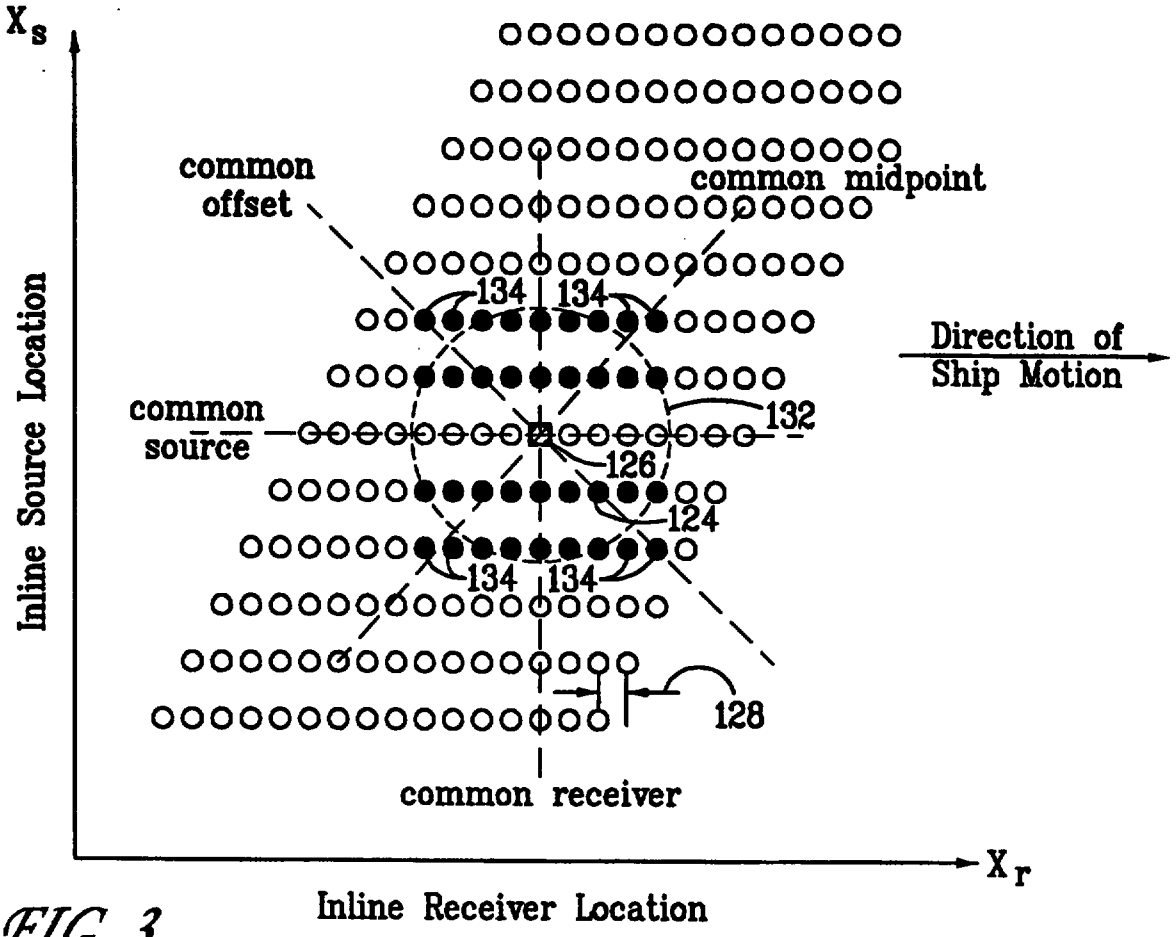

X_s
common offset
common midpoint
134
134
Direction of Ship Motion
common source
132
126
124
134
134
128
common receiver
Inline Source Location
X_r
Inline Receiver Location

FIG. 3

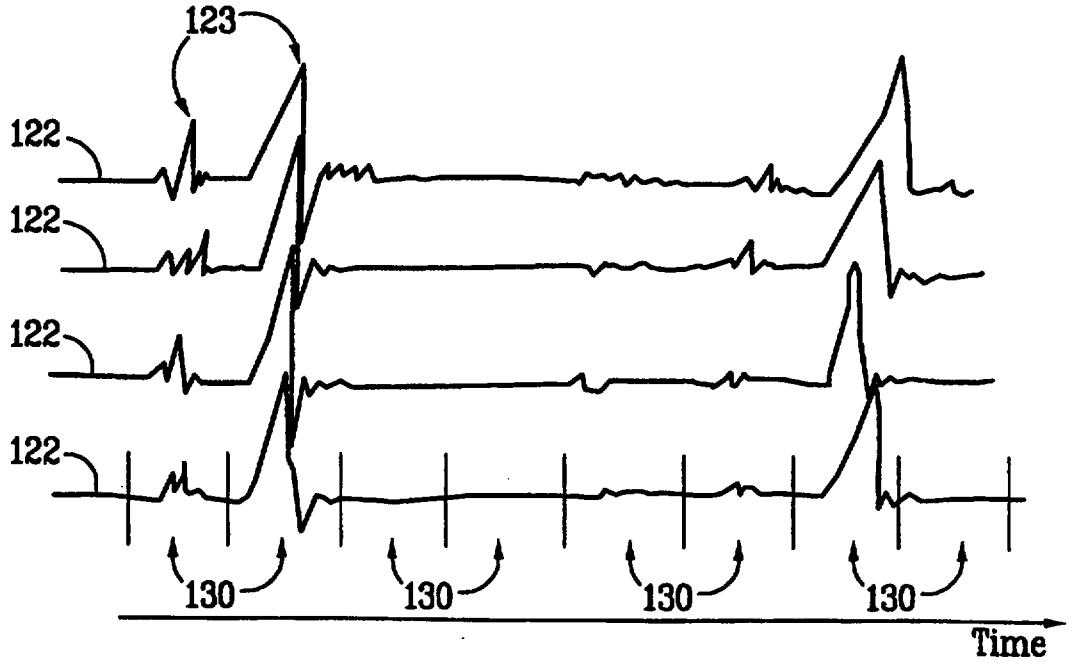

123
122
122
122
122
130
130
130
130
Time

FIG. 4

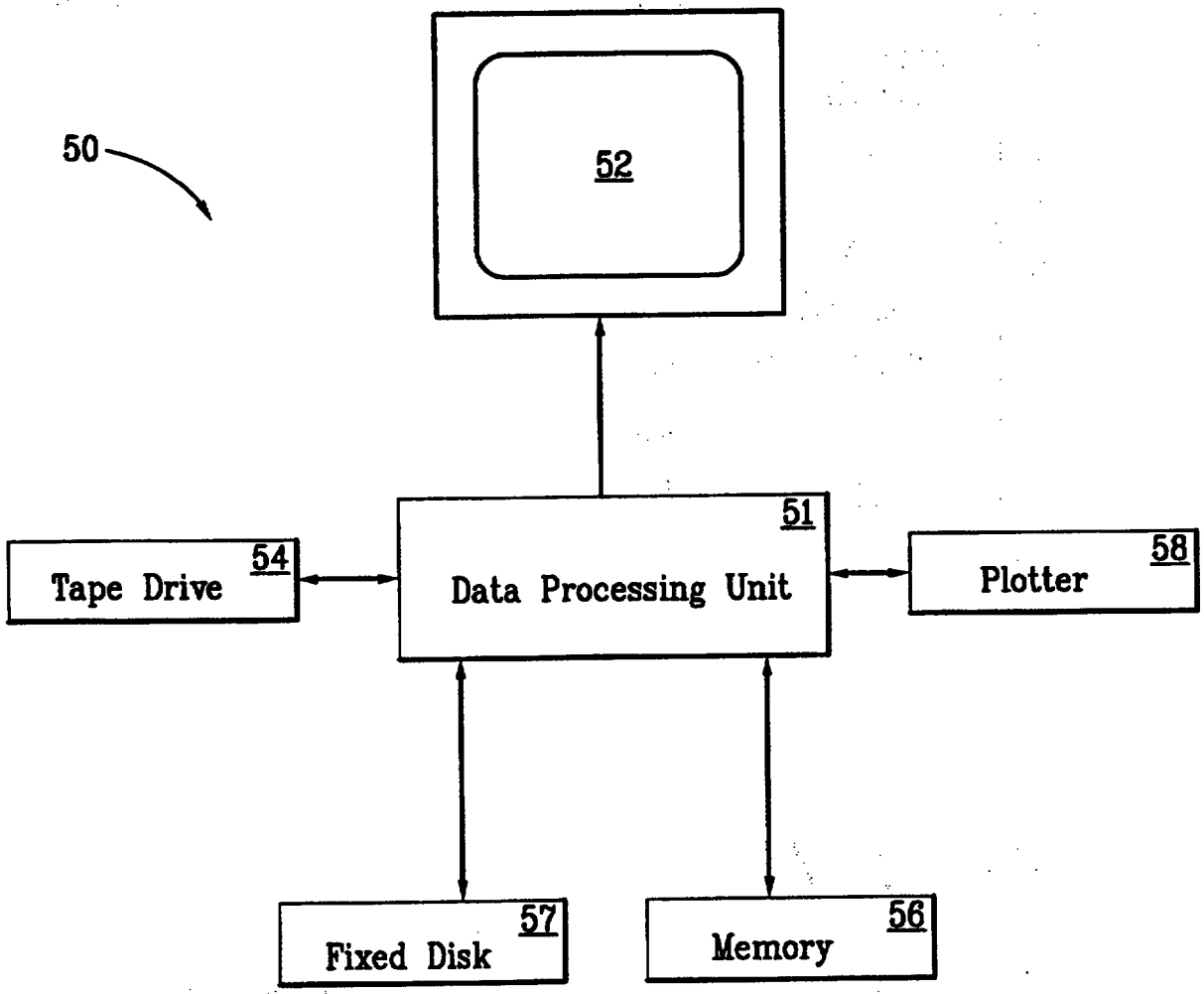

50
52
51
Data Processing Unit
Tape Drive
54
Plotter
58
Fixed Disk
57
Memory
56

FIG. 5

FIG. 6

