GPR_XFRM

Version: 1.04.17.01
Last revision date: 4-17-2001
Description: GPR_XFRM transforms GPR data from traces collected evenly in time to evenly spaced, spatially located traces. Use GPR_PROC to process the data before transforming. The conversion is performed as follows. The user specifies start and stop locations and a step size, and locations are calculated for the new traces. The user also specifies a bin size (which defaults to the step size but may be smaller or greater than the step size to accommodate dense or sparse data sets). Each bin is centered about a new trace location. A location (user-defined X, Y, and Z values) is assigned each old trace by splining, using the data from the MRK and XYZ files. All traces located within the bin are averaged to create the new trace for each bin. Empty bins are set to the mean value of the input data type (e.g., 32768 for unsigned short integers). Empty bins can be avoided by enlarging the step size or the bin size. X, Y, and Z values are calculated for each bin and saved to disk as an "XYZ" file. A "MRK" file is also saved.

The following computer storage formats are recognized: GSSI SIR-10A version 3.x to 5.x, Sensors and Software pulseEKKO, and SEG-Y. If the storage format does not conform to any of the above or this program is having trouble reading the file correctly, there are options in the CMD file for the user to specify required parameters. Data are stored to disk in the same format as they were read in. An exception is made for early versions GSSI DZT files that had 512-byte headers. A current (version 5.x) 1024-byte header is written into the output file. To convert from one GPR storage format to another, use GPR_CONV.EXE. Because each industry storage format maintains information about the data in each trace header, both the trace header and trace data block are stored as a unit as one column in the program's internal storage grid. If the size of the trace header is not an even multiple of the data element (sample) size, the program will stop and report an error. This should only be a problem with user-defined storage formats.

The input to this program is a "CMD" file, an ASCII text file containing keywords (or parameters) describing how to process the radar data. The CMD file specifies the data file name (and optionally the storage format). Inspect the example file GPR_XFRM.CMD for usage.

NOTES:
Only 1 to 4 headers are supported in DZT files.
There is no graphic display of the data.
To display the processed data, use programs such as GPR_DISP.EXE or FIELDVEW.EXE.

THE KEYWORDS

Following is the list of keywords and their default values. The documentation format is:
"KEYWORD: **keyword** = default value".
Look at GPR_XFRM.CMD as an example command file with correct usage and default keyword values. The file GPR_XFRM.CMD has most comments stripped out, and GPR_XFRM.CM_ has all comments removed.

********** PROGRAM CONTROL **********

KEYWORD: **batch** = "FALSE"
Place the program in batch mode (no pauses) if "TRUE". The program will normally pause at times before ending.

KEYWORD: **debug** = "FALSE"
Place the program in debug mode if "TRUE" (for developers)

********** SPECIFICATION OF INPUT DATA **********
The storage format is determined by inspecting the file. If the program cannot recognize a flavor of the three formats below then an error message may be issued.
Recognized storage formats are:
DZT GSSI SIR-10A file with embedded (512- or 1024-byte) info header
DT1 Sensors & Software pulseEKKO file with matching HD info file
SGY SEG SEG-Y format
DT1 and HD files are assumed paired, i.e. both have same filename with different extensions. So, if a data file with a ".DT1" extension is specified, the ".HD" filename will be assumed. Only DT1/HD files must have those filename extensions.

KEYWORD: **dat_infilename** = ""
This is the input GPR binary data file name.

KEYWORD: **channel** = 0
This keyword is for use with multiple-channel DZT files only. It is the channel number in multi-channel data sets and is indexed from 0. GSSI data can have up to 4 channels (channel = 0, 1, 2, or 3).

KEYWORD: **mrk_infilename** = ""

KEYWORD: **xyz_infilename** = ""

MRK and XYZ files are used to determine which traces are "marked" and what the coordinates are of the marked traces. They contain the number of sets stated on the first file record with the sets listed on following records.

Example MRK file containing marked trace locations:
```
3
104
256
897
```

Example XYZ file containing X, Y, and Z locations of the marked traces:
```
3
10.0  10.0  293.456
20.0  10.0  294.567
30.0  10.0  295.678
```

RAMAC and user-defined data files can be read by assigning correct values to the next five keywords.

########## NOTE ##########
IF the GPR format DOES NOT CONFORM to any of the above formats then the next six parameters (other_format, file_header_bytes, trace_header_bytes, samples_per_trace, total_time, and input_datatype) MUST be specified. Otherwise, IGNORE THEM. If you want to convert the storage format then GPR_CONV is the program to use. GPR_INFO will report this basic information for recognized storage formats.
##############################

KEYWORD: **file_header_bytes** = 0
Replace with number of bytes in the file header. PulseEKKO data files do not have a file header - the information is held in another file with a .HD extension. GSSI files have either a 512-byte (old style) or 1024-byte (current style) header. However, DZT files can have up to 4 file headers - one for each channel. SEG-Y files have a 3600-byte header. RAMAC data files have no file header.

KEYWORD: **trace_header_bytes** = 0
Replace with number of bytes in each trace header. For pulseEKKO files, a 128-byte header precedes each GPR trace. For GSSI files, no header precedes each trace, but the first 2 samples (not necessarily bytes) are reserved. SEG-Y files have a 240-byte trace header. RAMAC data files have no trace headers.

KEYWORD: **samples_per_trace** = 0
Replace with the number of samples per trace. For pulseEKKO data, the number of samples per trace is recorded in the HD file (NUMBER OF PTS/TRC). For GSSI data, the number of samples per trace is a power of 2, from 128 to 2048, typically 256, 512, or 1024. The information is recorded in the DZT file header in the rh_nsamp field. For RAMAC files, the RAD text file records the number of samples. For SEG-Y files, look in the comment area of the file header.

KEYWORD: **total_time** = 0
Replace with total number of nanoseconds per trace. For pulseEKKO data, look at the "TOTAL TIME WINDOW" field in the .HD file. For GSSI data the value is recorded in the file header. For SEG-Y files, look in the comment area of the file header. For RAMAC files, the TIMEWINDOW parameter records the time per trace in microseconds (multiply by 1000 to get ns).

KEYWORD: **input_datatype** = 0
This defines the type of input data element. Replace with one of the following element types:

```
 1 for 1-byte signed characters
-1 for 1-byte unsigned characters           (GSSI)
 2 for 2-byte signed short integers         (pulseEKKO, RAMAC, SEG-Y)
-2 for 2-byte unsigned short integers       (GSSI)
-5 for 2-byte unsigned short integers, but only first 12-bits used
 3 for 4-byte signed long integers          (SEG-Y)
-3 for 4-byte unsigned long integers
-6 for 4-byte unsigned long integers, but only first 24-bits used
 4 for 4-byte floats                        (SEG-Y)
 8 for 8-byte doubles
```

For example: 8-bit GSSI data are unsigned characters (values from 0 to 255), use -1 for input_datatype. Use –2 for 16-bit GSSI data (values from 0 to 65535). PulseEKKO and RAMAC data are typically 16-bit signed integers (values from -32768 to 32767), use 2 for input_datatype. For SEG-Y data, the input_datatype can be 2 (signed short integers), 3 (signed long integers), or 4 (4-byte floating point reals). Data types are stored in the file header of DZT and SGY files. PulseEKKO and RAMAC do not record the data type.

********** SPECIFICATION OF OUTPUT FILE **********
The processed GPR data are stored in the SAME FORMAT as the input data. Use GPR_CONV.EXE to convert between storage formats.

KEYWORD: **dat_outfilename** = ""
This is the name of the binary GPR data file that is written to disk. For S&S files, an "HD" file will also be created. Note that output files are single-channel only.

********** SELECTING TRANSFORMING OPTIONS **********
This group determines how the GPR data are transformed. Each trace in the input data file is assigned an X, Y, and Z value based on the XYZ and MRK files that were supplied. You must select either the "X" or "Y" direction to use as the reference axis for the new, binned, output data.

KEYWORD: **spatial_dir** = 0
This is the reference axis to use for the binning operation and for the "spatial_....." keywords. Select 0 for the X-axis, or 1 for the Y-axis.

KEYWORD: **spatial_start** = 0.0
This is the starting coordinate on the reference axis.

KEYWORD: **spatial_stop** = 0.0
This is the ending coordinate on the reference axis.

KEYWORD: **spatial_step** = 0.0
This is the uniform distance between binned output traces.

############ NOTE #############
"spatial_start" and "spatial_stop" must have the same directional sense as the coordinates in the XYZ file. For example, if "spatial_dir" is 1 and the Y values in the XYZ file are increasing as trace numbers are increasing, then "spatial_start" must be less than "spatial_stop" and "spatial_step" must be positive. On the other hand, if the Y values decrease as the trace numbers increase, then "spatial_start" must be greater than "spatial_stop" and "spatial_step" must be negative.
##############################

KEYWORD: **bin_size** = 0.0
This is the size of the bin used to calculate new traces. If it is less than or equal to 0.0, then it is the same as "spatial_step" size. If it is greater than 0.0, then it can be less than or greater than "spatial_step". The bins are centered about the new trace locations (traces are "placed" at the center of a bin) determined from the above 4 values. All traces that are located in a bin are averaged to create the new trace (this is called stacking).

Usage: `GPR_XFRM cmd_filename`
Required command line arguments:
cmd_filename - The name of the keyword file.
Optional command line arguments (do not include brackets): none
Examples:

```
gpr_xfrm xfile1.cmd
```