UNITED STATES DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

Digital Mapping of Sidescan Sonar Data With the Woods Hole Image Processing System Software

By

Valerie Paskevich[1]

Open-File Report 92-536

This report is preliminary and has not been reviewed for conformity with U.S. Geological Survey Editorial standards. Use of tradenames is for purpose of identification only and does not constitute endorsement by the U. S. Geological Survey.

September 1992
(Revised June 1993)

[1]Woods Hole, MA. 02543

CONTENTS

FIGURES

Abstract

Since 1985, the Branch of Atlantic Marine Geology has been involved in collecting, processing and digitally mosaicking high and low resolution sidescan sonar data. In the past, processing and digital mosaicking has been accomplished with a dedicated, shore-based computer system. Recent development of a UNIX-based image-processing software system includes a series of task specific programs for pre-processing sidescan sonar data. To extend the capabilities of the UNIX-based programs, development of digital mapping techniques have been developed. This report describes the initial development of an automated digital mapping procedure. Included is a description of the programs and steps required to complete the digital mosaicking on a UNIX-based computer system, and a comparison of techniques that the user may wish to select.

Introduction

To process, map and digitally mosaic sidescan sonar data, the Branch has utilized the Mini Image Processing System (MIPS) software (Chavez, 1984) first on a Digital Equipment Corporation (DEC) PDP 11/73 computer system with RSX-11/M+ operating system and, more recently, on a DEC MicroVAX-II computer system with VMS operating system. Recent development of a UNIX-based image processing software with sidescan sonar specific programs has resulted in a shift to a UNIX-based system. To extend the UNIX-based sidescan sonar support, a series of task specific programs were written to support the development of digital mapping techniques for this unique dataset. It is assumed that the user is familiar with the sidescan sonar pre-processing requirements and detailed discussion of that subject is not to be found here. Discussion of sidescan sonar processing techniques are available elsewhere, for example, Chavez, 1986, Miller et al, 1991, Paskevich, 1992. Paskevich (1992) provides an in-depth discussion of pre-processing using the UNIX based software.

In the past, digital mapping and mosaicking of sidescan sonar data has been a user intensive job. To place a swath of sidescan sonar data within a specific map area, the user was required to select 4 points along nadir to be used to tie the sonar strip down to the map area. Additional bookkeeping and mathematical computations were required by the user to provide the necessary information to the various programs. To complete the initial mapping process, the user needed to execute several programs that included geometric transformation routines. Interpolation and rubber-sheet stretching of the imagery between the selected points was applied by the transformation programs to place the selected line segment in the requested cartographic space. This procedure was successful with straight-line segments, but was not useful with sinuous tracklines. Once the individual swaths were mapped, they often required additional registration and warping to match features from adjacent swaths.

For this new mapping procedure, the sidescan sonar data must be in the WHIPS format (Paskevich, 1992) which utilizes the Unidata NetCDF data access software (Unidata, 1991) and must contain the necessary sonar fish information. The new mapping procedure provides a simpler, more automated, method of mapping sidescan sonar data and works well with sinuous track lines. In general, a swath of data which took approximately 1 to 1 1/2 hours to prepare and map with the older method may now be accomplished in approximately 20 to 30 minutes with less user intervention. Of course these timings may vary depending on the size and scale of the sonar image being mapped. The new method also takes into account the fish heading of the individual scan rather than an overall heading of the entire swath and provides a more accurate mapping of the sonar scans and swaths.

One drawback to the method discussed in this report can be the holes or gaps that occur during the placement of the consecutive lines of sonar data. Since the heading values for the sonar are critical in placing the sonar pixels, minor fluctuation in headings that can occur between the adjacent lines can cause the lines to overwrite on another rather than being placed neatly alongside each other. The overwriting of the adjacent lines and the resulting gaps can be minimized by smoothing the heading or fish coordinates contained in the sonar header before beginning the mosaicking procedure.

By processing the sidescan sonar data on UNIX workstations, we are able to take advantage of the UNIX program **proj** (Evenden, 1990). Program **proj** currently allows the user to select from approximately 70+ map projections for their final map product. Program **proj** is fully documented. The source code and documentation for **proj** may be obtained via anonymous ftp at kai.er.usgs.gov (128.128.40.69).

With the exception of program **proj**, programs utilized to complete the sidescan sonar mapping are a subset of WHIPS. Complete documentation for the WHIPS programs utilized in the mapping procedure, along with various utility programs, may be found in Appendix A. Those programs would include:

avg_heading	apply a running average to the heading values contained in a WHIPS netCDF sidescan sonar image header.
avg_position	apply a running average to the fish positions contained in a WHIPS netCDF sidescan sonar image header.
dk2dk	create a new WHIPS netCDF image file by extracting a sub-area from an existing WHIPS image.
filter	applies a low-pass, high-pass, zero replacement or divide filter to an image.
gss	computes geographic coordinates of sidescan sonar pixels.
gss_vel	computes geographic coordinates of sidescan sonar pixels and applies the necessary velocity correction to the sonar swath.
gssv	computes geographic coordinates of sidescan sonar pixels from a velocity corrected image.
listhdr	list the contents of a sidescan sonar header in a WHIPS image.
lowpass2b2	applies a 2-by-2 low-pass filter to an image.
median3	applies a 3-by-3 median filter to a WHIPS image.
mode3	applies a 3-by-3 mode filter to a WHIPS image.
mode5	applies a 5-by-5 mode filter to a WHIPS image.
projss	place projected sidescan sonar data in an image file which represents a specific map area.
qmos	Mosaics (overlays) the specified input file over the specified output file. The program will either overlay where the input file has priority over the existing output file, where the output file has priority over the input file, or average non-zero pixel values together from the input and existing output file.
quickview	displays an 8-bit WHIPS netCDF image in an X-11 window.
raw2whips	convert a raw image data to a netCDF file for use with WHIPS

ssdk2dk	create a new WHIPS netCDF sidescan sonar image file by extracting a sub-area and the accompanying header information from an existing WHIPS netCDF sidescan sonar image.
sshead	computes a simple heading for a WHIPS netCDF sidescan sonar image.
sumss	summarize the information contained in a sidescan sonar header for mapping.
whips2raw	convert a WHIPS netCDF image file to a raw image file

As stated earlier, the user should be familiar with the general pre-processing requirements for sidescan sonar data, which for this system is summarized by Paskevich (1992). However, it is additionally important to stress the necessity of having as accurate a fish position and heading values contained in the sidescan sonar header as the user can supply. Pre-processing of the sidescan sonar data may take place without this information contained as part of the header information, but the fish heading and positions are critical to the placement of the sidescan sonar pixels in the mapping procedure. It is the user's responsibility to ensure that the information contained in the header is as accurate as can be obtained. How the user prepares the information and combines it with the sidescan sonar data is not within the scope of this report. As newer sidescan sonar systems record the fish position and heading values, this automated recording may alleviate the user from navigation requirements. However, older systems rely on the ship position and heading rather than having an accurate fish position and heading.

Mapping Procedure

There are three essential steps in mapping a swath of sidescan sonar data. The first step is to compute the geographic coordinates of each pixel within the sonar image. This step is accomplished by selecting one of three programs: **gss**, **gssv** or **gss_vel**. These three programs are essentially the same with the difference being in the input file that they accept and process. The second step is to convert those geographic coordinates to meter coordinates for a specific map area, scale, projection and spheroid. The final step is to properly place the projected coordinates in a WHIPS netCDF image file that represents the map area. These three steps are accomplished by executing one of the **gss** programs, **proj** and **projss**. These programs and processing procedures are preliminary and are subject to change.

The three steps may be executed separately with each step creating the appropriate output file. However, the programs may be scripted together by utilizing UNIX std_in and std_out files. In this manner, there would be less disk space utilized since the data would pass transparently from program to program. The programs, as they may be scripted together, are shown below. The actual program parameters required would differ according to the data and map scale selected.

```
 % gss_vel -i gloria.img -r 50 | proj_info | projss -o gloria.map
```

The file *proj_info* contains script information that would define the cartographic information required by program **proj** along with the map boundaries required by program **projss**. Though the information contained *in proj_info* could be entered as part of the run-line above, it is better to input the information into a file to eliminate the possibility of typographic errors when the information is required for subsequent executions. In addition to the cartographic information contained in file *proj_info*, the file also references an additional file that contains the geographic coordinates for the corners of the requested map area. A more detailed discussion of the programs and their parameters are discussed in Mapping Example.

Geographic placement of the sidescan sonar image pixels.

The first step in mapping the sidescan sonar imagery is to compute the geographic coordinates associated with each pixel contained in the sonar image. This step is accomplished by one of three programs: **gss**, **gssv** or **gss_vel**. The major difference between the programs is the input file that they accommodate. Though each program requires the input file to be in a WHIPS netCDF sidescan sonar image format, the actual pre-processing that was applied to the sidescan sonar image swath up to this point would be different. A brief description of the file differences follows.

Each of the programs mentioned above will compute the geographic coordinates for the pixels from within the selected part of the sidescan sonar image and each program computes the pixel coordinates in the same manner. The pixel coordinates are computed by utilizing the heading and nadir latitude and longitude coordinates contained in the WHIPS sidescan sonar header. With the user supplied image pixel resolution, the program computes the pixel coordinates from nadir to the far range of the image. The coordinates are computed first for the port and then starboard side of the individual scans.

The WHIPS netCDF sidescan sonar image must contain a valid fish heading value in the sidescan sonar header. If the data set does not contain fish heading values, a heading can be computed from the fish positions contained in the header with program **sshead**. Program **sshead** computes the heading between consecutive lines and places the computed value in the sidescan sonar header of the output image file.

Program **gss** will process a sidescan sonar image that has not been through the pre-processing program velocity. A geographic position for each pixel contained in each sonar swath is computed and output to the UNIX std_out device. The program applies no correction for along-track resolution. **Gss** processes each and every user-specified line and sample from the input file. If the input file has been processed by program **velocity**, and multiple lines with the same nadir coordinate exist in the input file, the multiple lines will be processed over one another. This is obviously redundant processing to the program and time consuming. When attempting to create a map at a scale equal to or less than the input data, gaps will result between consecutive lines because of the greater resolution in the across- versus along-track direction of some data. Program **gss** is best suited for sidescan sonar imagery data in which the along- and across-track distances represented by a pixel are approximately equal. Program **gssv** should be used if the sonar image has been pre-processed through velocity.

During pre-processing of some sidescan sonar data, the data requires a velocity correction be applied to it to produce an image with equal across- and along-track resolution. Generally, this correction has been accomplished by computing the distance between a given segment of lines. With the distance traveled from the beginning of the segment to the end known, a line duplication factor is computed. This line duplication factor represents how many times each individual line should be duplicated to produce an output image with equal across- and along-track resolution. The segments of line are then processed by duplicating the individual lines in the current segment to produce an image with equal across- and along-track resolution. In duplicating the image lines, the associated sidescan sonar header is also duplicated, resulting in an image, which upon closer examination has multiple lines with the same nadir coordinate. Program **gssv** will handle a sidescan sonar image that has been velocity corrected in this manner. This program differs slightly from program **gss** in that it will not place the lines with the same nadir position over each other. When consecutive lines with the same nadir coordinates are encountered, **gssv** will compute a placement of the line adjacent to the previous line one using the heading value from the sidescan sonar header and the user supplied image pixel resolution. With the offset computed for the duplicate line and applied to the nadir position, the result is to have the duplicate lines placed side-by-side. This helps reduce the gaps which otherwise would appear between individual scan lines. This approach however can result in a blocky image being produced by excessive replication of the sonar swaths when attempting to produce an image with equal across- and along-track resolution.

Program **gss_vel** combines the functions of pre-processing a sonar image with **velocity** and the pixel computations of program **gss**. As **gss_vel** is processing the sidescan sonar input image, the distance traveled between consecutive lines is computed. Using the image-input resolution supplied by the user and the computed distance between the consecutive lines, a line duplication factor is then computed. This is somewhat different than applying a velocity correction to the image as part of the pre-processing because the duplication factor is computed and applied to consecutive lines rather than a general duplication factor for a segment of lines as is done in the pre-processing. As with program **gssv**, program **gss_vel** will compute the offset for the required duplicate lines and place the consecutive lines along-side each other. This, as with utilizing a velocity corrected image and program **gssv**, will help reduce the gaps that can occur between adjacent lines.

Whether selecting a velocity corrected image or non-velocity corrected image, these programs may be used to process sonar data with a sinuous track, as it will properly place the lines in the map space. The user selecting a map resolution that best accommodates the input data resolution can minimize the gaps that may occur. However, gaps and holes that may result, particularly around sharp turns, can be filled as part of the final mapping steps.

Good heading and fish positions are critical to placing the sonar swaths. If the heading or fish positions are not smooth, the placement of the lines will appear to be somewhat scattered and will overwrite data. It may be necessary, as part of the data preparation prior to beginning the mapping procedure, to smooth the fish position or heading values so the placement of adjacent lines will fall parallel to each other. Applying either avg_heading or avg_position or both to the WHIPS netCDF sidescan sonar image can do this. These programs each apply a user specified running average over the appropriate sonar header values to smooth the values for mapping. An example of utilizing avg_heading may be found under Mapping Example.

These programs will, by default, compute the geographic coordinates for each of the sidescan sonar pixels contained in a WHIPS netCDF sidescan sonar image file. The user may optionally specify a sub-area of the image to be processed. Processing always takes place from nadir out to the far range. Only positions with valid pixels (non-zero values) are output.

Program *proj*

This program is critical to the user's creation of the final map area and represents the second step in completing the mapping procedure. As used in this processing scenario, **proj** accepts as input the computed geographic coordinates of the pixels from program **gss**, **gssv** or **gss_vel** and converts the longitude and latitude coordinates into Cartesian coordinates.

Program **proj** operates as a standard UNIX filter utility. Input data may be piped directly into proj. Output of the program is to std_out and would generally be piped directly to the third and final program in the mapping procedure.

Proj utilizes many optional parameters such as mapping spheroid selections. However, some parameters, which are defined as optional in the **proj** documentation, must be specified to accommodate the format of the input data and properly format the output data. The optional parameters, which must be specified, are:

`-f `%.0f'`	Specifies the format string for writing the output values. This format will round-up the Cartesian values and output them as integer values.
`-r`	This option reverses the order of the input values from the expected longitude-latitude values to latitude-longitude.
`-s`	This option reverses the order of the output values from longitude-latitude to latitude-longitude.

Program **proj** supports approximately 70+ cartographic projections. The user must properly specify any of the projection specific parameters that may be required by the selected projection. It is essential that the user have a good understanding of the available map projections and the required projection parameters to create the desired map. For more detailed cartographic characteristics of the projections, the user may refer to additional documentation (Snyder, 1988 or Snyder and Voxland, 1989).

In addition to the required **proj** parameters and input data to be projected, the user must supply the bounds of the selected map area. The map bounds must precede any pixel data to be projected. The map corner coordinates must be specified in the DMS system utilized by **proj** and must be specified in the following order:

upper left
upper right
lower right
lower left

Specifying the map boundaries is best accomplished by entering the coordinates in a separate file. The file may then be specified on the **proj** run-line before the specification of the input data. This will be discussed further under ***Mapping Example***.

Program projss

Program **projss** accepts the output from program **proj** and places the pixels in their proper image location. The program begins by reading the first four pairs of coordinates passed to it from program **proj**. **Projss** assumes that these values are the map boundaries in Cartesian coordinates. With the four map corners known, the program computes the size of the image area. Since the image must be defined as a rectangle, the maximum coordinates are used to compute the image size. For some map projections (i.e. conic projections), the desired map area will be smaller than the actual image size because of the arc of the longitude lines. Figure 1 shows the typical placement of a Universal Transverse Mercator (UTM) map in a defined image area and the map corners are designated as points **A**, **B**, **C** and **D**.

After the image size has been defined and created, the program continues by placing the sidescan sonar pixels in their proper map space. Pixel Cartesian coordinates that are outside of the defined map area and yet fall within the defined image area are placed in the output image. The image is filled on a pixel-by-pixel basis as the pixels are individually placed. There are three drawbacks to the pixel-by-pixel method, and they are discussed below. Note of the drawbacks are made to warn the user of possible problems which could affect the quality of their final maps when using the **gss_vel** | **proj** | **projss** processing scenario, and to discuss possible future processing alternatives.

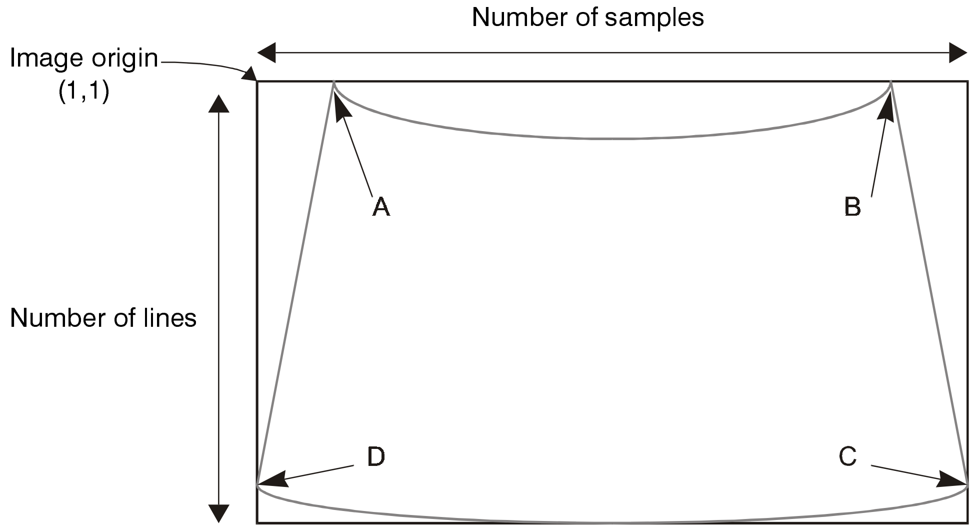

Figure 1 - UTM map placed in image space.

The first drawback is due to the random order of the input pixels being placed. This random input results in a last-in placement for coordinates that may have two or more valid pixels. Since the program can not place in order the multiple values for a given coordinate, the program must deal with each pixel as a unique value and is placed in it's coordinate location regardless of any pixel values which may have been previously placed at that point. Even a simple comparison for placement of pixels in non-zero locations and averaging of the pixels would be restrictive due to the overhead placed in computing.

Secondly, it would be more desirable to order the pixels based on their sorted placement in the image area. If the input pixels could be placed in sorted order from the image origin down to the last line and last sample to be placed, processing could be accomplished by "building" a complete image line. This would result in fewer writes to the output file since a complete line of image data could be written rather than the many single writes which must be done for each pixel input. Fewer writes would speed up processing. However, when processing over a half million pixels at a time (a modest file at best), speed may be best accomplished by increased workstation performance. The sorted order of the data would also be beneficial by allowing the program to compare multiple pixel values for a specific coordinate and select the final output pixel value by averaging or selecting either the minimum or maximum value.

The third drawback is the "holes" which develop between lines of data as the pixels are placed in the *map* space. The "holes" may be best eliminated, or reduced in number, by the user carefully selecting the resolution of the sonar data being processed with the goal in mind of the final resolution of the map to be created. If the user selects to pre-process the sonar swaths at a half meter resolution and the final map resolution is 1 meter, fewer "holes" will develop during the mapping stage than if the output image is at a .5 meter pixel size. An obvious drawback is the volume of pixels that will have to be processed resulting in longer execution time. The user must consider the trade-off of processing such enormous volumes of data or what "holes" may be created in the mapping process. These "holes" should be filled in some manner to produce a complete map image. The preferred method would be some form of interpolation based on the orientation of the scan line and the appropriate neighboring pixels. To accomplish such a task, the sonar data must be processed on a swath-by-swath basis with two consecutive swath's being processed at a time. Unfortunately, the ability to handle the data on a pixel-by-pixel basis is the only option currently available. This results in the "holes" or gaps that may by created to be filled in some manner after the sonar map has been created. Current options include a low-pass filter replacing zero values, a 3-by-3 or 5-by-5 mode or median filter, or some combination of those filters. The "filtering" options are less than perfect since the zero pixels which may be valid image data are replaced and the zero pixels are filled relative to the orientation of the boxcar passing over the image, not the orientation of the scan lines. The boxcar orientation is perhaps better suited for a series of east-west or north-south track lines. However, with careful selection and application of the filters, the sidescan sonar map mosaics with any scan line orientation can be processed and the image "holes" filled without degrading the image significantly.

Mapping Example

The example which follows shows the steps required to map a 6-hour GLORIA (Geological Long-Range Inclined Asdic) sidescan sonar image into a specific map area. The 6-hour pass used in the example, and referred to as *Pass-13*, was initially pre-processed at 45 meters in the across-track resolution and has an average resolution of 160 meters in the along-track direction. The data were collected as the ship passed in a easterly direction with a course heading of 90^o. The selected input image, shown in its non-velocity corrected form as Figure 2, contains 720

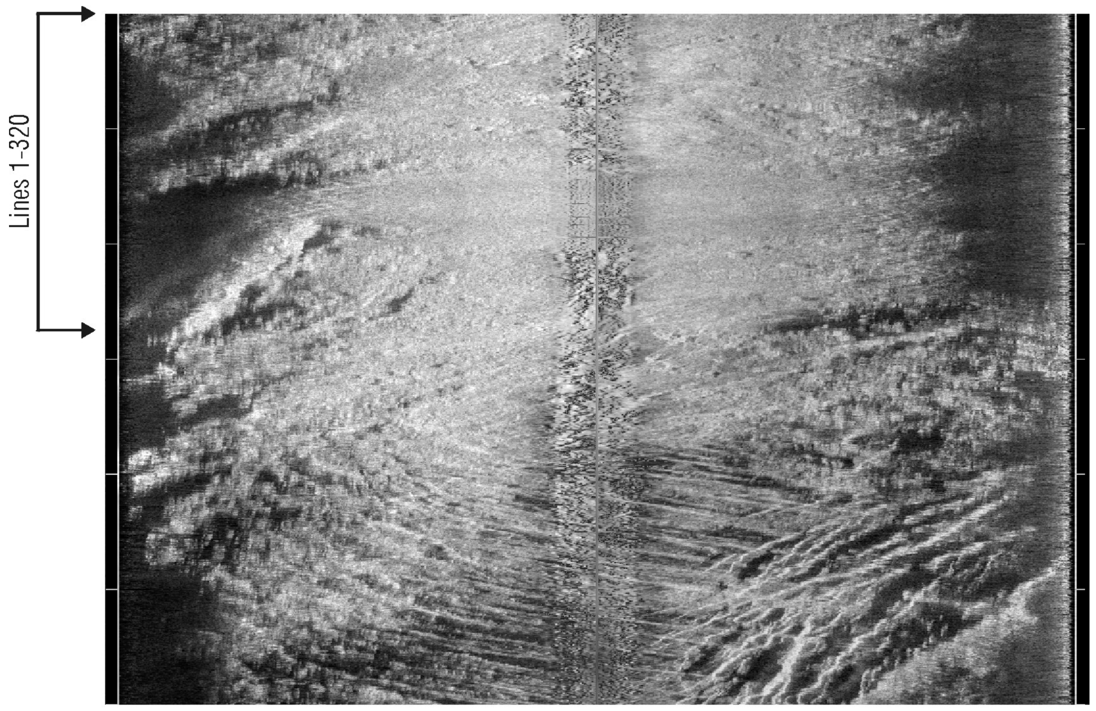

Figure 2. Pass 13 pre-processed without velocity correction applied.

lines and 1024 samples with approximately only lines 1 through 320 falling within the desired map area. Additionally, approximately 16 pixels at the far range of the image contain the hour marks for the sonar data. These hour marks are not to be included in the final map, so the **-s** option with a value of 490 will be specified when executing one of the geographic computation programs (i.e. **gss**, **gssv** or **gss_vel**). Since the image contains 1024 samples (512 samples per side), specifying the **-s** option with 490 samples halts the processing of the individual scans before reaching the hour marks.

To produce an image with equal along and across-track resolution, the image would require a 3.5 aspect ratio correction be applied by program **velocity**. The velocity-corrected image is shown as Figure 3. In many cases, the computed aspect ratio cannot be applied directly because of the fraction computed. In the example of Pass-13, the aspect ratio is 3.5 and an individual image line cannot be duplicated by a half. To accommodate the round-off problem, the velocity program applies a duplication factor that alternates between 3 and 4 by first duplicating a line 3 times, the next line 4 times, the next line 3 times and so forth. Alternating the round off for the duplication factor produces an output segment that will equal the appropriate number of lines for the distance traveled in the given segment. Duplicating the sonar scans produces a blocky image, which, in this example, is most noticeable at the far range of the image. Generally a small low-pass filter (2 lines by 2 samples) is applied to the image during the pre-processing to reduce the affect. For purposes of comparison, no smoothing filter has been applied at this time.

The sonar image is mapped at a Universal Transverse Mercator (UTM) projection at a 100-meter scale. The bounds of the map area are 34.25^o to 35^o south latitude and 112.75^o to 112^o west longitude with the central longitude specified as 111^o west. The mapping parameters for **proj** are defined as:

```
+proj=utm +lon_0=-111 -m 1:100
```

The map bounds may be specified on the **proj** run-line, but it is recommended that they be entered into a file to be passed to **proj** to reduce the possibility of errors and to allow the information to be accessed during additional executions of the mapping procedure. The map corner coordinate pairs must be entered in the order of upper left, upper right, lower right and lower left. The coordinate pairs must be separated by one or more spaces or tabs, and may be entered in the DMS format supported by **proj**. The file *bounds.dat* contains the following information:

```
-34.25          -112.75
-34.25          -112
-35             -112
-35             -112.75
```

The file *bounds.dat*, which contains the map boundaries, is entered on the run-line before the sidescan sonar data file, *side-scan.dat.* With the files entered in this manner, the *bounds.dat* file is processed first and the file *side-scan.dat* processed next. Additional **proj** parameters are required to properly read the input data and format the output data for input to program **projss**. These parameters include *-r*, *-s*, and *-f `%.0f'* to read and produce the formatted data properly. An example of a complete **proj** run-line to produce the desired results follows:

```
% proj +proj=utm +lon_0=-111 -r -s -m 1:100 -f `%.0f' \
                bounds.dat sidescan.dat
```

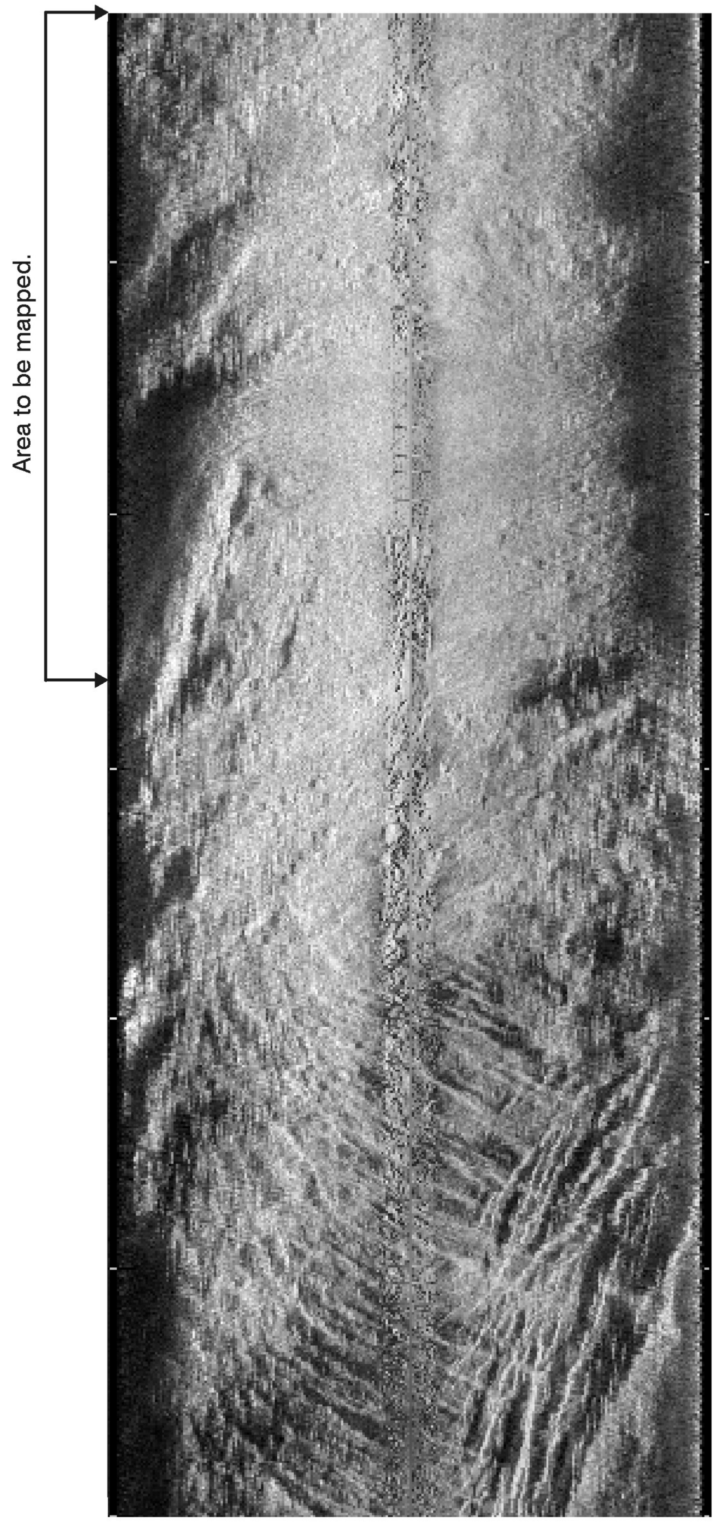

Figure 3. Pass-13 pre-processed with velocity correction applied.

The previous run-line would process the specified data files and output the projected information to the UNIX std_out device. A more practical use of the run-line would be to have the sidescan sonar pixel coordinate data passed to **proj** via std_in. By specifying the sidescan sonar pixels as std_in, the above run-line would translate to:

```
% proj +proj=utm +lon_0=-111 -r -s -m 1:100 -f `%.0f' bounds.dat -
```

Since the above run-line expects the sidescan sonar pixel data to be passed to it via std_in, the **proj** execution must be preceded by selecting one of the geographic coordinate computational programs. The user should select the appropriate program **gss**, **gssv** or **gss_vel** depending on the pre-processing applied to the sidescan sonar data and the desired mapping results. Those programs will process the WHIPS netCDF sidescan sonar image and output the computed geographic coordinates of the pixels to the std_out device to allow piping of the programs.

The data from **proj** is output to the UNIX std_out device and may be re-directed to a user-specified file to be saved for further processing. It is recommended, however, that the user continue the processing stream by piping the output data directly into the third program in the processing scenario, **projss**. By connecting the three programs, the intermediate files that would be created are transparent to the user and only the final map image is stored on disk.

The **proj** run-line should be entered into a file that can be referenced during the processing procedure. This reduces the possibility of errors that can be introduced by the user re-typing the input for repetitive operations. Assuming the user has entered the **proj** information into file utm-proj (the run line listed above), a complete processing scenario can be summarized as:

```
% gss_vel -i pass13.img -r 45 -s 490 -l 1,320 | \
         utm-proj | projss -o pass13.map
```

It appears that the best results for these data are accomplished by utilizing program **gss_vel**. There are some trade-offs the user should be aware of when selecting to map either a velocity or non-velocity corrected sidescan sonar image. Mapping a non-velocity corrected image with program **gss_vel** increases the total time required to complete the mapping part of the procedure. However, applying the velocity correction to the sidescan sonar image during the pre-processing procedures will increase the time to complete the pre-processing phase while increasing the size of the final file by 2 to 3 or 4 times it's original size. This increases the amount of disk space required to complete the pre-processing and store the final image while decreasing the amount of disk space available for the mapping procedure.

In the past, the velocity correction was applied to the sonar swaths by program **velocity**. By utilizing program **gss_vel**, the sonar image can be velocity corrected and mapped at the same time.

To begin, the sonar image (Figure 2) was processed using program **gss** and mapped. From the resultant image, (Figure 4), one can see that the gaps which are produced between consecutive lines are rather large since the map resolution exceeds the actual resolution of the input image. To reduce the gaps between the adjacent sonar scans, the sidescan sonar data must be velocity corrected or the map should be re-scaled to meet the along-track resolution (160 meters). Though re-scaling the map to a 160-meter resolution would result in smaller gaps in the along-track direction, the data in the across-track resolution would be reduced and the final affect would be an image with reduced detail as compared to the original image swath.

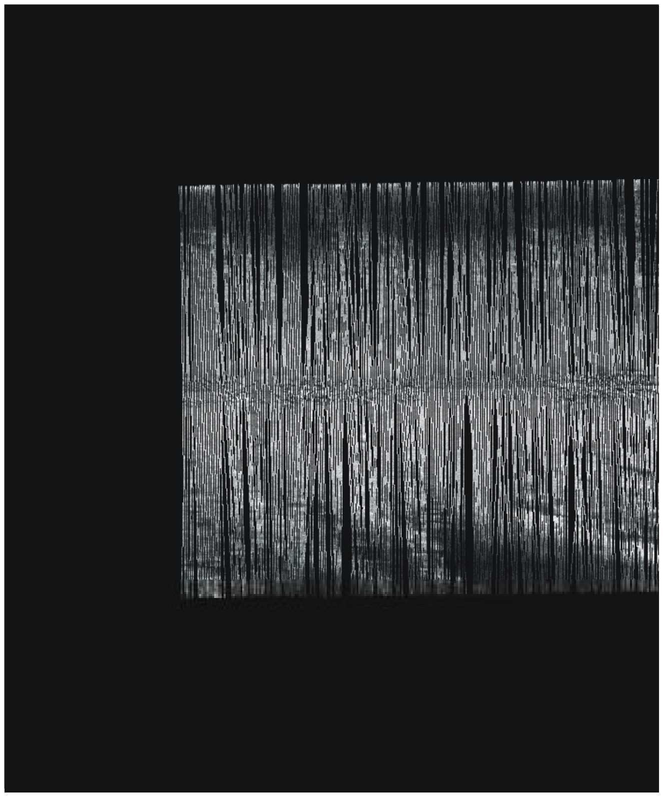

Figure 4 - Pass-13 non-velocity corrected and mapped.

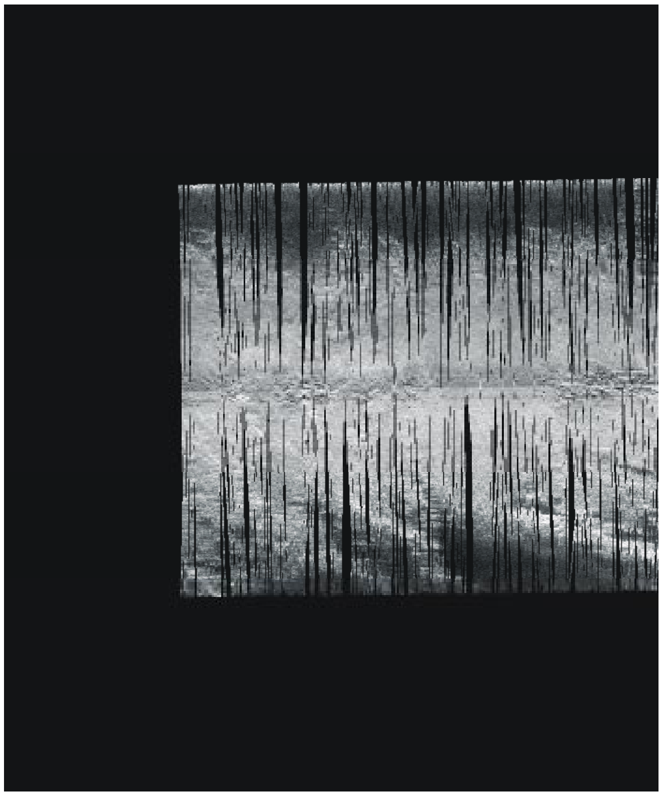
Figure 5 - Pass-13 velocity corrected and mapped.

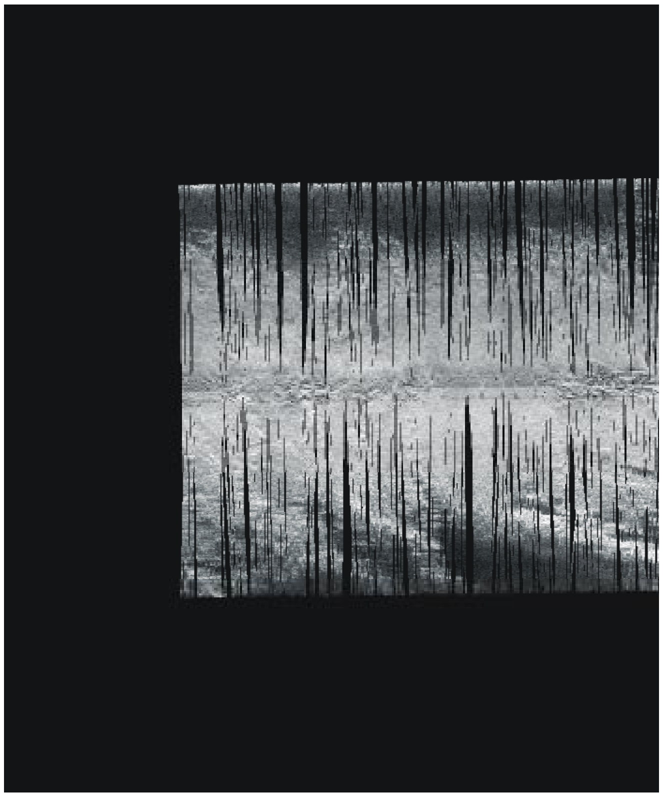
Figure 6 - Pass-13 velocity corrected and mapped simultaneously.

For comparison, the velocity-corrected image (Figure 3) was processed with program **gssv** and mapped with the same parameters. Figure 5 shows the results of mapping the velocity corrected image. It is obvious that utilizing the velocity-corrected image reduces the gaps between consecutive lines and produces a more desirable image.

Since the user may not always want to pre-process the sidescan sonar data through the **velocity** program, **gss_vel** combines the functions of programs **velocity** and **gss**. The non-velocity-corrected image (Figure 2) was processed using program **gss_vel** and was mapped again with the same parameters. The mapped results are shown in Figure 6.

The images processed by **gssv** and **gss_vel** were compared with Figure 5 being subtracted from Figure 6. The resulting difference is shown as Figure 7 and can be attributed to the round-off factor applied during the velocity correction. Generally these two files are the same. These additional lines help reduce the affect of the gaps which can result in the heading fluctuation between lines. The user may select a round-up option when running program **gss_vel** by specifying **-R** on the program run-line. When **-R** is specified, the program will round up the number of lines to be output rather than truncate the number. In other words, if the **-R** option is specified and the duplication factor computes as 3.5, the line will be duplicated 4 times rather than just 3 times.

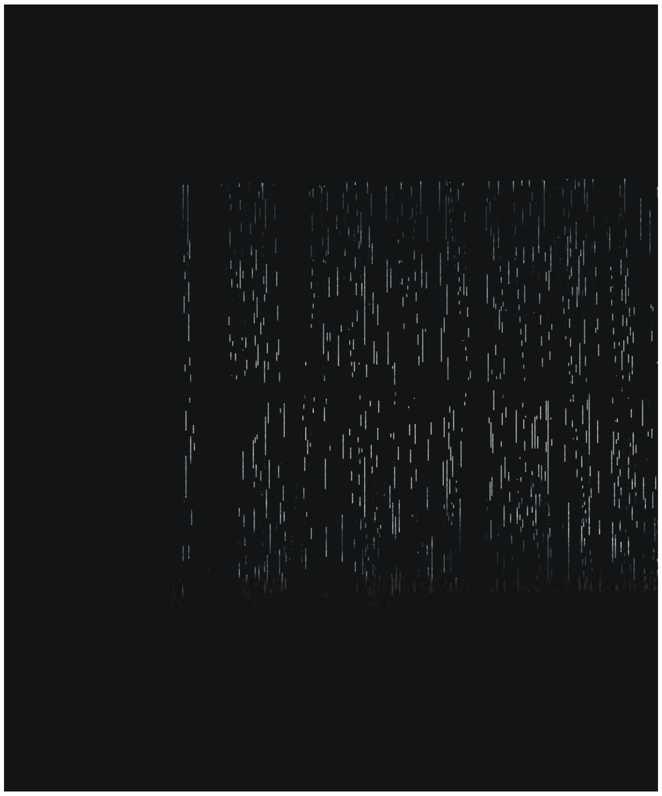
Figure 7 - Difference between Figure 5 and Figure 6.

The fish heading and sonar position values are critical to placing the sonar imagery in its proper Cartesian space. In the past, when digital mapping of the sidescan sonar was done utilizing straight-line segments an overall heading was applied to the entire line. This user-specified heading was then applied to all the scans contained in the line segment. With the same heading value used for each line, the adjacent lines would fall neatly alongside each other. To allow for a smoother heading value for the sidescan sonar file being processed, the user may want to consider smoothing the heading values to allow placement of the consecutive lines falling neatly alongside each other. The following examples (Figures 8, 9, 10 and 11) show the affect of mapping the same image as shown in Figure 2 after the heading values have been smoothed by program **avg_heading**. The pixel coordinates of the sidescan sonar data have been computed using program **gss_vel** so as to apply the velocity correction during the mapping. The Figures show the affect of having a 3, 5, 7 or 9-line average applied to the heading values before the mapping is done. The user should experiment with applying the smoothing filters to determine an acceptable averaging size.

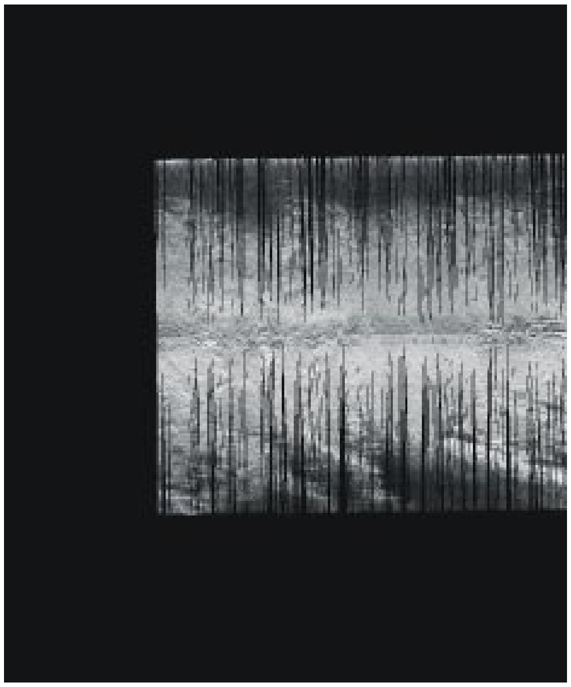

Figure 8 - Sonar swath mapped with heading averaged by 3 lines.

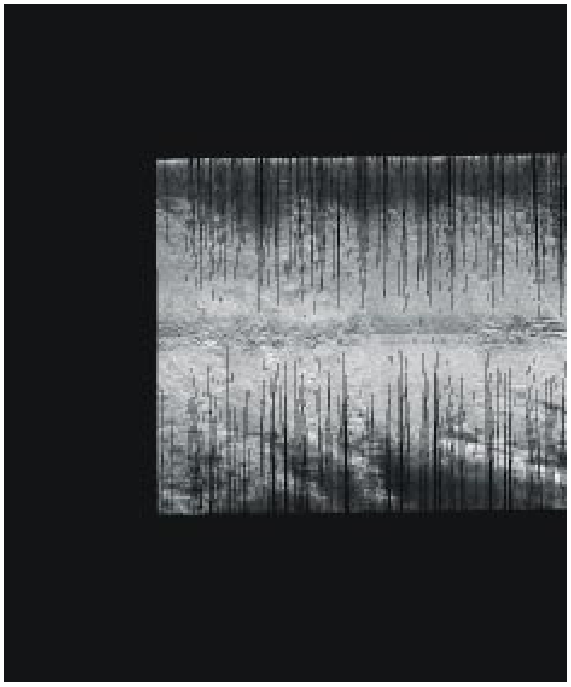

Figure 9 - Sonar swath mapped with heading averaged by 5 lines.

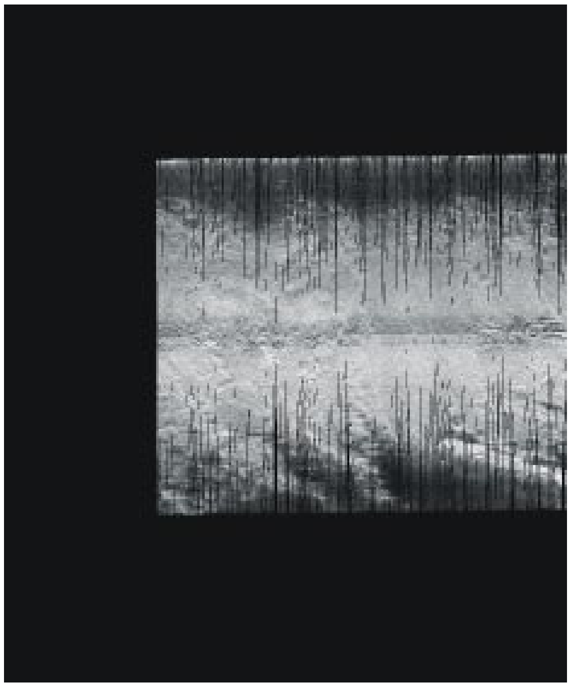

Figure 10 - Sonar swath mapped with heading averaged by 7 lines.

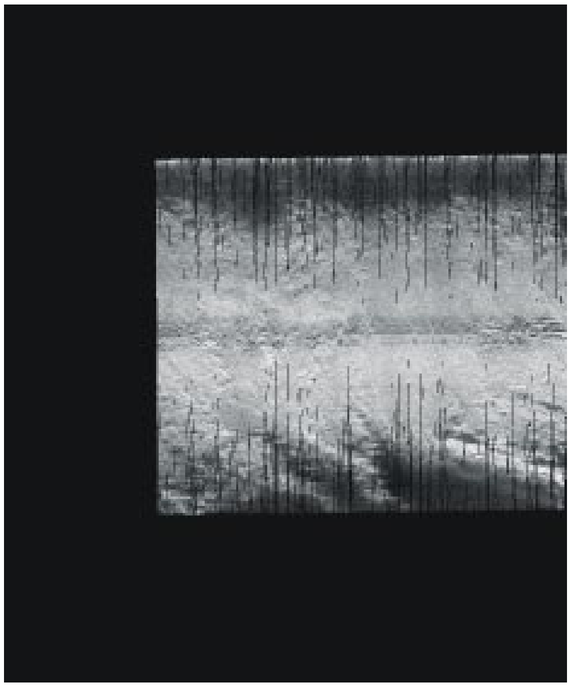

Figure 11 - Sonar swath mapped with heading averaged by 9 lines.

To continue the example, the preferred map selected is shown as Figure 11. This image provides relatively good placement of the scan lines without excessively smoothing the heading values. The steps and necessary files to process the swath to this point are summarized in Appendix B. To complete the map area, Pass-12 was processed to complete the easterly sonar swath through the map area. Pass-12 was processed utilizing the same programs and processing steps as applied to Pass-13. The two individual images were combined using program **qmos** and are displayed as Figure 12.

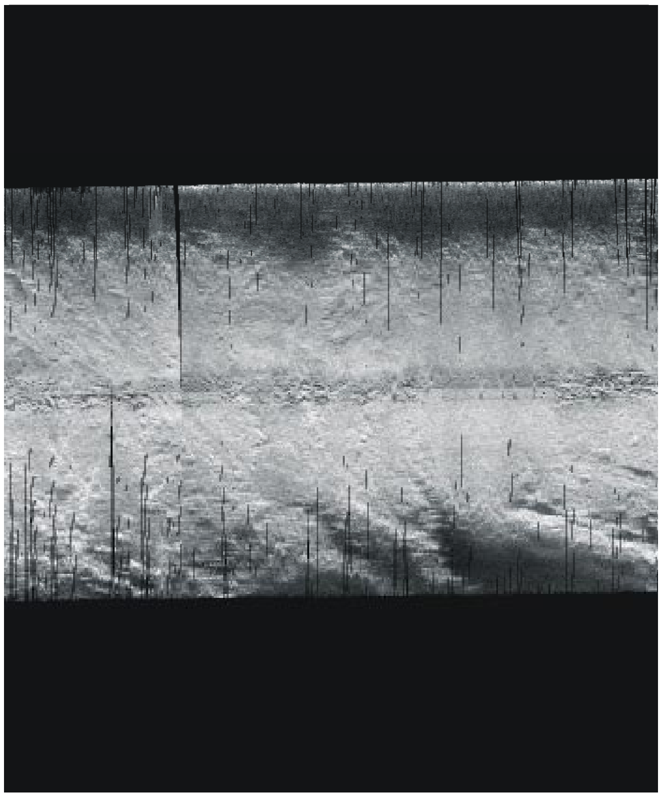

Figure 12 - Pass 12 and 13 processed and combined.

Map Completion

Depending on the resolution of the sidescan sonar image and the map scale selected, the user may need to fill in "holes" and "gaps" which may occur between pixels. The two combined images shown in Figure 12 illustrate this problem. The "holes" and "gaps" are generated as the individual pixel coordinates are placed within the map space. If the final map scale is equal to or smaller than the sonar resolution, "holes" will exist, as individual pixels may not fill a map area. Even when a map scale is greater than the sonar resolution, "holes" may exist, as the swaths may not produce even coverage. These "holes" may be filled in by one or more additional methods.

To complete the map area for this example, a series of mode filters were applied to the image next. A 3-by-3-mode filter and a 5-by-5-mode filter were applied to the image. Once the filters were completed, they were re-applied to the image to fill any remaining "holes". The mode is defined as the value that occurs most often within the boxcar neighborhood. These filters helped fill the gaps by replacing zero pixel values with the computed mode value for the specific neighborhood. However, continued application of these filters will not complete the image to our satisfaction since at times it is not possible to determine a mode value. For instance, in a 3-by-3 neighborhood 9 different values may occur or two unique values may occur 4 times. In this case no mode value can be determined and the zero pixel remains.

Finally, to complete the image a small (3-by-3) low-pass filter-replacing zero values was applied to the image. This filter simply replaces zero pixels with the average value computed from the neighborhood. This will fill the remaining zero pixels within the boxcar neighborhood and produce the final image shown in Figure 13.

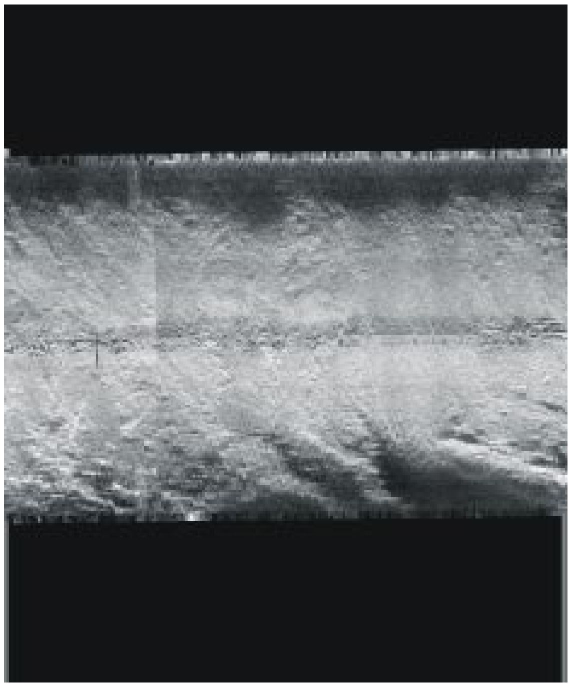
Figure 13 - East component filtered and completed.

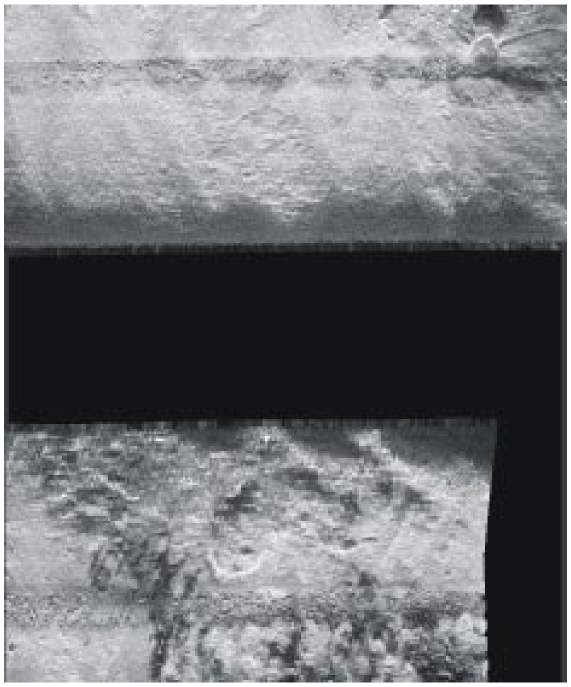
Figure 14 - West component filtered and completed.

To complete the map area, an image containing the west mapped sidescan sonar swaths was completed. The steps utilized in completing this portion of the map are summarized in Appendix C. The completed west portion of the map area is displayed as Figure14.

Stenciling

The east and west components must be combined to produce a final image with complete coverage of the map area. Before combining the images, both components required stenciling to remove unwanted areas from the images before combining them. Stenciling provides the user with the ability to select certain look angles of the data as well as the ability to cut unwanted data from the images.

Software and utilities to complete the stenciling procedure have not been implemented as part of the WHIPS software as of yet. To accomplish the stenciling procedure, several MIPS programs were utilized. To prepare for the stenciling procedure, the images were first transferred, via *ftp*, from the UNIX computer system to the Branch's MicroVAX-II computer system where the MIPS software resides. To prepare the images for transfer, the WHIPS netCDF images were translated to raw binary stream images by program **whips2raw**.

Once the individual images were stored on the MicroVAX system, they were re-assembled using the MIPS program **IMPORT**. The MIPS images were then displayed and stenciled to eliminate the unwanted portion of the images using available MIPS programs, which included **TVIN**, **TVSTCL**, **VEC2RAS** and **MASKIM**. The stenciled east and west components were then transferred back to the UNIX system. The stenciled images were prepared for transfer over the network by **EXPORT**. Once the images were stored on the UNIX system, they were converted back to the WHIPS netCDF format by program **raw2whips** and assembled together using program **qmos**. The stenciled, combined images are shown as Figure 15.

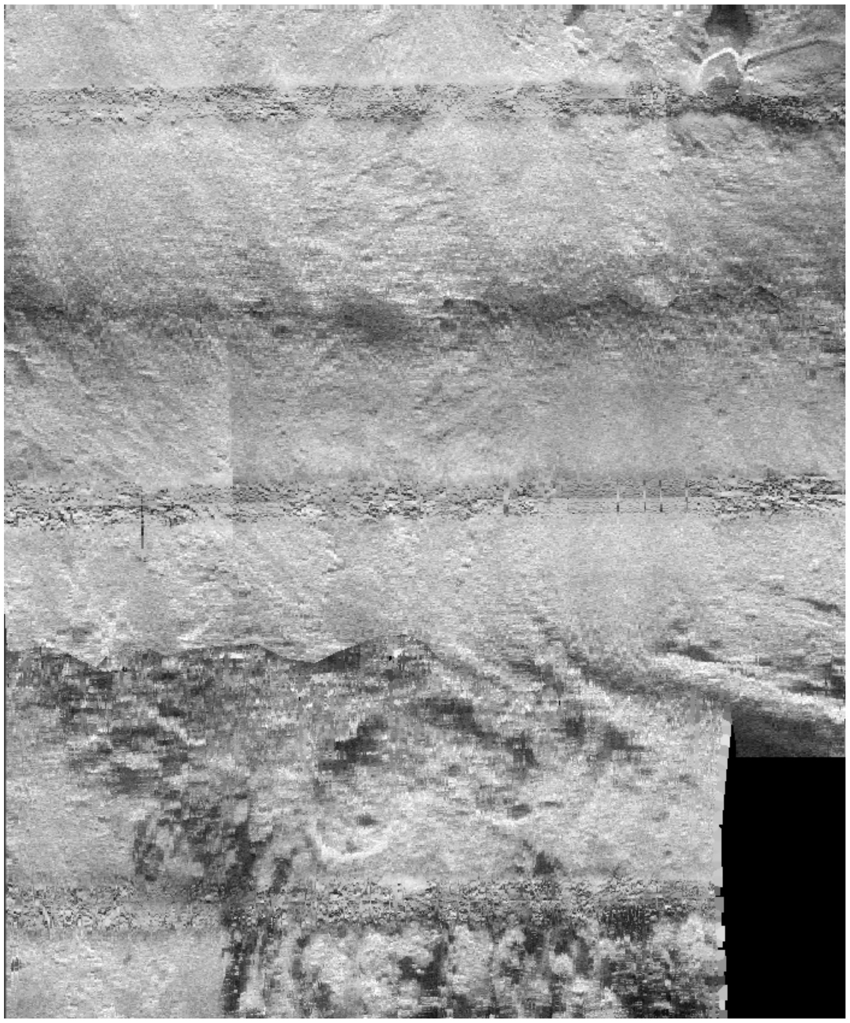

Figure 15 - East and west components stenciled and combined.

The data used in this example are a subset of a 2^{o}-quadrangle map with an area bounded by 34^{o} to 36^{o} south latitude and 116^{o} to 114^{o} west longitude. To complete the map area, all or part of twelve 6-hour GLORIA sidescan sonar passes were processed using the described techniques and the WHIPS processing is summarized in Appendix D. The 2^{o} quadrangle was completed at a one hundred-meter resolution. Computer processing time to complete this map took approximately 8 1/2 hours. Other than creating the necessary script files to complete the processing, no additional bookkeeping, math or interactive computer work was required by the user.

The pixel geographic computations were accomplished using program **gss_vel**. The individual passes were processed to produce three map components: east, west and south. Once the three map components were completed, they were transferred to the MicroVAX-II computer system for stenciling. The stenciled images were then transferred back to the UNIX computer system, assembled and smoothed with a 2-by-2 low pass filter (**lowpass2b2**) to produce the final map shown in Figure 16.

Summary

As stated earlier, the programs and techniques used in this digital mapping procedure are experimental and subject to change. However, the advantage of automated sonar mapping in eliminating the user-intensive labor required by previous digital mapping techniques is obvious. In addition, this new method provides an easy way to map sinewy tracklines that was not previously available.

A critical component of this digital mapping procedure is the ability to produce accurate fish navigation. As navigation of the sonar fish is improved, accuracy of mapping the sonar data will improve. Accurate navigation is required to properly place the sidescan sonar swaths and to have the underwater features properly line-up from swath to swath without the additional work required to warp and fit the images to one another.

In the future, it is hoped that more accurate mapping of the sonar data can be accomplished by mapping consecutive swaths simultaneously. Mapping of consecutive swaths together should provide the ability to interpolate and fill the "holes" using the data oriented to the scan and not the orientation of a square boxcar to the image. It is also hoped that future developments will include the ability to stencil the images on a UNIX based computer system with X-11 support.

Aknowledgements

I wish to thank Roger Larson and Robert Bird from the Graduate School of Oceanography, University of Rhode Island, and Roger Searle from the Department of Geological Sciences, University of Durham, United Kingdom, for the GLORIA sidescan sonar data used in developing and testing the mapping procedures described here.

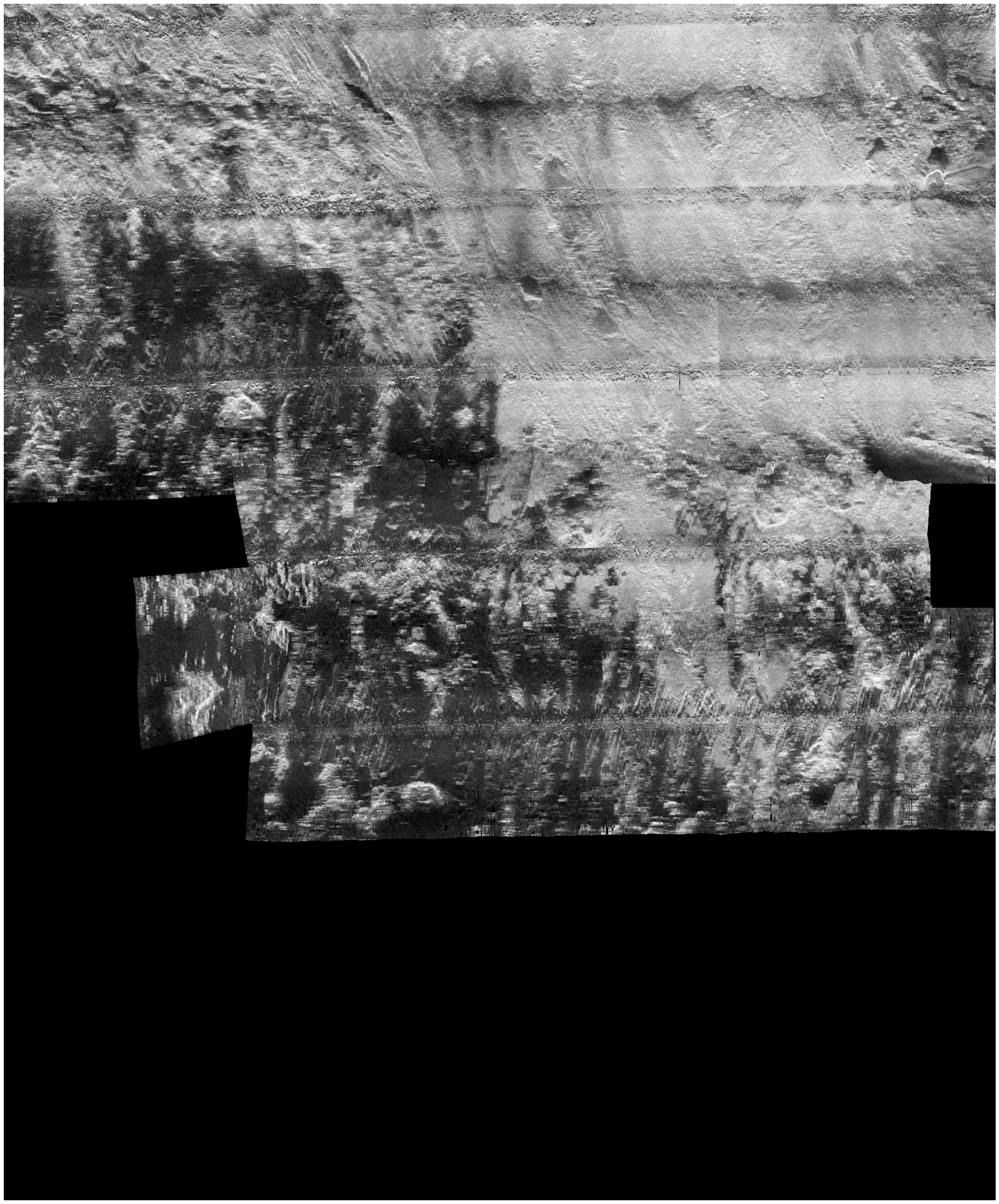

Figure 16 - Completed two-degree GLORIA quadrangle.

APPENDIX A

Woods Hole Image Processing System (WHIPS)
Program Documentation

WHIPS Programs

avg_heading	apply a running average to the heading values contained in a WHIPS netCDF side-scan sonar image header
avg_position	apply a running average to the fish positions contained in a WHIPS netCDF side-scan sonar image header
dk2dk	create a new image file by extracting a sub-area from an existing image
filter	applies a low-pass, high-pass, zero replacement or divide filter to an image
gss	computes geographic coordinates of side-scan sonar pixels
gss_vel	computes geographic coordinates of side-scan sonar pixels and applies the necessary velocity correction to the sonar swath
gssv	computes geographic coordinates of side-scan sonar pixels from a velocity corrected image
listhdr	list the contents of a side-scan sonar header in a WHIPS image
lowpass2b2	applies a 2-by-2 low-pass filter to an image
median3	applies a 3-by-3 median filter to an image
mode3	applies a 3-by-3 mode filter to an image.
mode5	applies a 5-by-5 mode filter to an image.
projss	place projected side-scan sonar data in a map space
qmos	Mosaics (overlays) the specified input file over the specified output file. The program will either overlay where the input file has priority over the existing output file, where the output file has priority over the input file, or average non-zero pixel values together from the input and existing output file.
quickview	displays an 8-bit WHIPS netCDF image in an X-11 window
raw2whips	convert a raw image data to a netCDF file for use with WHIPS
ssdk2dk	create a new image file by extracting a sub-area and accompanying header information from an existing WHIPS netCDF side-scan sonar image
sshead	computes a simple heading for a WHIPS netCDF side-scan sonar image
sumss	summarize the information contained in a side-scan sonar header for mapping
whips2raw	convert a WHIPS netCDF image file to a raw image file

NAME

avg_heading - apply a simple running average to the heading values contained in a WHIPS netCDF sidescan sonar image header

SYNOPSIS

avg_heading -i *input* **-o** *output* [**-l** *nl*] [**-H**]

DESCRIPTION

The **avg_heading** program allows the user to apply a simple running average over the heading values contained in a WHIPS netCDF sidescan sonar image header and to effectively smooth the values. The number of values averaged may be specified by the user by selecting the **-l** option along with the number of lines (*nl*) to average on the run-line. The specified number of lines must be 3 or greater and must be an odd integer value. If this option is not specified, the program defaults to averaging the heading values from 3 lines. Special processing takes place to handle the beginning and end of the sidescan sonar files. However, general processing is done by accumulating the heading values for an equal number of lines before and after the actual line being processed, averaging the value, and outputting the new heading value in the output file.

Special processing takes place to handle the beginning and ending lines contained in the files. It is not possible to have an equal number of lines before and after the line being processed unless processing is taking place in the center of the image, away from the beginning and ending lines. To compute the total to be averaged at the beginning of the file, the first line is weighted by an additional 1/2 the number of lines to be totaled. This means when nl = 3 and the first line is to be processed, the first heading value is computed as:

```
new_heading1 = (head1 + head1 + head2) / 3
```

In this simple case, processing the second line will produce an equally spaced number of heading values before and after the record being processed. Even spacing of the lines will continue until the last line is to be processed. To process the last line, as with the first line, the heading from the last line is double weighted.

As the number of lines to be processed is increased by the user, the weighting of the first line is increased. For example, when the user specifies nl=5 the new heading for the first record is computed as:

```
new_heading1 = (head1 + head1 + head1 + head2 + head3) / 5
```

In the example above using 5 lines and processing moves from the first to the second record, the first line is not weighted as heavily.

```
new_heading2 = (head1 + head1 + head2 + head3 + head4) / 5
```

As described earlier, moving away from the beginning of the file will eventually produce line processing where the heading values are weighted evenly before and after the specific line to be processed. The even processing will continue until processing nears the end-of-file. At that time, any necessary weighting of the last line in the image is similar to that done for the first line in the file.

Special processing was added to correct a problem that occurred when averaging north-heading values. During times when the heading value would alter between high values (approximately 325^o - 360^o) to low values (approximately 0^o - 35^o) averaging the low and high values produced accurate results though wild heading values. These shifts in the heading value typically occur in one of two scenarios: 1- a ship is attempting to follow a northerly course and it's heading waivers; or 2- the ship is executing a turn. Regardless of the cause, averaging the high and low values together produces undesirable results. In simplest terms, if the program were attempting to compute an average heading from two values of 359^o and 1^o, the mathematically correct result would be 180^o though not the desired heading value of 0^o. To correct for this problem and to compute an accurate heading value, additional checks were added to the program to test when the heading values were approaching 0^o from either direction. When the program has detected heading values both in the high and low range, it focuses on those headings that will be used to compute the average heading for a given line. As the total is computed, 360^o is added to the low values to bring the headings into a similar data range. The average heading is then computed from these values. If the average heading is greater than 360^o, 360^o is subtracted from the average and becomes the new heading value for the line being processed. As the processing continues to the next line, the program checks to see if both high and low heading values continue to exist. If so, the special computation applies once again. When the heading values being totaled fall entirely within a high or low range, the default method of computing the new heading value by a running average is resumed.

The special handling required to properly compute the heading values when mixing the high and low heading values will slow file processing in those files were the heading values shift frequently in the north direction. The heading computation and program execution is more efficient when processing can be accomplished without having to address the special handling for the north heading values or for a few cases such as a valid course turn in a file.

The following run-line options must be specified and can appear in any order.

-i *input*

Specifies the input file to be processed. The input file must 8-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop

-o *output*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-l *nl*

Specifies the *number of lines* (nl) for which the heading value is to be totaled and averaged.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The input file selected must contain the required sidescan sonar header information.

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

```
% avg_heading -i pass29.hdr -o pass29.avg_hd
```

The example below shows a possible execution of the program to compute the running average using seven heading values.

```
% avg_heading -i pass29.hdr -o pass29.avg_hd -l 7
```

SEE ALSO

avg_position(1)

WHIPS(5), whips_sonar(5)

"Digital Mapping of Sidescan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found	-	the selected input file does not contain the proper sidescan sonar header information

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

avg_position - apply a simple running average to the fish positions contained in a WHIPS netCDF sidescan sonar image header

SYNOPSIS

avg_position -i *input* **-o** *output* [**-l** *nl*] [**-H**]

DESCRIPTION

The **avg_position** program allows the user to apply a simple running average over the fish positions contained in a WHIPS netCDF sidescan sonar image header and to effectively smooth the latitude and longitude positions. The number of positions to be totaled and averaged may be specified by the user by selecting the **-l** option along with the number of lines (*nl*) to average on the run-line. The specified number of lines must be 3 or greater and must be an odd integer value. If this option is not specified, the program defaults to averaging the fish position for 3 lines. Special processing takes place to handle the first and last records of the sidescan sonar image file as well as the beginning and end of the sidescan sonar files. However, general processing is done by accumulating the latitude and longitude coordinates for an equal number of lines before and after the actual line being processed, averaging the position, and writing the new coordinates to the output file.

The first and last records are handled uniquely. The fish positions for those records are output unchanged. This was implemented to ensure that consecutive, separate file/line segments would match. In addition to the special processing for the first and last records of the sonar file, special processing takes place to handle the beginning and ending lines contained in the files. When *nl* (**-l**) is specified as 5 or greater, it is not possible to have an equal number of lines before and after the line being processed unless processing is taking place in the center of the image, away from the beginning and ending lines. To compute the totals to be averaged at the beginning of the file, the first line is weighted by an additional 1/2 the number of lines to be totaled. This means when nl = 5 and the first line is to be processed, the second positions are computed as:

```
new_lat = (lat1 + lat1 + lat2 + lat3 + lat4) / 5
new_lon = (lon1 + lon1 + lon2 + lon3 + lon4) / 5
```

In this simple case, processing the third line will produce an equally spaced number of heading values before and after the record being processed. Even spacing of the lines will continue until the end-of-file is approached. As processing nears the end-of-file, the fish positions from the last line are weighted accordingly.

The following run-line options must be specified and can appear in any order.

-i *input*

Specifies the input file to be processed. The input file must 8-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop.

-o *output*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-l *nl*

Specifies the number of lines (*nl*) for which the sidescan sonar fish positions are to be totaled and averaged.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The input file selected must contain the required sidescan sonar header information.

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

```
% avg_position -i pass29.mg2 -o pass29.avg_pos
```

The example below shows a possible execution of the program to compute the running average using seven values.

```
% avg_position -i pass29.mg2 -o pass29.avg_pos -l 7
```

SEE ALSO

avg_heading(1)

WHIPS(5), whips_sonar(5)

"Digital Mapping of Sidescan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found	- the selected input file does not contain the proper sidescan sonar header information

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

dk2dk - create a new WHIPS netCDF image file by extracting a sub-area from an existing image

SYNOPSIS

dk2dk -i *input_file* **-o** *output_file* **-a** *sl,ss,nl,ns* [**-l** *linc*] [**-s** *sinc*] [**-H**]

DESCRIPTION

The **dk2dk** (disk-to-disk) program will create a new image file by extracting a user-specified sub-area from an existing WHIPS netCDF image file. The sub-area is selected by specifying the **-a** option on the program run-line. The image sub-area is specified by the starting line (*sl*), starting sample (*ss*), number of lines (*nl*) and number of samples (*ns*) to be extracted. Image area variables are specified relative to the image origin (sl=1 and ss=1) and is the upper left corner of the matrix.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit.

-o *output_file*

Specifies the input file to be created.

-a *sl,ss,nl,ns*

Specifies the sub-area of the input file to be extracted. The sub-area is specified by entering the origin as the starting line (*sl*) and starting sample (*ss*) of the area to be extracted and the number of lines (*nl*) and the number of samples (*ns*) to be extracted.

When specifying the sub-area, the user must specify the *sl* and *ss* values. The program will compute default values for the *nl* and *ns* parameters as the remaining lines and samples in the input image from the user specified starting position.

Options: The following run-line commands are optional to the execution of the program.

-l *linc*

Specifies the line increment (*linc*) at which to output the lines from the input image sub-area. A value greater than one will reduce the input image sub-area. A value less than one will duplicate the input image lines and expand the image area selected. For example, a *line_increment* of 2 would result in every other line from the input sub-area being output. A *line_increment* of .5 would result in every line from the input image sub-area being duplicated for output. The default line increment value is 1.

-s *sinc*

Specifies the sample increment (*sinc*) at which to output the samples from the input image sub- area. This option is similar to the line increment **(-l**) option. The default sample increment value is 1.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

NOTES

This program should be used if the user wishes to extract and create a sub-area of the image portion of a WHIPS netCDF image file or a WHIPS netCDF side-scan sonar image. If the user wishes to extract a sub-area of the image portion of a WHIPS netCDF side-scan sonar image and retain the accompanying sonar header information for the image lines, the user should use program **ssdk2dk**.

If the user desires to extract only the image data from a WHIPS netCDF side-scan sonar image file and eliminate the sonar header, they may use program **dk2dk**.

EXAMPLE

The first example would extract the GLORIA side-scan sonar imagery from a file removing the header information.

```
% dk2dk -i gloria.slr -o gloria.sub -a 1,129
```

The second example would reduce the input file by transferring every other line and sample to the output file.

```
% dk2dk -i mickey.pic -o mickey.sub -a 1,1 -l 2 -s 2
```

SEE ALSO

ssdk2dk(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The 16 and 32-bit options of the program have not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

filter - apply a low-pass, high-pass, zero replacement or divide filter to an image

SYNOPSIS

filter -i *input* **-o** *output* **-b** *nl,ns* [**-l** | **-z** | **-L** | **-h** | **-d**] [options ... [**-H**]]

DESCRIPTION

The **filter** program allows the user to select one of five filter operations and apply it to a WHIPS image. The filters are applied to the image by a moving boxcar. The boxcar (**-b**) is a user specified sampling size that is two odd integer values that are not necessarily equal. The values represent the number of lines and samples (*nl,ns*) to be considered when accumulating the sums. Pixel values at the center of the boxcar are modified and are affected by the surrounding valid values. The boxcar totals are applied over the image starting at line_1/sample_1 to line_n/column_n. The boxcar is shifted left to right over the image line.

Valid data are specified by the user selecting the run-line option **-v**. The data values entered by the user will define the valid data range for processing. Values less than the minimum value or greater than the maximum value are considered non-valid and are not included in the operation of computing the boxcar totals. Specifying a valid data range may have other impacts on the selected filter. See the specific filter operation described on the following pages.

Each pixel surrounding the center of the boxcar is compared to the low and high range before the filtering operation is done. If the value of the original pixel falls outside of the valid range, it is not included in the boxcar sum and count. Boxcar totals represent the total of the valid pixel values and the number of valid points surrounding the center of the boxcar. The original unchanged pixel values are used to calculate the boxcar totals.

A minimum number of valid data points (**-m**) are required to be contained within the boxcar totals before the filtering process takes place. If there are fewer points in the boxcar than the user specified minimum, the resulting dn value will be set to zero on output for all filters. The default minimum value for this option is 1. The user may specify a value greater than or equal to 1 and less than or equal to the total boxcar size (*nl* * *ns*).

The minimum valid data points that must be contained in the filter may also be specified by selecting a fraction (**-f**) of the boxcar that must contain valid data points. If this option is selected, the minimum valid points is computed as:

```
((nl * ns) * fraction) + .5
```

For example, a 5-by-5 filter with a .5 fraction specified would then have to contain a minimum of 13 valid data points in the boxcar totals for the filter to be applied. This would have the same result as specifying *-m 13*, but is easier to specify for large filters. If a fraction greater than one is specified, it is reset to one. If the computed value is less than one, it will be set to one as default.

A coefficient value (**-c**) to expand the results of the data may also be specified. This is a real number that is used by all filters to expand the range of the results of the filter operations. The default value is 1.

The following run-line options must be specified and can appear in any order. Optional program keywords are listed and described following the detailed filter descriptions.

-i *input_file*

Specifies the input file to be processed. The input file may be an 8, 16 or 32-bit image.

-o *output_file*

Specifies the output file to be created.

-b *nl,ns*

Specifies the size of the boxcar by the number of lines (*nl*) and the number of samples (*ns*).

-l | **-z** | **-L** | **-h** | **-d**

Specifies the type of filter to perform. Valid filter selections are *low-pass* (**-l**), *low-pass filter with zero replacement* (**-z**), *low-pass filter changing only valid data* (**-L**), *high-pass* (**-h**) filter or *divide* (**-d**) filter.

The core to all the filtering operations is the computation of the *low-pass filter* (LPF). The LPF is a smoothing spatial filter that is good at reducing noise and removing the high-frequency content of an image. The LPF is computed by averaging the total valid pixel values in the pixel *neighborhood*. The *neighborhood* is the boxcar size specified by the user.

One consideration when developing a filtering program is how to deal with the edges of the image. As the boxcar begins and moves across the image, it is not properly centered and would not contain the proper sums. One possibility is to ignore the edges, thereby reducing the image content. The approach taken in this program is to compute the boxcar totals at the image edges by a folding/unfolding method. In essence, the program duplicates the neighboring pixels inside the edges, centered on the boxcar. As the boxcar moves away from the edge and across to the center of the image, these duplicated values are removed and replaced by pixel values from the center of the image. As the boxcar meets the right and bottom edge of the image, the pixel values inside the edge are slowly duplicated back as if to fold the edge of the image back over itself. The variables for the individual filter are defined as:

(i,j)	-	The image coordinate of the pixel being computed. For line 29 and sample 12, the image coordinate would be (29,12).
P(i,j)	-	The original pixel value of the input image at coordinate i,j.
S(i,j)	-	The sum of the points over the boxcar centered at i,j.
N(i,j)	-	The number of valid points within the boxcar surrounding the pixel being processed.

The LPF computation is defined below and is followed by a description of the individual filters.

```
LPF(i,j) = S(i,j)/N(i,j)
```

The ***low-pass filter*** (LPF) is a smoothing spatial filter. Only input image pixels values that fall within the user specified valid data range and processing boxcar size are totaled and averaged to produce the LPF component. If the minimum valid points (**-m** or **-f**) for the boxcar is not satisfied, the output pixel is set equal to zero. If the minimum valid points have been satisfied, the LPF is applied regardless of the original pixel value. In other words, this option will modify all input pixel values. The *low-pass filter* is computed using the following equation:

```
LPF(i,j) = S(i,j)/N(i,j)
  X(i,j) = COEF*LPF(i,j)
```

If the sum of the boxcar or the number of valid points contained in the boxcar is zero, the value returned by the LPF computation will be zero.

The ***zero replacement filter*** (LPFZ) is a low-pass filter with one minor difference. The difference is that during the filtering process only input pixel values equal to zero are modified. This allows the user an option to fill in "holes" based on the value of surrounding pixels. If the user does not specify a valid range for computing the boxcar totals, the minimum valid value is automatically set to 1 to eliminate zeros from the boxcar totals. The minimum valid value **MUST** be greater than zero.

The ***low-pass filter*** changing valid data only (LPFV) is also similar to the low-pass filter described above in the initial boxcar computations. The major difference is this option will only modify input pixel values that fall between the valid minimum and maximum values specified by the user. If an input pixel value is less than the specified minimum value, the output pixel is set to 0 for all filters. If the input pixel value is greater than the maximum value, the output pixel value is set to the maximum allowed value for the bit type. For an 8-bit image this value is 255, 16-bit is 32767 and 32-bit is set equal to FLT_MAX as defined in the file */usr/include/limits.h*.

The ***high-pass filter*** (HPF) enhances the high-frequency details of an image. Edge enhancement of an image is also possible with the application of a *high-pass filter*. The HPF is computed using the *low-pass filter* described above. The boxcar values are computed by totaling the input image pixel values that fall within the valid data range. The *high-pass filter* (HPF) is computed as follows:

```
HPF(i,j) = NORM*(1-ADDBACK) + P(i,j)*COEF*(1+ADDBACK) -
                    LPF(i,j)*COEF
```

Before computing the *high-pass filter*, the original pixel value is compared against the valid data range. The *high-pass filter* is applied to the image coordinate only if the original pixel value falls within the valid data range. If the original value is less than the specified minimum valid value, the output pixel is set equal to zero. If the input pixel value is greater than the maximum value, the output pixel value is set to the maximum allowed value for the bit type. For an 8-bit image this value is 255, 16-bit is 32767 and 32-bit is set equal to FLT_MAX as defined in the file */usr/include/limits.h*.

The ***divide filter*** (DIV), when utilized by specifying a valid data range, will produce a binary image (0 or 255 values) similar to a mask image. When the LPF component of the filter is greater than the maximum valid value specified by the user, the input pixel will be output as 255. If the LPF component is computed as less than the minimum valid value specified by the user, the output pixel is zero. The resulting image would then be a "mask" of the valid values. The *divide filter* is computed as follows:

```
DIV(i,j) = COEF*(P(i,j)/LPF(i,j)) - NORM
```

The *divide filter* is applied to the image coordinate only if the original input pixel value falls within the valid data range. If the original value is less than the minimum value specified by the user, the output pixel value is set equal to zero. If the input pixel value is greater than the maximum value, the output pixel value is set to the maximum allowed value for the bit type. For an 8- bit image this value is 255, 16-bit is 32767 and 32-bit is set equal to FLT_MAX as defined in the file */usr/include/limits.h*.

It is recommended that the user apply the resulting DIV "mask" with caution. In some cases, the "mask" outlines are not continuous. When the "mask" is applied to the image, the discontinuous lines can result in portions of the image, which are to be preserved, being dropped during the masking operation.

Options: The following run-line commands are optional to the execution of the program.

-a *addback*

Specifies the add back (*addback*) value that is used by **the *high-pass*** and ***divide filters*** only.

-v *minval,maxval*

Specifies the minimum and maximum values (*minval,maxval*) to be used to define the valid data range.

-c *coef*

Specifies the coefficient (*coef*) to be used during the filtering process to expand the results of the data during filtering. The value specified may be a floating-point value and may be any value the user desires. The default coefficient value is 1.

-n *norm*

Specifies the normalization value (*norm*) used in the *high-pass* and *divide filter* computations. The default normalization value for 8-bit image data is 127. For 16 or 32-bit data, the default value is 0.

-m *mingood*

Specifies the minimum number of good points (*mingood*) that must be contained within the boxcar before the filter is applied. The default value is 1. This option may be superseded by specifying -f.

-f *fraction_good*

Specifies the fraction of the boxcar (*fraction_good*) that must contain valid data points before the filter is applied to the image coordinate. Specifying this option would override the **-m** option. The *fraction_good* is specified as the percentage of the boxcar that must contain valid data points before a filter can be applied.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

SEE ALSO

/usr/include/limits.h

lowpass2b2(1), median3(1), mode3(1), mode5(1), ssfilter(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The 8 and 16-bit options have been extensively tested. The 32-bit option has not been fully tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

gss - compute geographic coordinates of sidescan sonar pixels from a non-velocity processed image

SYNOPSIS

gss -i *input* **-r** *resolution* >std_out [**-l** *sl,el*] [**-h** *heading_adjustment*] [**-s** *nsamps* | **-d** *distance*] [**-H]**

DESCRIPTION

The **gss** program will compute the geographic coordinates of each pixel in the scan line of a WHIPS netCDF sidescan sonar image. The coordinate of each pixel is computed using the heading value contained in the sidescan sonar attribute field, the sonar position associated with the scan line, and the user supplied resolution value. The resolution value (**-r**) is the pixel resolution, in meters, of the input image file.

By default, the program will process every image line and sample contained in the input file. The user may select which lines of the image to process by specifying the **-l** option. When this option is specified, the user must specify the starting line and ending line to process within the image. The user may also optionally specify the number of samples (*nsamps*) or the distance (*distance*) from nadir to be processed within each scan by selecting either the **-s** or **-d** program option. Each sonar image line is processed from nadir to the specified far range with the port side being processed first, followed by the starboard side. As the geographic coordinates of the pixels are computed, the positions, along with the associated pixel value, are output to the *std_out* device. The coordinates are recorded as decimal degree values and are output in latitude/longitude order.

The fish heading value contained in the sidescan sonar header fish attributes field may be adjusted by the user by selecting the **-h** option. When this option is selected, the user must also specify a heading adjustment value (*heading_adjustment*) to be added to the fish heading value before the pixel coordinates are computed. This option, along with the ability to select a portion of the input image to be processed, provides a simple way to "tweak" a segment of a sonar line when the navigation data may not accurately reflect the track of the sonar fish. Hopefully this option will assist in adjusting the sonar swaths and provide a simple way to lineup adjacent swaths.

As stated above, the program output is to *std_out*. This option was implemented to allow for piping of the data through a series of steps to facilitate a stream processing to place the pixels directly into a defined map space. If the input pixel value is equal to zero, the program will NOT output the coordinate and pixel information. This was implemented to help reduce the amount of pixels output and later processed.

The maximum and minimum geographic coordinates computed by the program are output to the program print file.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the *input file* to be processed. The input file must be 8 or 16-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the required information, the program will display the message **ncvarid: variable "date" not found** and processing will stop.

-r *resolution*

Specifies the *pixel resolution*, in meters, of the input image. This value is used to compute the position of the pixel values from the known position at nadir.

Options: The following run-line commands are optional to the execution of the program.

-l *sl,el*

Specifies a sub-area of the input file to be processed. The sub-area is specified by entering the starting line (*sl*) and the ending line (*el*) of the sonar image file to be processed. This option, along with **-s** or **-d**, may be selected to specify a sub-area of the sonar image to process.

-h *heading_adjustment*

Specifies a user-supplied value to be added to the fish heading value before computing the pixel coordinates. This option may be used to adjust the fish heading and swath orientation if the user suspects the fish coordinates do not accurately reflect the direction of the fish.

-s *nsamps*

Specifies the number of samples (*nsamps*), port and starboard, to be processed from nadir. For example, specifying *nsamps* as 225 would flag the program to process 225 samples from either side of nadir resulting in a total of 450 samples being output per scan. This option, along with **-l**, may be selected to specify a sub-area of the sonar image to process. If the user specifies *nsamps* as greater than the number of samples actually contained in each side of the image scan, the value will be set to half the actual number of samples contained in a line of the image file.

The user may optionally select the **-d** option to specify the distance from nadir to be processed.

-d *distance*

Specifies the distance (*distance*), in meters, from nadir for which to process the image samples. The user may specify this option, along with **-l**, to select a sub-area of the sonar image to process.

The user may optionally select the **-s** option to specify the number of samples from nadir to be processed.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The program only accepts 8 or 16-bit image files.

The output file to be created must not currently exist.

The **-s** or **-d** options are exclusive. Only one may be selected.

NOTES

This program differs from **gssv**. Program **gssv** assumes a velocity stretched image. **Gss** assumes an image that has not been velocity stretched and the program does not accommodate the duplicate lines produced by velocity. If a velocity stretched image is input to program **gss**, the swaths with duplicate positions are simply reprocessed over the previous line and would be redundant processing.

EXAMPLE

The example below shows a simple application of the program to compute the coordinate values for a sidescan sonar image processed at .5 meters.

```
% gss -i bos13.wco -r .5 >bos13_cord.dat
```

The example below shows the program usage to select a specified number of lines (275 lines) and width (375 samples per side).

```
% gss -i gloria.head -r 50 -l 201,475 -s 375 >pass74b.dat
```

SEE ALSO

gss_vel(1), gssv(1), mapit(1), projss(1)

avg_heading(1), avg_position(1), sshead(1), sumss(1)

WHIPS(5), whips_sonar(5)

"Digital Processing of Side-Scan Sonar data with the Woods Hole Image Processing System Software": U. S. Geological Survey Open-File Report 92-204, 11p.

"Digital Mapping of Side-Scan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

The 16-bit option has not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

gss_vel - compute geographic coordinates of sidescan sonar pixels and corrects the sonar imagery for changes in ship's velocity

SYNOPSIS

gss_vel **-i** *input* **-r** *resolution* [[*options* ...] [**-E** *ellips*] ... [**-H**]] >std_out

DESCRIPTION

The **gss_vel** program will compute the geographic coordinates of each pixel in the scan line of a WHIPS netCDF sidescan sonar image. <u>The input file MUST be a sidescan sonar image that has NOT been velocity stretched.</u> **Gss_vel** combines the functions of programs **gss** and **velocity** because it will apply the velocity corrections as each swath is processed. The velocity correction is applied by the program by first computing the distance traveled between consecutive swaths and then the number of times an individual line should be duplicated to fill the image to the position of the next swath. Velocity stretching of a sidescan sonar image is accomplished by duplicating individual swath lines in an attempt to produce an image with the same across and along-track resolution. The velocity correction may not be necessary for mid or low-range sidescan sonar which have `*square*' pixels.

The program begins by calculating the number of times the individual line to be processed should be duplicated so it will meet the following line and reduce the gap in the map space between the consecutive lines. The duplication factor is calculated first by determining the distance traveled between a pair of consecutive lines contained in the sidescan sonar image. The distance traveled between the consecutive swaths is determined by comparing the position coordinates contained in the sidescan sonar header. The user may select the spheroid (**-E**) to be used in the distance computation from one of five available ellipsoids. With the distance traveled between the consecutive lines known, the program then computes the number of times a line should be duplicated to fill the image space up to the placing of the next swath based on the sidescan sonar image resolution (**-r**) specified by the user.

As each individual swath is processed, the geographic coordinate of each pixel is computed using the heading value contained in the sidescan sonar attribute field, the sonar position associated with the scan line, and the user supplied resolution value. The resolution value (**-r**) is the pixel resolution, in meters, of the input image file. As each individual line is duplicated, a new nadir position is calculated based on the sonar resolution (**-r**) and the heading contained in the sidescan sonar header. Geographic coordinates are then computed for the pixels contained in the duplicate lines to allow the proper placement of the swaths and to reduce the gaps that can occur between swaths.

The program computes the geographic coordinates of the pixels and, if necessary, the new positions of the duplicate lines by a simple distance computation utilizing one of five earth ellipsoids. By default, **gss_vel** uses the Clarke 1866 spheroid in these computations. The user may select alternate earth ellipsoids by specifying the **-E** option on the program run-line. The available ellipsoids are 1) Clarke 1866 (default); 2) International 1909 (Hayford); 3) Geodetic Reference System 1980; 4) WGS 1984; and 5) WGS 1972.

By default, the program will process every image line and sample contained in the input file. The user may select which lines of the image to process by specifying the **-l** option. When this option is specified, the user must specify the starting line and ending line to process within the image. The user may also optionally specify the number of samples (*nsamps*) or the distance (distance) from nadir to be processed within each scan by selecting either the **-s** or -**d** program option. Each sonar image line is processed from nadir to the specified far range with the port side being processed first, followed by the starboard side. As the geographic coordinates of the pixels are computed, the position, along with the associated pixel value, is output to the std_out device. The coordinates are recorded as decimal degree values and are output in latitude/longitude order.

The fish_heading value contained in the sidescan sonar header fish attributes field may be adjusted by the user by selecting the **-h** option. When this option is selected, the user must also specify a heading adjustment value (heading_adjustment) to be added to the fish heading value before the pixel coordinates are computed. This option, along with the ability to select a portion of the input image to be processed, provides a simple way to "tweak" a segment of a sonar line when the navigation data may not accurately reflect the track of the sonar fish. Hopefully this option will assist in adjusting the sonar swaths and provide a simple way to lineup adjacent swaths.

As stated above, the program output is to std_out. This option was implemented to allow for piping of the data through a series of steps to facilitate a stream processing to place the pixels directly into a defined map space. If the input pixel value is equal to zero, the program will NOT output the coordinate and pixel information. This was implemented to help reduce the amount of pixels output and later processed.

The maximum and minimum geographic coordinates computed by the program are output to the program print file.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the *input file* to be processed. The input file must be 8 or 16-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the required information, the program will display the message **ncvarid: variable "date" not found** and processing will stop.

-r *resolution*

Specifies the *pixel resolution*, in meters, of the input image. This value is used to compute the position of the pixel values from the known position at nadir.

Options: The following run-line commands are optional to the execution of the program.

-l *sl,el*

Specifies a sub-area of the input file to be processed. The sub-area is specified by entering the starting line (*sl*) and the ending line (*el*) of the sonar image file to be processed. This option, along with **-s** or **-d**, may be selected to specify a sub-area of the sonar image to process.

-h *heading_adjustment*

Specifies a user supplied value to be added to the fish heading value before computing the pixel coordinates. This option may be used to adjust the fish heading and swath orientation if the user suspects the fish coordinates do not accurately reflect the direction of the fish.

-s *nsamps*

Specifies the number of samples (*nsamps*), port and starboard, to be processed from nadir. For example specifying *nsamps* as 225 would flag the program to process 225 samples from either side of nadir resulting in a total of 450 samples being output per scan. This option, along with **-l**, may be selected to specify a sub-area of the sonar image to process. If the user specifies *nsamps* as greater than the number of samples actually contained in each side of the image scan, the value will be set to half the actual number of samples contained in a line of the image file.

The user may optionally select the **-d** option to specify the distance from nadir to be processed.

-d *distance*

Specifies the distance (*distance*), in meters, from nadir for which to process the image samples. The user may specify this option, along with **-l**, to select a sub-area of the sonar image to process.

The user may optionally select the **-s** option to specify the number of samples from nadir to be processed.

-E *ellips*

Specifies the earth ellipsoid (*ellips*) to be used in the computation of the pixel and line coordinates. The *ellips* value may be specified as one of the following values:

1 = Clarke 1866
2 = International 1909 (Hayford)
3 = Geodetic Reference System 1980
4 = WGS 1984
5 = WGS 1972

-R

Flags the program to round-up the number of times a sidescan sonar swath is to be duplicated to accommodate the velocity stretching. For example, when the program computes a line duplication factor of 3.5, the program, by default, will output the line only three times. If the **-R** option has been selected on the program run-line, line duplication factor would be rounded-up and the specific line being processed would be duplicated four times.

This option may help reduce the gaps that can exist between adjacent lines as they are placed in the map space. It will also increase the processing time and will affect the resultant image by increasing the blockiness of the sonar map image.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The program only accepts 8 or 16-bit image files.

The output file to be created must not currently exist.

The **-s** or **-d** options are exclusive. Only one may be selected.

NOTES

This program combines the functions of programs **gss** and **velocity**. In addition to the pixel coordinate computation which program **gss** provides, this program will correct the image for changes in ship's velocity by duplicating the line being processed up to the position of the following line.

This program also differs slightly from **gssv**. Program **gssv** requires the sidescan sonar input file to have been previously processed through program velocity.

EXAMPLE

The example below shows a simple application of the program to compute the coordinate values for a sidescan sonar image processed at .5 meters using the default Clarke 1866 spheroid.

```
% gss_vel -i bos13.wco -r .5 >bos13_cord.dat
```

The example below shows the program usage to select a specified number of lines (275 lines) and width (375 samples per side). The **-R** option has been selected to help reduce the gaps between the placement of the consecutive sidescan sonar swaths. The **-E** option has also be specified to utilized the International 1909 spheroid.

```
% gss_vel -i gloria.head -r 50 -l 201,475 -s 375 -E 2 -R >pass74b.dat
```

SEE ALSO

gss(1), gssv(1), mapit(1), projss(1)

avg_heading(1), avg_position(1), sshead(1), sumss(1)

WHIPS(5), whips_sonar(5)

"Digital Processing of Sidescan Sonar data with the Woods Hole Image Processing System Software": U. S. Geological Survey Open-File Report 92-204, 11p.

"Digital Mapping of Sidescan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

The 16-bit option has not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

gssv - compute geographic coordinates of sidescan sonar pixels from a velocity corrected image

SYNOPSIS

gssv -i *input* **-r** *resolution* >std_out [**-l** *sl,el*] [**-h** *heading_adjustment*] [**-s** *nsamps* | **-d** *distance*] [**-H]**

DESCRIPTION

The **gssv** program will compute the geographic coordinates of each pixel in the scan line of a WHIPS netCDF sidescan sonar image. The input file should be a sonar image that has been velocity corrected. Velocity correction of a sidescan sonar image is generally accomplished by duplicating swath lines to produce an image with the same across and along-track resolution. Since the lines are duplicated by velocity, two or more consecutive image lines will contain the same latitude and longitude coordinates. **Gssv** will compare the latitude and longitude coordinates of the current line being processed from the coordinates of the last line processed. If the position values are the same, **gssv** will adjust the position by computing a new pair of latitude and longitude coordinates based on the image resolution (**-r**) and the heading contained in the sidescan sonar header. The new positions are computed for the duplicate lines to allow the proper placement of the swaths and to reduce the gaps that can occur between the swaths.

As each swath is processed, the coordinate of each pixel is computed using the heading value contained in the sidescan sonar attribute field, the sonar position associated with the scan line, and the user supplied resolution value. The resolution value (**-r**) is the pixel resolution, in meters, of the input image file.

By default, the program will process every image line and sample contained in the input file. The user may select which lines of the image to process by specifying the **-l** option. When this option is specified, the user must specify the starting line and ending line to process within the image. The user may also optionally specify the number of samples (*nsamps*) or the distance (*distance*) from nadir to be processed within each scan by selecting either the **-s** or **-d** program option. Each sonar image line is processed from nadir to the specified far range with the port side being processed first, followed by the starboard side. As the geographic coordinates of the pixels are computed, the positions, along with the associated pixel value, are output to the *std_out* device. The coordinates are recorded as decimal degree values and are output in latitude/longitude order.

The fish heading value contained in the sidescan sonar header fish attributes field may be adjusted by the user by selecting the **-h** option. When this option is selected, the user must also specify a heading adjustment value (*heading_adjustment*) to be added to the fish heading value before the pixel coordinates are computed. This option, along with the ability to select a portion of the input image to be processed, provides a simple way to "tweak" a segment of a sonar line when the navigation data may not accurately reflect the track of the sonar fish. Hopefully this option will assist in adjusting the sonar swaths and provide a simple way to lineup adjacent swaths.

As stated above, the program output is to *std_out*. This option was implemented to allow for piping of the data through a series of steps to facilitate a stream processing to place the pixels directly into a defined map space. If the input pixel value is equal to zero, the program will NOT output the coordinate and pixel information. This was implemented to help reduce the amount of pixels output and later processed.

The maximum and minimum geographic coordinates computed by the program are output to the program print file.

The following run-line options must be specified and can occur in any order.

-i *input_file*

specifies the *input file* to be processed. The input file must be 8 or 16-bit.

The input file must contain the necessary sidescan sonar header information. If the user selects a file that does not contain the sidescan sonar header information, the program will display the message **`ncvarid: variable "date" not found`** and processing will stop.

-r *resolution*

specifies the pixel resolution (*resolution*), in meters, of the input image. This value is used to compute the position of the pixel values from the known position at nadir.

Options: The following run-line commands are optional to the execution of the program.

-l *sl,el*

specifies a sub-area of the input file to be processed. The sub-area is specified by entering the starting line (*sl*) and the ending line (*el*) of the sonar image file to be processed. This option, along with **-s** or **-d**, may be selected to specify a sub-area of the sonar image to process.

-h *heading_adjustment*

specifies a user supplied value to be added to the fish heading value before computing the pixel coordinates. This option may be used to adjust the fish heading and swath orientation if the user suspects the fish coordinates do not accurately reflect the direction of the fish.

-s *nsamps*

specifies the number of samples (*nsamps*), port and starboard, to be processed from nadir. For example specifying *nsamps* as 225 would flag the program to process 225 samples from either side of nadir resulting in a total of 450 samples being output per scan. This option, along with **-l**, may be selected to specify a sub-area of the sonar image to process. If the user specifies *nsamps* as greater than the number of samples actually contained in each side of the image scan, the value will be set to half the actual number of samples contained in a line of the image file.

The user may optionally select the **-d** option to specify the distance from nadir to be processed.

-d *distance*

specifies the distance (*distance*), in meters, from nadir for which to process the image samples. The user may specify this option, along with **-l**, to select a sub-area of the sonar image to process.

The user may optionally select the **-s** option to specify the number of samples from nadir to be processed.

-H

displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The program only accepts 8 or 16-bit image files.

The output file to be created must not currently exist.

The **-s** or **-d** options are exclusive. Only one may be selected.

NOTES

This program differs from **gss** since this program assumes a velocity stretched image and can accommodate the duplicate lines and fish position values.

This program also differs slightly from **gss_vel**. Program **gss_vel** combines the functions of programs **gss** and **velocity**.

EXAMPLE

The example below shows a simple application of the program to compute the coordinate values for a sidescan sonar image processed at .5 meters.

```
% gssv -i bos13.wco -r .5 >bos13_cord.dat
```

The example below shows the program usage to select a specified number of lines (275 lines) and width (375 samples per side).

```
% gssv -i gloria.head -r 50 -l 201,475 -s 375 >pass74b.dat
```

SEE ALSO

gss(1), gss_vel(1), mapit(1), projss(1)

avg_heading(1), avg_position(1), sshead(1), sumss(1)

WHIPS(5), whips_sonar(5)

"Digital Processing of Sidescan Sonar data with the Woods Hole Image Processing System Software": U. S. Geological Survey Open-File Report 92-204, 11p.

"Digital Mapping of Sidescan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

The 16-bit option has not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

listhdr - list the contents of a sidescan sonar header in a WHIPS netCDF sidescan sonar image

SYNOPSIS

listhdr -i *input* [**-l** *linc*] [**-j**] [**-S**] [**-P** *print file*] [**-H**]

DESCRIPTION

The **listhdr** program will list the header information from a sidescan sonar image. By default, the line number, year, month, day, hour, minute, seconds, latitude, longitude, heading and fish altitude information is displayed. If a field contained in the sidescan sonar header is printed that does not contain valid information, *inf* will be output in the appropriate field. The *inf* reflects that the field has been filled with system dependent infinity value for that data type.

The information is displayed on the user's terminal but may be output to a file by re-direction of *std_out* or selecting the **-P** option on the run-line.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file must be 8-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop.

Options: The following run-line commands are optional to the execution of the program.

-l *linc*

Specifies the line increment (*linc*) at which to output the lines from the input file. The default linc value is 1.

-j

Flags the program to replace the day and month information with the day of year value in the output.

-S

Flags the program that, in addition to the default information, output the sonar pitch roll and yaw information.

-P *print_file*

Specifies the print file to re-direct the output information to.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The input file selected must contain the required sidescan sonar header information.

EXAMPLE

The example will output the sidescan sonar header from file *bos13.slr* and output the information to *bos13.hdr*. The first ten records of the header file are the listed below.

```
% listhdr -i bos13.slr -P bos13.hdr

% head bos13.hdr
 1: 1985 10   9  5   0   0.000   25.392599   -84.828300  0.0  3356.00
 2: 1985 10   9  5   0  30.000   25.391350   -84.827751  0.0  3356.00
 3: 1985 10   9  5   1   0.000   25.390100   -84.827202  0.0  3356.00
 4: 1985 10   9  5   1  30.000   25.388849   -84.826653  0.0  3356.00
 5: 1985 10   9  5   2   0.000   25.387600   -84.826103  0.0  3356.00
 6: 1985 10   9  5   2  30.000   25.386351   -84.825546  0.0  3356.00
 7: 1985 10   9  5   3   0.000   25.385099   -84.824997  0.0  3356.00
 8: 1985 10   9  5   3  30.000   25.383850   -84.824463  0.0  3356.00
 9: 1985 10   9  5   4   0.000   25.382601   -84.823898  0.0  3356.00
10: 1985 10   9  5   4  30.000   25.381350   -84.823372  0.0  3356.00
```

SEE ALSO

replacehdr(1), strphdr(1)

WHIPS(5), whips_sonar(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

lowpass2b2 - applies a 2 by 2 low-pass filter to an image

SYNOPSIS

lowpass2b2 -i *input* **-o** *output* [**-H**]

DESCRIPTION

Program **lowpass2b2** applies a small low-pass smoothing filter to an image. The low-pass filter consists of a 2-by-2 moving boxcar. The image is smoothed by the filter with the boxcar applied starting at the upper left origin of the image and moving across each line of input data and down through the input image. With the exception of the first line and ending sample of each line, the filter is applied by computing the average of 4 neighboring pixels with the result being stored in the lower left pixel. The filter is applied to the entire image.

An example of how the program computes the pixel averages is as follows:

input

line 1:	11	22
line 2:	**33**	44

output

11	22
28	44

The first input image line is a special case and is smoothed by applying a 1-by-2 low-pass filter.

input

line 1:	11	22	33	44
line 2:	33	44	55	66

output

17	28	39	..
28	39	50	..

The last sample of each line is also a special case and is processed as a 2-by-1 low-pass filter.

input

line 1:	11	22	44
line 2:	33	44	66
line 3:	55	66	88
line 4:	77	88	99

output

17	33	44
28	38	55
50	66	77
72	85	94

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file must be 8-bit.

-o *output_file*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The program accepts only 8-bit image files.

The output file to be created must not currently exist.

EXAMPLE

```
% lowpass2b2 -i gloria.vel -o gloria.2b2
```

SEE ALSO

filter(1), median3(1), mode3(1), mode5(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

median3 - apply a 3-by-3 median filter to an image

SYNOPSIS

median3 -i *input* **-o** *output* [**-z**] [**-H**]

DESCRIPTION

The **median3** program allows the user to apply a 3-by-3 median filter to a WHIPS image. Median3 will modify the value centered on a 3-by-3 boxcar with the median value computed from the neighborhood distribution. The neighborhood, *n*, consists of 9 values (3x3), and the median value is computed as i,j = 5 = (n+1)/2 after the data has been arranged in increasing order.

The user may apply this program to fill zero values contained within an image by selecting the **-z** option on the run-line. When this option is selected, only those pixel values from the input file equal to zero are modified. However, all input pixel values, including those equal to zero, are used to compute the median value.

Median3 is most suitable for data that has a skewed distribution. However, the value obtained for the median may not be representative if the individual items do not tend to cluster at the center of the distribution.

Special processing takes place to handle the first and last lines of the image file. Adjacent lines are weighted to allow for unfolding to take place during the processing. When computing the 3-by-3 median of the first image line from the input file, the second line is read twice and used in the computation. To process the last line contained in the image line, the next to last line is read twice and used for the computation.

In addition to the special line processing, the program applies a similar overlapping procedure to the samples at the beginning and ending of each line. For the first and last samples contained in the image lines, the neighboring pixels are doubly weighted to allow for the foldover computations.

The following run-line options must be specified and can appear in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit image.

-o *output_file*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-z

Flags the program to apply the filter only when the center value of the neighborhood is zero. This will allow the user to apply the 3-by-3 median filter as a zero only replacement filter.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

```
% median3 -i map.cdf -o map.med3
```

NOTES

Though a histogram method could be employed to calculate the median value for 8-bit data, the histogram method would be more difficult to implement for 16 and 32-bit data. Therefore median3 is set-up to sort (via the UNIX library function *qsort*) the data values and can be quickly applied to 16 and 32-bit data as well as 8-bit.

SEE ALSO

lowpass2b2(1), filter(1), mode3(1), mode5(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The 16 and 32-bit options have not been thoroughly tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

mode3 - apply a 3-by-3 mode filter to an image

SYNOPSIS

mode3 -i *input* **-o** *output* [**-z**] [**-Z**] [**-H**]

DESCRIPTION

The **mode3** program allows the user to apply a 3-by-3 mode filter to a WHIPS image. **Mode3** will modify the value centered on a 3-by-3 boxcar with the mode value computed from the neighborhood distribution. The neighborhood, *n*, consists of 9 values (3x3), and the mode value is the value which occurs most often within the neighborhood. In a neighborhood of 9 values it is possible that no mode value can be determined. For example, 9 different values may occur within the neighborhood or 2 different values may occur 4 times. When no mode value can be computed for the neighborhood, the input pixel value for that location is output unchanged.

The user may apply this program to replace zero values contained within an image by selecting the **-z** option on the run-line. When this option is selected, only those pixel values from the input file equal to zero are modified. However, all input pixel values, including that equal to zero, are used to compute the mode value.

In addition to the zero replacement option (**-z**), the user may select not to include the zero values from the image when computing the mode value by selecting the **-Z** program option. When this option is selected, the zero values for the neighborhood are not included when totaling the occurrences of the unique values for the neighborhood. The **-z** and **-Z** options are not mutually exclusive and may be selected individually or together during a single execution of the program. Selection of these program options is at the user's discretion depending on the results he or she wishes to achieve.

Special processing takes place to handle the first and last lines of the image file. Adjacent lines are weighted to allow for unfolding to take place during the processing. When computing the 3-by-3 mode of the first image line from the input file, the second line is read twice and used in the computation. To process the last line contained in the image line, the next to last line is read twice and used for the computation.

In addition to the special line processing, the program applies a similar overlapping procedure to the samples at the beginning and ending of each line. For the first and last samples contained in the image lines, the neighboring pixels are doubly weighted to allow for the foldover computations.

The following run-line options must be specified and can appear in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit image.

-o *output_file*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-z

Flags the program to apply the filter only when the center value of the neighborhood is zero. This will allow the user to apply the 3-by-3 mode filter as a zero only replacement filter.

-Z

Flags program not to include zero values when computing the mode value. This option can be helpful when trying to apply the mode as a zero replacement filter and the mode in some neighborhoods are zero.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

```
% mode3 -i map.cdf -o map.mode3
```

The example below shows a possible execution of the program to replace zero values within the image that excludes zero values from the mode computation.

```
% mode3 -i map.cdf -o map.mode3zr -z -Z
```

NOTES

Though a histogram method could be employed to calculate the mode value for 8-bit data, that method would be more difficult to implement for 16 and 32-bit data. Therefore **mode3** is set-up to sort (via the UNIX library function *qsort*) the data values and then count the number of times a unique value occurs within the neighborhood. The program will then compute the mode value and this technique can be applied to 16 and 32-bit data as well as 8-bit.

SEE ALSO

lowpass2b2(1), filter(1), median3(1), mode5(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The 16 and 32-bit options have not been thoroughly tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

mode5 - apply a 5-by-5 mode filter to an image

SYNOPSIS

mode5 **-i** *input* **-o** *output* [**-z**] [**-Z**] [**-H**]

DESCRIPTION

The **mode5** program allows the user to apply a 5-by-5-mode filter to a WHIPS netCDF image. **Mode5** will modify the value centered on a 5-by-5 boxcar with the mode value computed from the neighborhood distribution. The neighborhood, *n*, consists of 25 values (5x5), and the mode value is the value which occurs most often within the neighborhood. In a neighborhood of 25 values it is possible that no mode value can be determined. For example, 25 different values may occur within the neighborhood or 5 different values may occur 5 times. When no mode value can be computed for the neighborhood, the input pixel value for that location is output unchanged.

The user may apply this program to fill zero values contained within an image by selecting the **-z** option on the run-line. When this option is selected, only those pixel values from the input file equal to zero are modified. However, all input pixel values, including that equal to zero, are used to compute the mode value.

In addition to the zero replacement option (**-z**), the user may select not to include the zero values from the image when computing the mode value by selecting the **-Z** program option. When this option is selected, the zero values for the neighborhood are not included when totaling the occurrences of the unique values for the neighborhood. The **-z** and **-Z** options are not mutually exclusive and may be selected individually or together during a single execution of the program. Selection of these program options is at the user's discretion depending on the results he or she wishes to achieve.

Special processing takes place to handle the first two and last two lines of the image file. Adjacent lines are weighted to allow for unfolding to take place during the processing. When computing the 5-by-5 mode of the first image line from the input file, the second and third lines are read twice and used in the computation. To process the second line in the input file, the third and fourth lines are read once and the first line is read twice to allow for the foldover processing. Similarly, the last two lines contained in the image are handled with unique weighting done to the adjacent lines.

In addition to the special line processing, the program applies a similar overlapping procedure to the samples at the beginning and ending of each line. For the first and last two samples contained in the image lines, the neighboring pixels are doubly weighted to allow for the foldover computations.

The following run-line options must be specified and can appear in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit image.

-o *output_file*

Specifies the output file to be created.

Options: The following run-line commands are optional to the execution of the program.

-z

Flags the program to apply the filter only when the center value of the neighborhood is zero. This will allow the user to apply the 5-by-5-mode filter as a zero only replacement filter.

-Z

Flags program not to include zero values when computing the mode value. This option can be helpful when trying to apply the mode as a zero replacement filter and the mode in some neighborhoods are zero.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

```
% mode5 -i map.cdf -o map.mode5
```

The example below shows a possible execution of the program to replace zero values within the image while excluding any zero values from the mode computation.

```
% mode5 -i map.cdf -o map.mode5zr -z -Z
```

NOTES

Though a histogram method could be employed to calculate the mode value for 8-bit data, that method would be more difficult to implement for 16 and 32-bit data. Therefore mode5 is set-up to sort (via the UNIX library function *qsort*) the data values and then count the number of times a unique value occurs within the neighborhood. The program will then compute the mode value and this technique can be applied to 16 and 32-bit data as well as 8-bit.

SEE ALSO

lowpass2b2(1), filter(1), median3(1), mode3(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The 16 and 32-bit options have not been thoroughly tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

projss - place projected side-scan sonar data in a map/image space

SYNOPSIS

projss -o *output* < std_in [**-H**]

DESCRIPTION

Projss will take the sidescan sonar data output from one of the sidescan sonar geographic coordinate computation programs (**gss**, **gss_vel** or **gssv**) and **proj** and create a WHIPS netCDF image which represents a map of the sonar data for a specific area, projection and scale. The first four input records must contain the map area bounds, in meters, for the user selected desired map area. Subsequent data records must contain the sidescan sonar pixel coordinate and their associated pixel value. Each pixel coordinate contained in the sidescan sonar data must have been previously converted to geographic coordinates by program **gss**, **gss_vel** or **gssv**, and, further, converted to meter coordinates for the selected map area by program **proj**. The map coordinates must be in y x (latitude longitude) order and the coordinate pairs must be integer values.

Program input is through *std_in*. As stated above, the actual input to program **projss** is the sidescan sonar pixel coordinates that have been converted to meter values based on a specific map projection and scale. The first four records of the input file **MUST BE** the map corner coordinates, in meters, for the map projection and scale. The map corner coordinates must be specified in the following order:

upper left
upper right
lower right
lower left

For example, if the user desires to create a 2^o map at a scale of 1:100 for an area bounded by 35^o to 37^o latitude and -75^o to -77^o longitude for a simple Mercator map, the first four records of the input file would contain the following information:

```
44132 -85717
44132 -83491
41391 -83491
41391 -85717
```

Program **projss** will begin by reading the first four pairs of coordinates from the input file. Once the program has obtained this information it will calculate the size of the image (number of lines and number of samples). When the size of the WHIPS netCDF image has been computed, **projss** will then create the image file on disk and fills the image with blank lines before beginning to place the pixel values in the image.

The remainder of data in the input file must be the pixel coordinates and pixel values which **projss** will place in the appropriate map space. The information which follows must be the pixel coordinates, in meters, along with the 8-bit pixel value. Program **projss** accepts and calculates the location (the actual image line and sample coordinate) within the map space for the input pixel dn values. The program then places, on a pixel-by-pixel basis, the sidescan sonar dn values. It is important to note that the pixel placement is done on a pixel-by-pixel basis. When multiple dn values are computed for the same location, **projss** will always place the last pixel input in the output location regardless of the coordinates previous content.

Program **proj** is essential in creating the final map product. The user should be familiar with its usage and various options. Some **proj** options must be specified to create the proper data output for program **projss**.

The following run-line options must be specified and can occur in any order.

-o *output_file*

Specifies the output file to be created. The output file will be an 8-bit WHIPS image.

Options: The following run-line commands are optional to the execution of the program.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

The input meter coordinates must be recorded as integer values.

EXAMPLE

The example below shows a simple execution of the program. The first four pairs of coordinates contained in the file, *project.dat*, must be the projected map area bounds.

```
% projss -o map2.cdf <projec.dat
```

The following is a typical example using program **gss** and **proj** as filters to complete the processing and mapping of a sidescan sonar image. The input image to program **gss**, *gloria.wco*, contains a pixel resolution of 50 meters. The data is to be mapped at 100 meter resolution to an Albers Equal Area projection using standard parallels and a central longitude of 85^o. The file, *bounds.dat*, must contain the map corner coordinates and is show below.

```
% gss -i gloria.wco -r 50 | \
  proj +proj=aea +lon_0=-85 +lat_1=29.5 +lat_2=45.5 -r -s \
  -m 1:100 -f `%.0f' bounds.dat - | projss -o map2.cdf

% more bounds.dat
25.5 -85    # upper left corner
25.5 -84    # upper right corner
24.5 -84    # lower right corner
24.5 -85    # lower left corner
```

SEE ALSO

gss(1), gss_vel(1), gssv(1), proj(1), mapit(1)

filter(1), median3(1), mode3(1), mode5(1)

WHIPS(5)

User's Manual for MAPGEN (UNIX version): a method of transforming digital cartographic data to a map: U. S. Geological Survey Open-File Report 85-706, 134 p.

Cartographic Projection Procedures for the UNIX Environment - A User's Manual: U. S. Geological Survey Open-File Report 90-284, 62p.

"Digital Mapping of Sidescan Sonar Data with the Woods Hole Image Processing System Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

NOTES

There are three major drawbacks to the pixel-by-pixel method employed in this program/mapping procedure and they are discussed below. Note of the drawbacks are made in an attempt to warn the user of possible problems which could affect the quality of their final mosaic when using the **gss | proj | projss** processing scenario, and to discuss possible future processing alternatives.

The first and second drawbacks are due to the random order of the input pixels being placed. This random input results in a last-in placement for coordinates that may have two or more valid pixels. Since the program can not place in order the multiple values for a given coordinate, the program must deal with each pixel as a unique value and is placed in its coordinate location regardless of any pixel values which may have been previously placed. Even a simple comparison for placement of pixels in non-zero locations and averaging of the pixels would be restrictive due to the overhead placed in computing. Secondly, if the input pixels could be placed in sorted order from the image origin down to the last line and last sample to be placed, processing could be accomplished by "building" a complete image line. This would result in fewer writes to the output file since a complete line of image data could be written rather than the many single writes which must be done for each pixel input. Fewer writes would hopefully speed up processing. However, when processing over a half million pixels at a time (a modest file at best), speed may be best accomplished by increased workstation performance. The sorted order of the data would also be beneficial by allowing the program to compare multiple pixel values for a specific coordinate and select the final output pixel value by averaging or selecting either the minimum or maximum value.

The third drawback is the "holes" which may develop as the pixels are placed in the map space. These "holes" may be best eliminated, or reduced in number, by the user carefully selecting the resolution of the sonar data being processed with the goal in mind of the final resolution of the map to be created. If the user selects to process their sonar strips at .5 meters and their final map resolution is 1 meter, fewer "holes" will develop from the pixel placement. An obvious drawback is the volume of pixels that will have to be processed resulting in longer execution time. The user must consider the trade-off of processing such enormous volumes of data or what "holes" may be created in the mapping process. These "holes" must be filled in some manner. The preferred method would be some form of interpolation based on the orientation of the scan line and the appropriate neighboring pixels. To accomplish such a task, the sonar data must be processed on a swath-by-swath basis with two consecutive swath's being processed at a time. Unfortunately, the ability to handle the data on a pixel-by-pixel basis is the only option currently available. This results in the "holes" or gaps that may by created to be filled in some manner after the sonar map has been created. Possible options are either a low-pass filter replacing zero values, a 3-by-3 mode or median filter, a 5-by-5 median filter or some combination of the former. The "filtering" options are less than perfect since the zero pixels are filled relative to the orientation of the boxcar passing over the image, not the orientation of the scan lines. It may be with careful selection and application of the filters, the image will not be degraded significantly.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

The program was written to assist in testing the necessary programs and steps to utilize program **proj** while prototyping the sidescan sonar line-by-line mapping. Program **projss** was not intended to be a final product program and does not contain sufficient user input checks. Therefore, the program may appear to run successfully and yet produce wild results based on the user's incorrect input.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

qmos - quick mosaic of two WHIPS netCDF images

SYNOPSIS

qmos **-i** *input* **-o** *output* [**-I** | **-O** | **-A**] [**-H**]

DESCRIPTION

Program **qmos** will mosaic (overlay) the specified input file (**-i**) over the specified output (**-o**) file. The program will either overlay where the input file has priority over the existing output file (**-I**, program default), where the output file has priority (**-O**) over the input file, or average (**-A**) non-zero pixel values together from the input and existing output file.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit.

-o *output_file*

Specifies the output file to be modified. The output file must currently exist.

Options: The following run-line commands are optional to the execution of the program.

-I

Flags the program that non-zero pixel values in the *input file* are to take precedence. This is the program default. When the input file takes precedence, the non-zero value of a specific pixel coordinate will be replaced by the non-zero input pixel value for that location.

-O

Flags the program that non-zero pixel values in the *output file* are to take precedence. When the output file takes precedence, the non-zero value of a specific pixel coordinate will not be replaced by the non-zero input pixel value for that location.

-A

Flags the program to *average* non-zero pixel values from the input and output files. When averaging of the files is selected, the non-zero pixel values for a specific image coordinate are averaged together and the output pixel value is replaced with the new value.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The selected input and output file must be the same size.

Unlike the majority of WHIPS programs where the output file must not exist and is created by the application program, the output file to be modified must currently exist for this application to execute successfully.

EXAMPLE

```
% qmos -o map.comp -i l26.map
```

SEE ALSO

WHIPS(5)

"Digital Mapping of SideScan Sonar Data with the Woods Hole Image Processing system Software": U.S. Geological Survey Open-File Report 92-536, 90p.

"MAPIT: An improved method for mapping digital sidescan sonar data using the Woods Hole Image Processing System (WHIPS) Software": U.S. Geological Survey Open-File Report 96-281, 73p.

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

quickview - displays an 8-bit WHIPS netCDF image in an X-11 window

SYNOPSIS

quickview -i *input* [**-a** *sl,ss,nl,ns*] [**-c** *comp*] **-H**]

DESCRIPTION

Program **quickview** will produce a simple and quick display of a WHIPS netCDF image in an X-11 window. The image, if necessary, will be automatically compressed to fit the image on the screen. If the image must be compressed, it will be compressed to present a proportional image and the compression ratio, as 1:xx.x will be displayed above the image. If the image is being displayed at full resolution, the word uncompressed will be displayed above the image.

The 24-bit version of the program currently available on the Data General systems will display the line and sample coordinate relative to the actual image and the pixel value within a message box located below the image. This option has not been implemented for the 8-bit display version of the program for the SUN or ULTRIX computer systems.

The following run-line options must be specified and can appear in any order.

-i *input*

Specifies the WHIPS netCDF file to be processed. The input file may only be 8-bit.

Options: The following run-line commands are optional to the execution of the program.

-a *sl,ss,nl,ns*

Specifies the sub-area of the input file to be displayed. The sub-area is specified by entering the origin as the starting line (*sl*) and starting sample (*ss*) of the area to be extracted and the number of lines (*nl*) and the number of samples (*ns*) to be extracted. When specifying the sub-area, the user must specify the *sl* and *ss* values. The program will compute default values for the *nl* and *ns* parameters as the remaining lines and samples in the input image from the user specified starting position.

If the **-c** option is not specified, the program will automatically compute an image compression factor, if necessary, to proportionally display the full sub-area selected.

-c *comp*

Specifies a user selected compression factor (*comp*) at which to display the image. The program must be able to display the complete image or selected sub-area using the specified compression factor.

The selection of a user specified compression factor may override the compression factor automatically computed by the program. By default, the program computes a compression factor to be used to proportionally display the full image or user selected image sub-area. If the user specified compression factor is greater than or equal to the compression factor computed by the program, the user specified compression factor will be used to display the image. Otherwise, the program computed compression factor will be used. The program will always maintain the ability to proportionally display the image, as well as displaying the entire image or sub-area selected.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

Quickview can only display an 8-bit image. The display device must be capable of displaying an 8-bit image (i.e. the display device must be 8 or 24-bit deep).

NOTES

When utilizing this program on 8-bit displays, some "flashing" is done when the proper color table is installed. When the mouse is placed within the image window, the X-11 server installs ab 8-bit, 256 grey-level color table. This color table replaces any previous color table and may result in any previously displayed X-11 windows outside the image window to become invisible. Moving the cursor outside the **quickview** image window will result in the X-11 server restoring the default color table and restoring any previous X-11 windows that were displayed. When the program is exited, the default color table is restored to the X-11 server.

EXAMPLE

```
% quickview -i gloria.cdf
```

SEE ALSO

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

raw2whips - convert a raw image data to a netCDF file for use with WHIPS

SYNOPSIS

raw2whips -i *input* **-o** *output* **-l** *nl* **-s** *ns* [**-b** *bittyp*] [**-H**]

DESCRIPTION

Program **raw2whips** converts a raw binary stream image file to a netCDF file for use with WHIPS.

The following run-line options must be specified and can appear in any order.

-i *input_file*

Specifies the binary stream input file to be converted. The input file may be 8, 16 or 32-bit.

-o *output_file*

Specifies the netCDF output file to be created.

-l *nl*

Specifies the number of lines (*nl*) or rows contained in the image data.

-s *ns*

Specifies the number of samples (*ns*) or columns contained in the image data.

Options: The following run-line commands are optional to the execution of the program.

-b *bittyp*

Specifies the bit type (*bittyp*) of the data being processed. Valid selections are 8, 16 or 32. The default *bittyp* is 8.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

Currently, the program assumes the image data is in the proper bit and byte order for the selected bit type. No swabbing takes place.

The output file to be created must not currently exist.

EXAMPLE

```
% raw2whips -i mickey.raw -o mickey.cdf -l 480 -s 472
```

SEE ALSO

whips2raw(1), WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

There appears to be a small problem when transferring files with less than 512 samples from MIPS unless the number of samples are evenly divisible by four. If this is a bug or merely a quirk, is not entirely known. One way around the problem is to either enlarge or truncate the file on MIPS so the number of samples is evenly divisible by four. Then create the raw image with the MIPS program **EXPORT**, transfer the file and convert it with **raw2whips**. Files with image lines greater than 512 don't seem to be a problem.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

ssdk2dk - create a new sidescan sonar image file by extracting a sub-area and accompanying sidescan sonar header from an existing WHIPS netCDF sidescan sonar image

SYNOPSIS

ssdk2dk **-i** *input_file* **-o** *output_file* **-a** *sl,ss,nl,ns* [**-l** *linc*] [**-s** *sinc*] [**-H**]

DESCRIPTION

The **ssdk2dk** (sidescan sonar disk-to-disk) program will create a new image file by extracting a user-specified sub-area from an existing WHIPS netCDF sidescan sonar image file. Along with the image portion of the data, the accompanying sidescan sonar headers for the selected lines are carried over to the output file. The image sub-area is selected by specifying the **-a** option on the program run-line. The sub-area is specified by the starting line (*sl*), starting sample (*ss*), number of lines (*nl*) and number of samples (*ns*) to be extracted. Image area variables are specified relative to the image origin (sl=1 and ss=1) and is the upper left corner of the matrix.

The program can also be used to reduce or enlarge an input file by specifying the **-l** and/or **-s** run-line options.

The following run-line options must be specified and can occur in any order.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file may be 8, 16 or 32-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop.

-o *output_file*

Specifies the input file to be created.

-a *sl,ss,nl,ns*

Specifies the sub-area of the input file to be extracted. The sub-area is specified by entering the origin as the starting line (*sl*) and starting sample (*ss*) of the area to be extracted and the number of lines (*nl*) and the number of samples (*ns*) to be extracted.

When specifying the sub-area, the user must specify the *sl* and *ss* values. The program will compute default values for the *nl* and *ns* parameters as the remaining lines and samples in the input image from the user specified starting position.

Options: The following run-line commands are optional to the execution of the program.

-l *linc*

Specifies the line increment (*linc*) at which to output the lines from the input image sub-area. A value greater than one will reduce the input image sub-area. A value less than one will duplicate the input image lines and expands the image area selected. For example, a *line_increment* of 2 would result in every other line from the input sub-area being output. A *line_increment* of .5 would result in every line from the input image sub-area being duplicated for output. The default line increment value is 1.

-s *sinc*

Specifies the sample increment (*sinc*) at which to output the samples from the input image sub- area. This option is similar to the line increment **(-l**) option. The default sample increment value is 1.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The output file to be created must not currently exist.

NOTES

This program is similar to program **dk2dk** in that both programs can be used to extract a sub-area from the image portion of a WHIPS netCDF image file. In addition to processing the image portion of a sidescan sonar file, program **ssdk2dk** will process the accompanying sonar header information. This program should be used if the user wishes to extract a sub-area of a WHIPS netCDF sidescan sonar image and retain the accompanying sonar header information for the image lines. If the user desire to extract only the image data from a WHIPS netCDF sidescan sonar image file and eliminate the sonar header, they may use program **dk2dk**.

EXAMPLE

The first example would extract all the samples contained in the first 300 lines and associated sidescan sonar header of the selected image file.

```
% ssdk2dk -i gloria.slr -o gloria.sub -a 1,1,300
```

The second example would reduce the input file by transferring every other line and sample, starting at the image origin, to the output file.

```
% ssdk2dk -i gloria.slr -o gloria.sub -a 1,1 -l 2 -s 2
```

SEE ALSO

dk2dk(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

The 16 and 32-bit options of the program have not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

sshead - computes simple heading for a WHIPS netCDF sidescan sonar image

SYNOPSIS

sshead -i *input* **-o** *output* [**-h** *heading_adjustment*] [**-H**]

DESCRIPTION

Program **sshead** reads a WHIPS netCDF sidescan sonar file and computes the heading value from swath to swath. The new heading value is then recorded in the sonar-attribute field in the output file. The heading is computed from the sonar position values contained in the header information of a WHIPS sidescan sonar image.

The new heading value is computed from consecutive coordinate pairs. Since the first valid computed heading value is for the second swath of the sonar image, a valid heading cannot be computed for the first swath. Therefore, the new heading of the first record is assumed to be the same as the heading for the second swath and is set accordingly.

The user may specify a value to be added to the computed heading value by selecting the program option **-h**. This allows the user a simple way to adjust the heading value computed from the navigation if they feel the navigation does not accurately reflect the track of the sonar fish.

The following run-line options must be specified and can occur in any order.

-i *input_file*

Specifies the input file to be processed. The input file must be 8-bit.

The input file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop.

-o *output_file*

Specifies the output file to be created. The output file contains the sidescan sonar header information from the input file with the new heading values.

Options: **The following run-line commands are optional to the execution of the program.**

-h *heading_adjustment*

Specifies a heading adjustment value to be added to the computed heading value.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

Program currently accepts only 8-bit image files. The input file must contain the sidescan sonar header information.

The output file to be created must not currently exist.

EXAMPLE

```
% sshead -i gloria.wcox -o gloria.head
```

SEE ALSO

WHIPS(5), whips_sonar(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

atan2: DOMAIN error - may be displayed during programming. Processing continues until the end-of-file is encountered and the file is useable. In the past, the source of this error has been traced to improper fish position values in the header. Check the sidescan sonar header for consecutive records with the same position.

Version 1.3 of the program was modified to compare consecutive positions, and when duplicates are found, output the last good heading value computed. This additional check will add some time to the processing but will hopefully eliminate the error problem. This change, however, does not eliminate the need to clean-up the navigation data set as much as possible to eliminate the large number of consecutive duplicate positions that can be found in something like the EG&G data sets. It is also highly recommended that the data set have good unique positions at the beginning of the file so the heading can be computed accurately to start.

AUTHOR/MAINTENANCE

Valerie Paskevich
U. S. Geological Survey
Coastal and Marine Geology Program
384 Woods Hole Rd.
Woods Hole, MA 02543

NAME

sumss - summarize the information contained in a sidescan sonar header for mapping

SYNOPSIS

sumss -i *input* [**-P** *print file*] [**-h** *heading*] [**-H**]

DESCRIPTION

Program **sumss** will summarize the contents of the sidescan sonar header from the user specified WHIPS netCDF sonar image. The summary information contains the date, time, position and fish heading for the first and last record in the file. That information is followed by a report of line numbers where the specified heading value (**-h**) is exceeded. This information may be null if the specified heading value is not exceeded. The final information is a report on the minimum and maximum nadir coordinates.

The headings are checked using the first heading value contained in the sonar header. As the fish heading values are read from the file, the two values are compared. When the difference in the two heading values is equal to or greater than the specified heading (**-h**), the line number and fish heading value that exceed the specified tolerance are output to the *stdout* device and program print file. The heading value last read becomes the heading value used for future comparisons. When the heading change again exceeds the user-specified value, the program repeats the procedure for flagging the line and swapping the heading value for future comparisons.

The program provides a quick way to check the area covered by the file and any possible course changes contained in the file in anticipation of mapping the image. The summary information is displayed on the user's terminal (*std_out*) but the *std_out* information may be output to a file by re-direction of *std_out* or selecting the **-P** option on the run line. The information reported to the *std_out* file is also output to the program print file.

The following run-line options must be specified and can appear in any order.

-i *input*

Specifies the input file to be processed. The input file must be 8-bit.

The selected file must contain the sidescan sonar header information. If the user selects a file that does not contain the proper information, the program will display the message **ncvarid: variable "date" not found** and the processing will stop.

Options: The following run-line commands are optional to the execution of the program.

-h *heading*

Specifies a user selected heading value. The program will then check the changes in fish heading (*heading*) and will report the line numbers in the file when a change in heading exceeds the specified value. The default *heading* value is 15.

-P *print_file*

Specifies the print file to re-direct the output information to.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

The input file selected must contain the required sidescan sonar header information.

EXAMPLE

The example below shows a simple execution of the program and resulting output.

```
% sumss -i 26.ss

 First record:
         1982  1 30  8 0  0.000   ( 30)
           26.991600  -88.106300  --  heading: 153.5


 Last record (720):
         1982  1 30 11 59  0.666   ( 30)
           26.509069  -88.022507  --  heading: 171.2


 Heading change to 168.5 at line 557

 Nadir limits:
          Latitude  =  26.991600 26.509069
          Longitude = -88.106300 -88.020897
```

The following is an example requesting the program to report when the sonar heading changes exceed 5^{o}.

```
% sumss -i 26.ss -h 5

 First record:
         1982  1 30  8 0  0.000   ( 30)
           26.991600  -88.106300  --  heading: 153.5


 Last record (720):
         1982  1 30 11 59  0.666   ( 30)
           26.509069  -88.022507  --  heading: 171.2


 Heading change to 158.5 at line 294
 Heading change to 163.6 at line 391
 Heading change to 168.6 at line 559

 Nadir limits:
          Latitude  =  26.991600 26.509069
          Longitude = -88.106300 -88.020897
```

SEE ALSO

listhdr(1)

WHIPS(5), whips_sonar(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

ncvarid: variable "date" not found - the selected input file does not contain the proper sidescan sonar header information

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

NAME

whips2raw - convert a WHIPS netCDF image file to a raw, binary image file

SYNOPSIS

whips2raw -i *input* **-o** *output* [**-H**]

DESCRIPTION

Program **whips2raw** converts a WHIPS netCDF image file to a raw binary stream image file. The output of the program may then be converted to other image formats or imported to other software packages.

The following run-line options must be specified and can appear in any order.

-i *input*

Specifies the netCDF file to be processed.

-o *output*

Specifies the binary stream input file to be converted. The input file may be 8, 16 or 32-bit.

Options: The following run-line commands are optional to the execution of the program.

-H

Displays the usage help information for the program. If this option is selected, the program ignores any other run-line options specified.

RESTRICTIONS

None known.

EXAMPLE

```
% whips2raw -i mickey.cdf -o mickey.raw
```

SEE ALSO

raw2whips(1), whips2pgm(1)

WHIPS(5)

DIAGNOSTICS

The program exit status is 0 if no errors are encountered during processing and the program completes processing.

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA

APPENDIX B

Below is a list of the files used to complete the digital mapping procedure for Pass-13. Included is a listing of the **proj** script file and the text file containing the map boundaries. Those files, *utm-proj* and *bounds.dat,* have been underlined to help identify their proper reference place.

```
% more utm-proj
proj +proj=utm +lon_0=-111 -r -s -m 1:100 -f `%.0f' bounds.dat -

% more bounds.dat
-34.25 -112.75
-34.25 -112
-35 -112
-35 -112.75

% avg_heading -i pass13.head -o pass13.avg_head -l 9
% gss_vel -i pass13.avg_head -r 45 -s 490 -l 1,320 | utm-proj | projss
-o pass13.map
```

APPENDIX C

Below is a summary of the steps used to complete the processing of the east and west components of the side-scan sonar mapping example. Processing of the component images to complete the east images was accomplished by selecting the lines to be processed from the individual passes. Both the east and west components were completed by running a series of mode and low-pass filters to fill the "holes" contained in the images.

```
% avg_heading -i pass12.head -o pass12.avg_head -l 9
% avg_heading -i pass13.head -o pass13.avg_head -l 9
% gss_vel -i pass12.avg_head -r 45 -s 490 -l 570,720 | utm-proj | projss -o
pass12.map
% gss_vel -i pass13.avg_head -r 45 -s 490 -l 1,320 | utm-proj | projss -o
pass13.map
% cp pass12.map east.map
% qmos -i pass13.map -o east.map
% mode3 -i east.map -o east.mode3 -z -Z
% mode5 -i east.mode3 -o east.mode5 -z -Z
% mode3 -i east.mode5 -o east.mode3a -z -Z
% mode5 -i east.mode3a -o east.mode5a -z -Z
% filter -i east.mode5a -o east.lpfz -z -b 3,3
```

The processing summary below shows the steps utilized to complete the west component of the side-scan sonar mapping example. To map the data for these areas, all the lines in the sonar swaths were processed.

```
% avg_heading -i pass25.head -o pass25.avg_head -l 9
% avg_heading -i pass26.head -o pass26.avg_head -l 9
% avg_heading -i pass37.head -o pass37.avg_head -l 9
% avg_heading -i pass38.head -o pass38.avg_head -l 9
% gss_vel -i pass25.avg_head -r 45 -s 490 | utm-proj | projss -o pass25.map
% gss_vel -i pass26.avg_head -r 45 -s 490 | utm-proj | projss -o pass26.map
% gss_vel -i pass37.avg_head -r 45 -s 490 | utm-proj | projss -o pass37.map
% gss_vel -i pass38.avg_head -r 45 -s 490 | utm-proj | projss -o pass38.map
% cp pass25.map west.map
% qmos -i pass26.map -o west.map
% qmos -i pass37.map -o west.map
% qmos -i pass38.map -o west.map
% mode3 -i west.map -o west.mode3 -z -Z
% mode5 -i west.mode3 -o west.mode5 -z -Z
% mode3 -i west.mode5 -o west.mode3a -z -Z
% mode5 -i west.mode3a -o west.mode5a -z -Z
% filter -i west.mode5a -o west.lpfz -z -b 3,3
```

APPENDIX D

Below is a summary of the steps used to complete the mapping and processing of the three components (east, west and south) of the two degree GLORIA sidescan sonar map. Mapping was accomplished in a sub-directory from where the individual passes were stored. The required programs and their parameters, along with any UNIX required commands, were entered into a file and executed as a single script. Two additional script files, *utm-proj* and *bounds.dat*, were required to complete the processing. Those files are listed at the end of the processing script.

```
% more do-map2

# ****************************************************************
#   do_east component
# ----------------------------------------------------------------
#
rm -f pass41*.map pass42*.map pass43*.map
rm -f pass11.map pass12.map pass13*.map
rm -f pass26E.map
#
avg_heading -i ../pass41.head -o pass41.avg_head -l 9
gss_vel -i pass41.avg_head -l 400,720 -r 45 -s 490 -R | utm-proj | projss -o
pass41.map
rm pass41.avg_head
cp pass41.map east.map
#
avg_heading -i ../pass42.head -o pass42.avg_head -l 9
gss_vel -i pass42.avg_head -r 45 -s 490 -R | utm-proj | projss -o pass42.map
rm pass42.avg_head
qmos -i pass42.map -o east.map
#
avg_heading -i ../pass43.head -o pass43.avg_head -l 9
gss_vel -i pass43.avg_head -r 45 -s 490 | utm-proj | projss -o pass43.map
rm pass43.avg_head
qmos -i pass43.map -o east.map
#
avg_heading -i ../pass11.head -o pass11.avg_head -l 9
gss_vel -i pass11.avg_head -r 45 -s 464 | utm-proj | projss -o pass11.map
rm pass11.avg_head
qmos -i pass11.map -o east.map
#
avg_heading -i ../pass12.head -o pass12.avg_head -l 9
gss_vel -i pass12.avg_head -r 45 -s 464 | utm-proj | projss -o pass12.map
rm pass12.avg_head
qmos -i pass12.map -o east.map
#
avg_heading -i ../pass13.head -o pass13.avg_head -l 9
gss_vel -i pass13.avg_head -r 45 -s 490 -R | utm-proj | projss -o pass13.map
rm pass13.avg_head
qmos -i pass13.map -o east.map
#
avg_heading -i ../pass26.head -o pass26.avg_head -l 9
gss_vel -i pass26.avg_head -r 45 -s 464 -l 605,720 | utm-proj | projss -o
pass26E.map
qmos -i pass26E.map -o east.map
#   save pass26.avg_head for west.map and south.map processing
#
avg_heading -i ../pass27.head -o pass27.avg_head -l 9
```

```
gss_vel -i pass27.avg_head -r 45 -s 464 | utm-proj | projss -o pass27.map
rm pass27.avg_head
qmos -i pass27.map -o east.map
#
# ---------------------------------------------------------------
#   fill gaps in east component map
# ---------------------------------------------------------------
#
rm -f east.lpfz
#
mode3 -i east.map -o east.mode3 -z -Z
mode5 -i east.mode3 -o east.mode5 -z -Z
#
rm east.mode3
#
mode3 -i east.mode5 -o east.mode3 -z -Z
#
filter -i east.mode3 -o east.lpfz -z -b 3,3
rm east.mode3 east.mode5
#
# ***************************************************************
#   do_west component
# ---------------------------------------------------------------
#
rm -f pass37*.map pass38*.map pass39.map
rm -f pass25*.map pass26W.map
#
avg_heading -i ../pass37.head -o pass37.avg_head -l 9
gss_vel -i pass37.avg_head -r 45 -s 490 -R | utm-proj | projss -o pass37.map
rm pass37.avg_head
cp pass37.map west.map
#
avg_heading -i ../pass38.head -o pass38.avg_head -l 9
gss_vel -i pass38.avg_head -r 45 -s 490 -R | utm-proj | projss -o pass38.map
rm pass38.avg_head
qmos -i pass38.map -o west.map
#
avg_heading -i ../pass39.head -o pass39.avg_head -l 9
gss_vel -i pass39.avg_head -r 45 -s 464 | utm-proj | projss -o pass39.map
rm pass39.avg_head
qmos -i pass39.map -o west.map
#
avg_heading -i ../pass25.head -o pass25.avg_head -l 9
gss_vel -i pass25.avg_head -r 45 -s 464 -R | utm-proj | projss -o pass25.map
rm pass25.avg_head
qmos -i pass25.map -o west.map
#
#  pass26.avg_head created during processing of east.map
#
gss_vel -i pass26.avg_head -r 45 -s 464 -l 1,302 | utm-proj | projss -o
pass26W.map
qmos -i pass26W.map -o west.map
#
# ---------------------------------------------------------------
#   fill gaps in west component map
# ---------------------------------------------------------------
#
rm -f west.lpfz
#
mode3 -i west.map -o west.mode3 -z -Z
mode5 -i west.mode3 -o west.mode5 -z -Z
```

```
#
rm west.mode3
#
mode3 -i west.mode5 -o west.mode3 -z -Z
#
filter -i west.mode3 -o west.lpfz -z -b 3,3
rm west.mode3 west.mode5
#
# ***************************************************************
#   do_south component
# ---------------------------------------------------------------
#
#  pass26.avg_head created during processing of east.map
#
gss_vel -i pass26.avg_head -r 45 -s 446 -l 324,588  | utm-proj | projss -o
pass26S.map
rm pass26.avg_head
#
# ---------------------------------------------------------------
#   fill gaps in south component map
# ---------------------------------------------------------------
#
cp pass26S.map south.map
#
mode3 -i south.map -o south.mode3 -z -Z
mode5 -i south.mode3 -o south.mode5 -z -Z
#
rm south.mode3
#
mode3 -i south.mode5 -o south.mode3 -z -Z
#
filter -i south.mode3 -o south.lpfz -z -b 3,3
rm south.mode3 south.mode5
```

```
% more utm-proj

proj +proj=utm +lon_0=-111 -r -s -m 1:100 -f `%.0f' bounds.dat -

% more bounds.dat

-34 -114
-34 -112
-36 -112
-36 -114
```

REFERENCES

Chavez, Pat S., 1984, U. S. Geological Survey Mini Image Processing System (MIPS), Open-File Report 84-880, 12 p.

Chavez, Pat S., 1986, Processing Techniques for Digital Sonar Images from GLORIA, Photogrammetric Engineering and Remote Sensing, vol. 52, No. 8, pp. 1133-1145.

Evenden, Gerald I., 1990, Cartographic Projection Procedures for the UNIX Environment - A User's Manual, Open-File Report 90-284, 62 p.

Miller, Richard L., Dwan, Fa S. and Cheng, Chiu-Fu, 1991, Digital Preprocessing Techniques for GLORIA II Sonar Images, Geo-Marine Letters, 11:32-31.

Paskevich, Valerie, 1992, Woods Hole Image Processing System Software Implementation: Using NetCDF as a Software Interface for Image Processing, Open-File Report 92-25, 72 p.

Paskevich, Valerie, 1992, Digital Processing of Sidescan Sonar data with the Woods Hole Image Processing System Software, Open-File Report 92-204, 9 p.

Snyder, J.P., 1987, Map projections - A working manual: U.S. Geological Survey Professional Paper 1395, 383 p.

Snyder, J.P. and Voxland, R.M., 1989, An album of map projections: U. S. Geological Survey Professional Paper 1453, 249 p.

Unidata Program Center, NetCDF User's Guide: An Interface for Data Access, v1.11, March 1991, 150 p.