FILE

Institute of
Oceanographic Sciences
Deacon Laboratory

INTERNAL DOCUMENT No. 354

Swath bathymetry with GLORIA

T P Le Bas, M L Somers, Q J Huggett,
J M Campbell & R Beale

June 1995

Natural Environment Research Council

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

INTERNAL DOCUMENT No. 354

Swath bathymetry with GLORIA

T P Le Bas, M L Somers, Q J Huggett,
J M Campbell & R Beale

June 1995

Wormley
Godalming
Surrey GU8 5UB UK
Tel +44-(0)1428 684141
Telex 858833 OCEANS G
Telefax +44-(0)1428 683066

DOCUMENT DATA SHEET

AUTHOR	*PUBLICATION DATE*
LE BAS, T P, SOMERS, M L, HUGGETT, Q J, CAMPBELL, J M & BEALE, R	1995

TITLE

Swath bathymetry with GLORIA.

REFERENCE

Institute of Oceanographic Sciences Deacon Laboratory, Internal Document, No. 354, 80pp.
(Unpublished manuscript)

ABSTRACT

KEYWORDS

ISSUING ORGANISATION

Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB. UK.

Director: Colin Summerhayes DSc

Telephone Wormley (0428) 684141
Telex 858833 OCEANS G.
Facsimile (0428) 683066

Copies of this report are available from: ***The Library,*** *PRICE* *£0.00*

INSTITUTE OF OCEANOGRAPHIC SCIENCES
DEACON LABORATORY

Swath Bathymetry with GLORIA

by

T. P. Le Bas, M. L. Somers, Q. J. Huggett,

J. M. Campbell, and R. Beale.

June 1995

Brook Road, Wormley, Godalming, Surrey, GU8 5UB, United Kingdom.

Contents

Figure list

Introduction

This report describes the scientific background, processing procedure and calculations required to produce GLORIA swath bathymetry maps. Hand editing is not required (c.f. SeaMARC, EM12) as the whole process is automatic. The report is divided into four sections: 1) the mathematical background of swath bathymetry, 2) the computer software and its installation, 3) an explanation of the processing algorithms, and 4) an example dataset through all the processing stages and comparison with SeaBEAM data of the same area.

Background

Phase difference swath bathymetry was introduced with the SeaMARC II system in 1983 (Hussong et al). The basis of all swath bathymetry is the accurate measurement of an angle and a time for the reverberation returning from the seafloor. In hull-mounted multi-beam systems a number of pre-defined angles are (relatively) well known corresponding to the individual beams, and the time of arrival of the seabed echo along each beam has to be measured. This method requires the luxury of a large cross-track aperture which is not available in the towed vehicle systems deployed to acquire sidescan images. Here the only way to measure swath bathymetry is to infer arrival angles as a function of time from measurements of the phase differences between signals arriving at two or more closely spaced rows of receivers.

There are two problems to be overcome in measuring swath bathymetry by phase difference methods. In the first place the limited aperture means that small perturbations (e.g. due to noise) in the wavefront are fully reflected in differences in measured arrival angle, instead of being averaged over a large aperture. In the second place any attempt to alleviate this problem by using a larger aperture leads inevitably to the ambiguities which arise when the phase difference can exceed 360°. These problems have already been analysed in the literature (Denbigh 1988, Masnadi-Shirazi 1992 and De Moustier 1993).

In applying the method to the GLORIA system (Somers et al., 1978) there were two complications. Firstly it was necessary to use the existing arrays which are spaced more than three quarters of a wavelength apart rather than the maximum unambiguous value of half, and secondly the GLORIA system uses a linear FM pulse of 2 seconds duration which leads to some complications (and advantages) in the processing. In what follows we discuss the application of phase difference swath bathymetry to the GLORIA system with an analysis of performance and illustration of results.

Outline of the Method

Figure 1 shows an outline diagram of the GLORIA array configuration and the principles of phase difference bathymetry. The boresight is tilted down from the horizontal by 20°. The phase difference between the signals on the upper and lower rows is given, for a seabed echo arriving along the angle θ by

$$\phi = kd \sin(\theta - \alpha)$$

As θ varies from 90° at the nadir out to the horizontal (θ-α) varies from 70° through zero on the boresight 20° below the horizontal to -20° on the horizontal. Figure 2 shows how the phase differences for port and starboard signals vary while the echo direction goes through this range of angles. It shows how there is a fairly small range of angles in which the phase differences are quite unambiguous, especially when allowance is made for sound returning from above the horizontal by surface reflection. Over the rest of the circle the ambiguity has to be resolved.

Figure 1 Outline diagram of GLORIA transducer configuration and principles of phase difference swath bathymetry

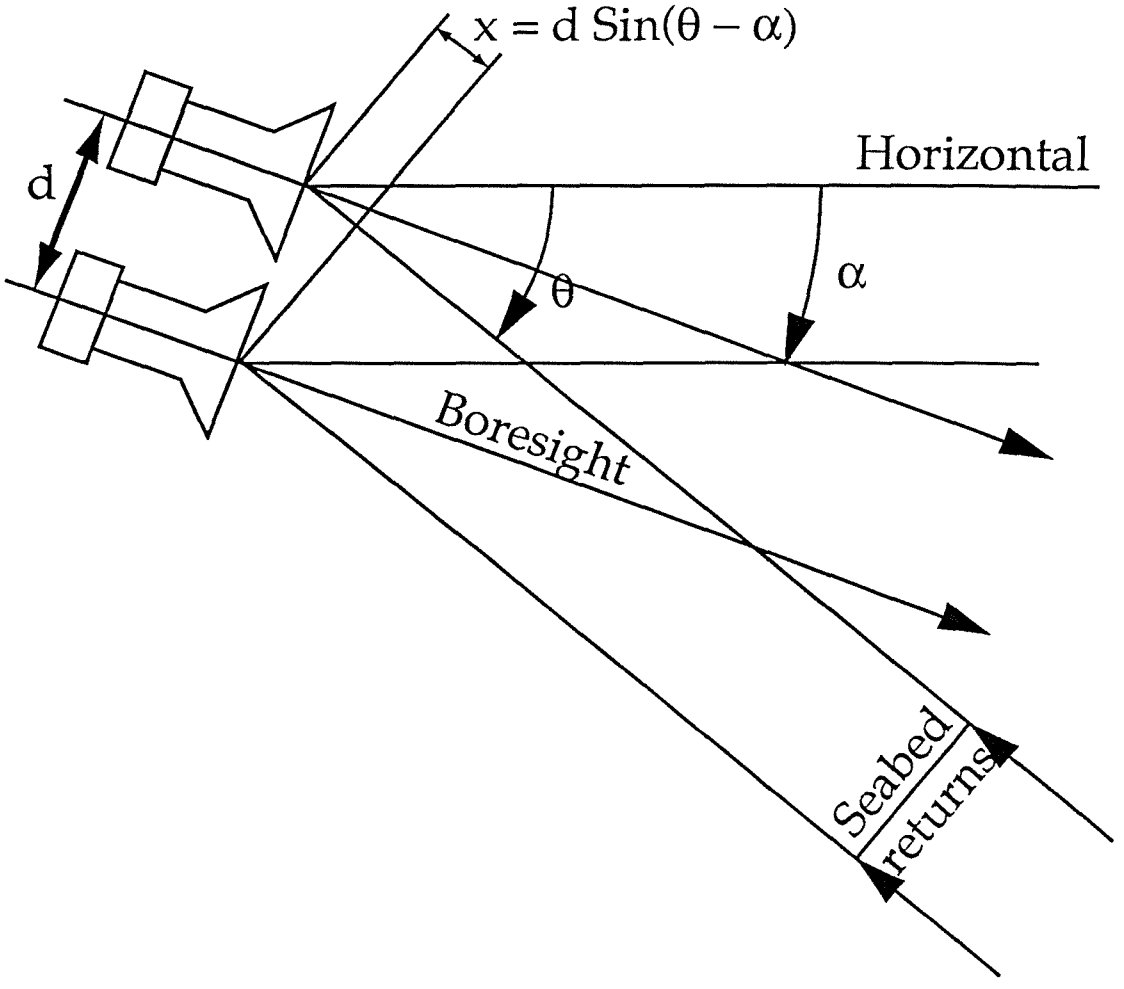


Figure 2 Phase difference to range angle conversion diagram

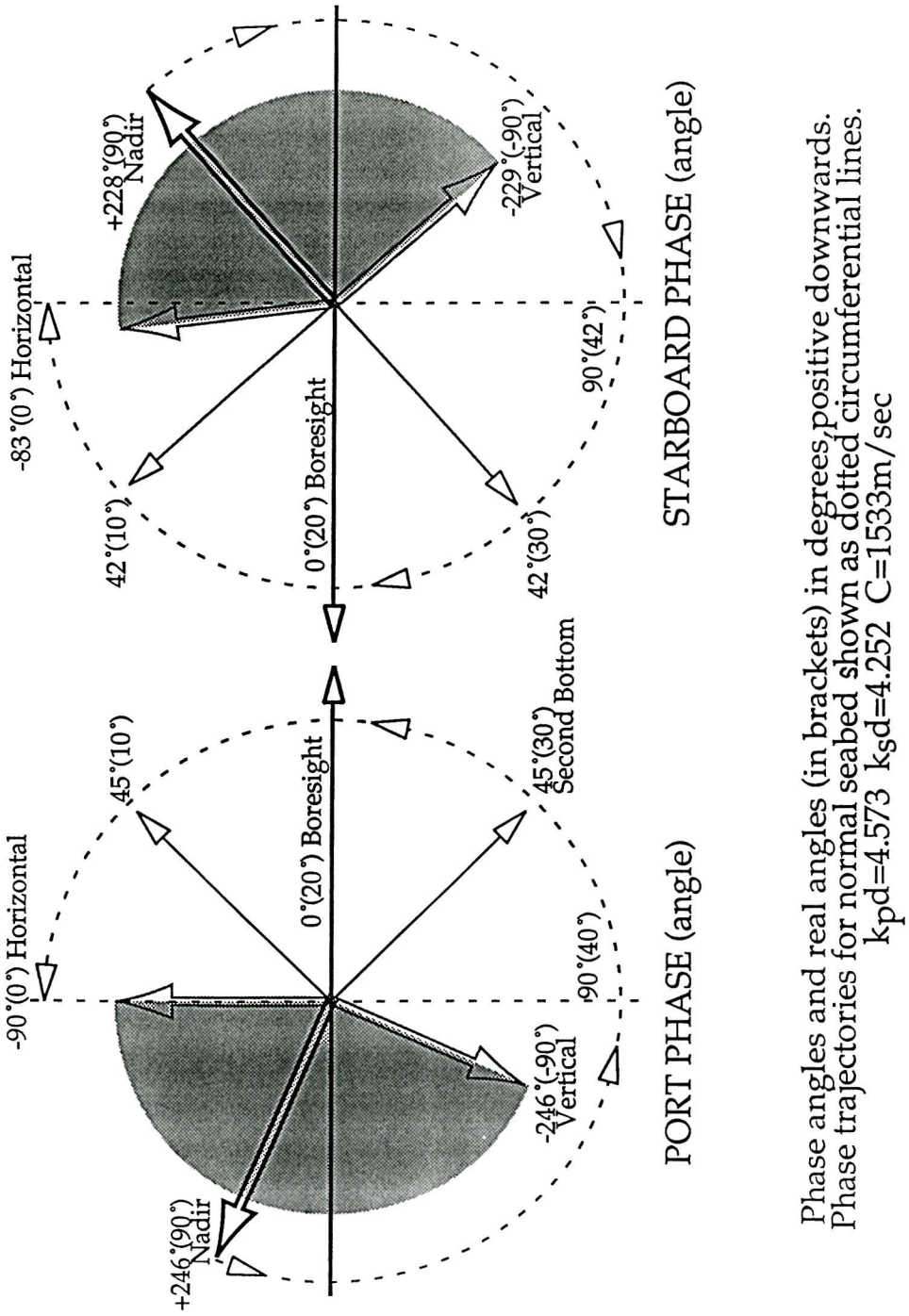


Figure 3 shows the problem with the long FM pulse. At any time the pulse footprint on the ground extends upwards of 1500m and this can subtend an angle a the array of 10° or more. The method chosen to process the long FM pulse is to work in the frequency domain on short time segments of data. Signals from the 6 sections of each GLORIA array are amplified, filtered and sampled as shown in Figure 4. The sampling is a complex quadrature sampling (Grace and Pitt, 1970) with each sample consisting of an in-phase (I) component and a quadrature (Q) component representing the cosine and sine amplitudes of the original wave. If the signal bandwidth is less than B Hz it suffices to take B such pairs per second, where each pair represents two snapshots of the waveform taken exactly a quarter period of the band centre frequency apart. Note that there are in this scheme more than $2B$ independent samples per second in agreement with the Sampling Theorem. The waveform is approximated by its samples according to:

$$e(t) = \sum_{n} (I_n \cos \omega 0t + Q_n \sin \omega 0t) U(nT + t)$$

where U(t) =0 t<=0, t>T
=1 0<t<=T

Here T is the sampling interval. Provided T <1/B the approximation becomes exact at the output of a zonal filter of band B centred on the correct frequency.

It will also be apparent that the phase of the signal is:

$$\phi = \arctan\left(Q_n / I_n\right)$$

However it is the phase difference between the rows which is of importance and it is not necessary to take two arctan functions for each phase estimate. If the upper row signal is $I_u(n)+jQ_u(n)$ and the lower one is $I_l(n)+jQ_l(n)$ then the vector quotient will have a phase equal to the required difference. This leads at once to:-

$$\Delta\phi(n) = \arctan\left(\frac{I_u(n)Q_l(n) - I_l(n)Q_u(n)}{I_u(n)I_l(n) + Q_u(n)Q_l(n)}\right)$$

The same formula applies working in the frequency domain, but with the single difference that I(n) and Q(n) are the real and imaginary parts of the contents of the nth frequency bin in the DFT of a given time slice. If we take all the frequency bins relating to a single target at a given range then since the various bins contain fully independent samples some sort of average should give a better estimate of the overall phase than a single bin. De Moustier has shown that the optimum estimator for the phase difference is a weighted Least Mean Square estimate given by:-

$$\overline{\Delta\phi} = \arctan\left[\frac{\sum_{n=1}^{N} W^2(n)\left(I_u(n)Q_l(n) - I_l(n)Q_u(n)\right)}{\sum_{n=1}^{N} W^2(n)\left(I_u(n)I_l(n) + Q_u(n)Q_l(n)\right)}\right]$$

Here W(n) is the nth member of a weighting sequence. This LMS formula allows for the energy in each frequency bin, as averaging simple angles cannot. In the absence of any concrete indication otherwise W(n) is set to unity. One can put forward arguments in support of giving greater weight to bins where the two signals are more nearly equal, but no significant benefit has been found for any weighting but the uniform one.

Figure 3 GLORIA linear FM pulse

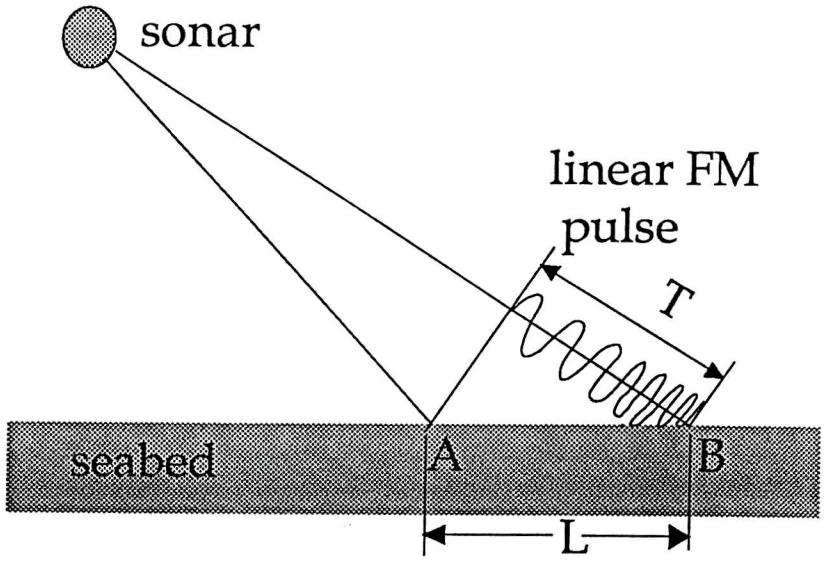
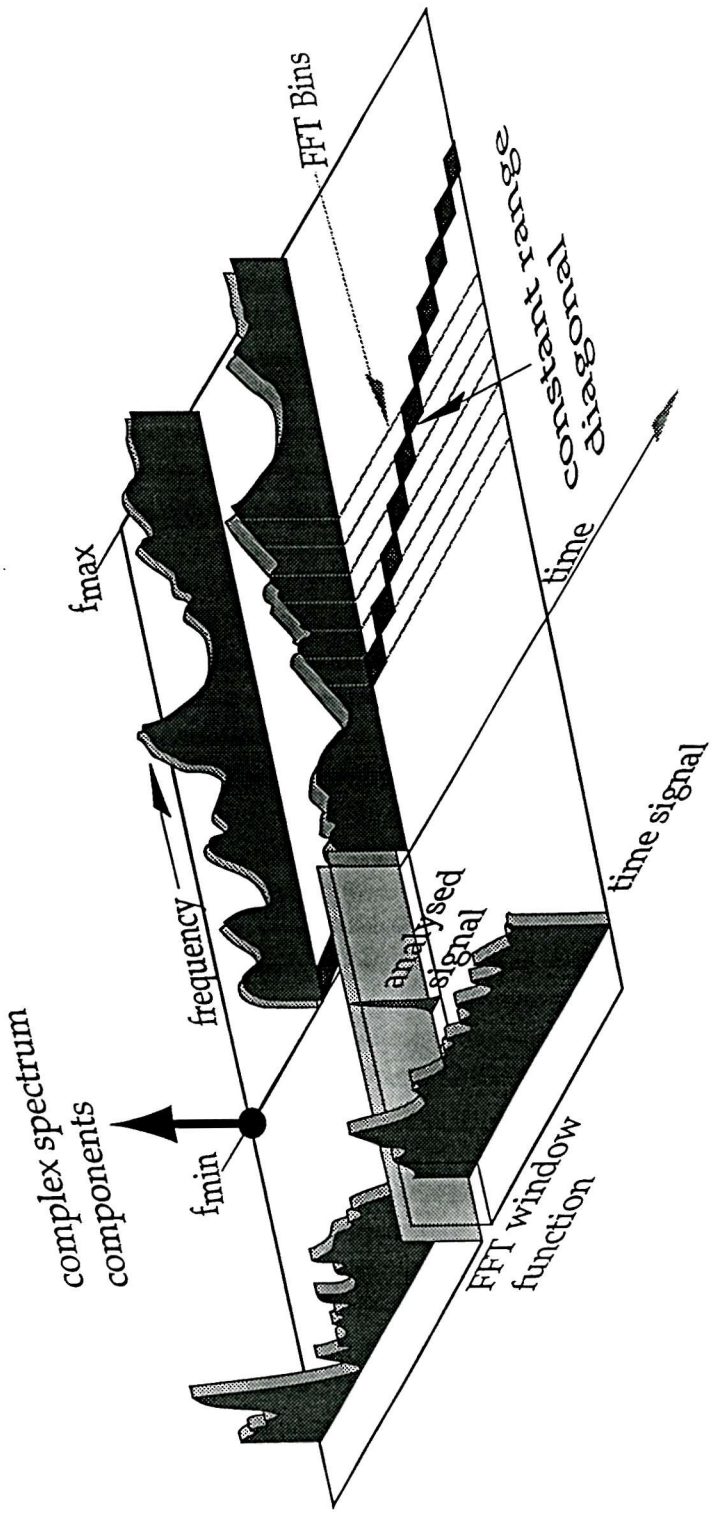


Figure 4 Conversion of the time signal into FFT bins for each transducer array



Data Processing

In the GLORIA processing the sampling rate (complex) is 228 Hz with a signal band of 100Hz i.e. an oversampling ratio of 2.28 used to counter aliassing and interpolation errors. Because of this the 64 bins in each spectrum actually only contain signal energy in 28.

The FFT used is a 64 point complex radix-4 computation carried out on a TMS320C30. It is computed at 16 sample intervals starting shortly after the end of the transmission. The 64 sample complex time series are weighted with a Kaiser-Bessel window to maximise the contribution of the central 16 points with a minimum of leakage. Because the sampling pulse stream is not synchronised with the pulse generation system there is inevitably a one sample jitter in the start of sampling. Figure 5 shows the processing steps. First the section signals are summed with the correct beam stabilising phase shifts, then the roll correction is applied (see later) and the signals are segmented for FFT computation. The FFT data is saved with a lot of auxiliary data as the raw phase data.

The data timing is shown in Figure 6 where the FFT data are represented as a matrix in which constant range points are diagonals. If we look upon the lowest frequency bin row as the time reference the travel time is calculated as:-

$$T_t = 2.208 - 32t + 16(n-1)$$

Here 2.00 is the pulse length in seconds 0.208 is a post pulse delay before the start of sampling. 32t is the number of sample intervals between the start of the first sample and the centre of the first time sequence, and n is the FFT number.

Starting from an elapsed time of just over 2 seconds data are accumulated for 18 seconds corresponding to a range of 15Km. Only the 28 in-band spectral samples are saved, and this produces a data record of 230KBytes. Using IEEE floating point numbers the data rate is some 27.5MBytes per hour.

Figure 5 Processing steps for GLORIA swath data correction

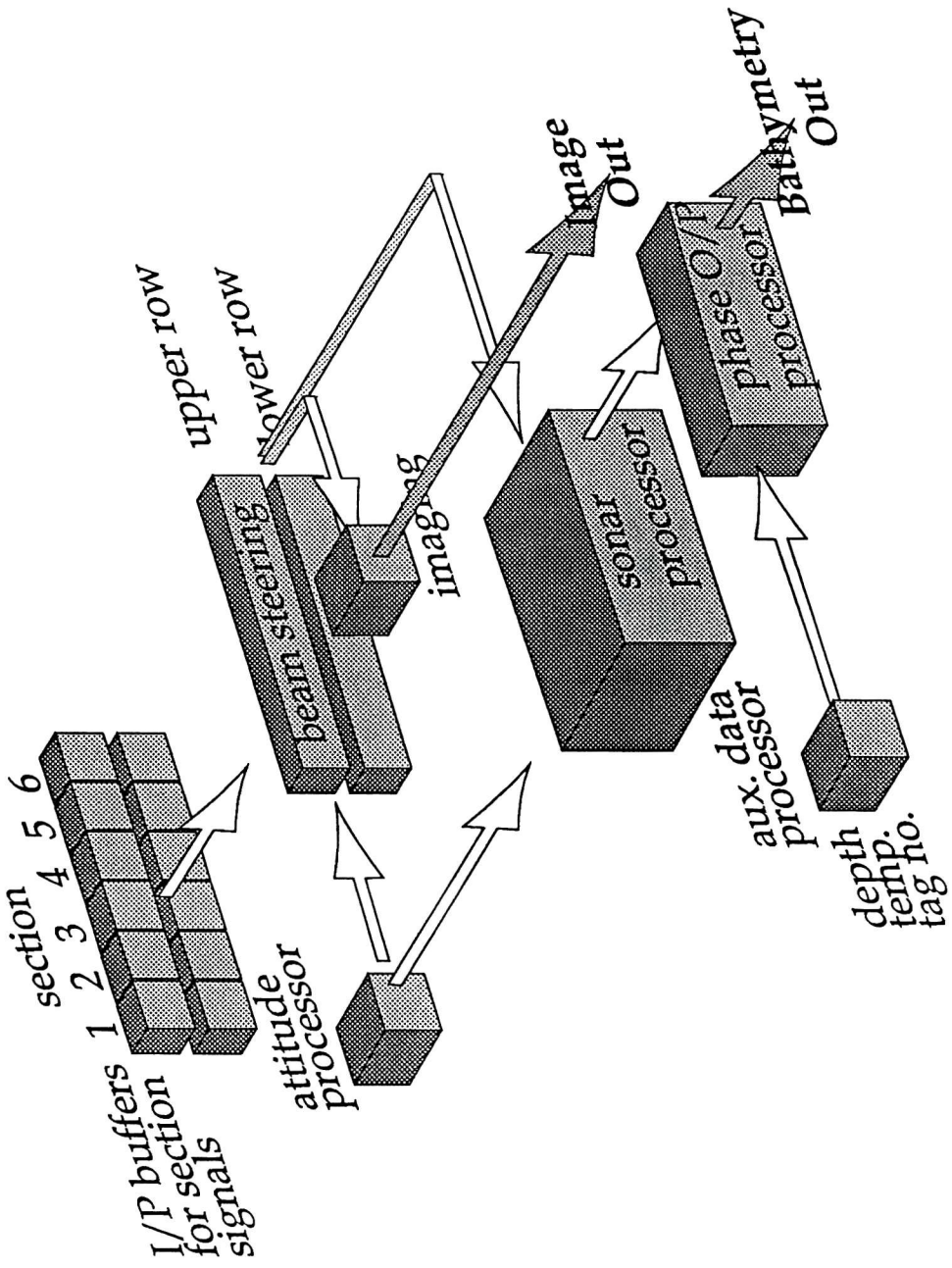
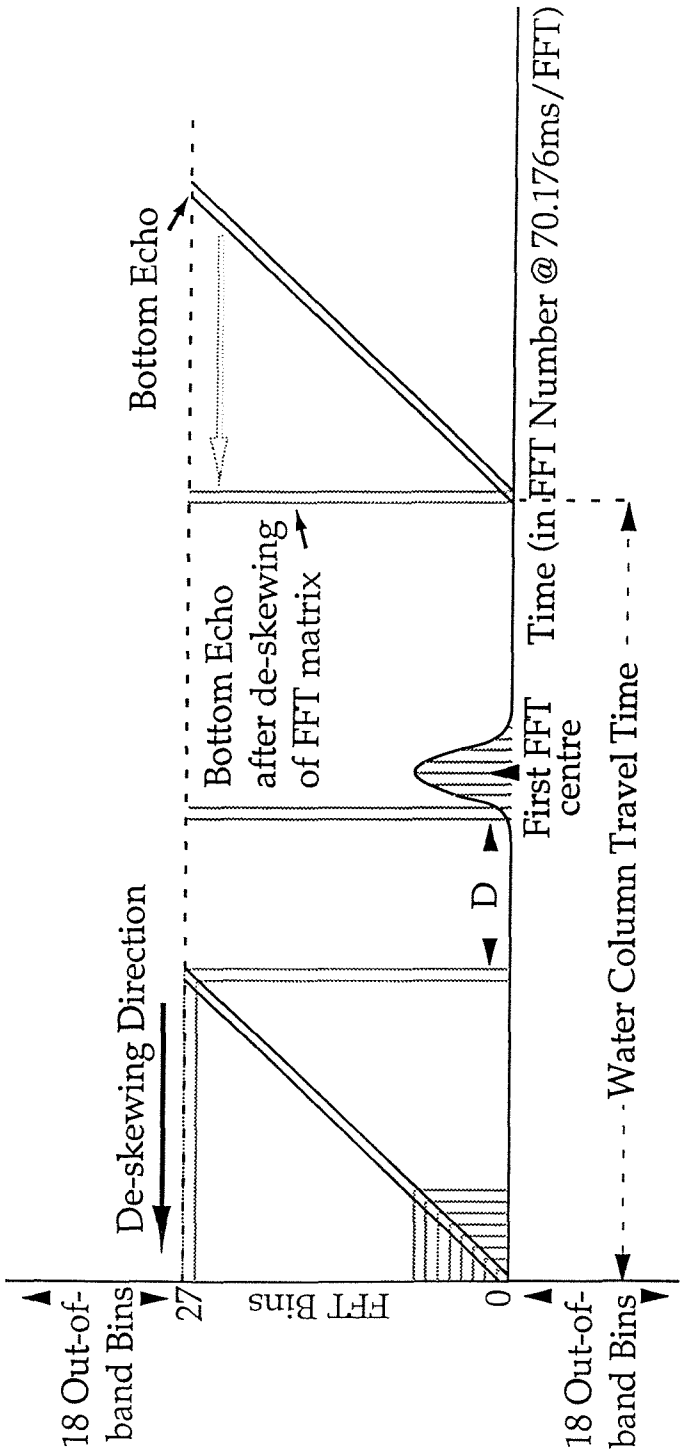


Figure 6 The data timing of the FFT data and de-skewing



Depth Processing

The first stage in the depth processing is to de-skew the FFT matrix so that points of constant range become columns. In fact it is at this stage that roll is most conveniently removed, the data having been transferred over the network to a UNIX workstation. Each FFT bin for a given range has a time attached to it and the corresponding roll value is available in the data header. The complication in compensating for roll is that the roll angle is related to phase by the sine function as follows:-

$$\phi = kd\sin(\theta - \alpha + R)$$

So that:-- $$\delta\phi \approx kd\cos(\theta - \alpha)\delta R$$

Since the GLORIA roll never exceeds 0.5° and is usually less than 0.15°, the correction is small in any case. In addition the roll period almost coincides with the pulse length so the roll effect is integrated over the LMS computation. Taking these two effects into account and using a constant value of 0.8 for $\cos(\theta-\alpha)$ a roll of 1° will be reduced to less than 0.007°, which will produce an error of less than 50m in 5000m of water at a range of 1.5 water depths.

After roll correction the LMS estimate is computed for each range point (at intervals of about 60m). Near the nadir there is a phase jump of 2π and in the presence of noise the boundary can be crossed and re-crossed several times. These ambiguities can be removed by working back from the unambiguous portion near the second bottom echo unwrapping the phase as necessary, a process which proves to be quite simple and robust.

The raw phase versus time values are converted to depth and offset from track in a two stage process. First a phase to angle conversion is performed. Corrections have to be added to account for the departure of the calibrated phase curve from the theoretical. There is also a bias to be removed, which arises as a result of surface reflections, as explained later. The times and launch angles are then entered into a propagation table to find the depth and offset.

In the absence of shadowing the offsets from the trackline should occur in the same time order as the FFT numbers, but the presence of noise and other extraneous signals causes phase errors which can disturb this order. The algorithm to select bad points is to run a short FFT smoothing filter over the line and reject points more than a threshold out of order. A short filter is needed to avoid hiding the real variations.

The achievable contour interval depends on the amount of smoothing applied. A large feature becomes more evident with smoothing and can thus be contoured more closely. By contrast, small features tend to get lost either by sinking beneath the noise or being smoothed out of existence. Hence the contouring interval will be a compromise.

Errors and Bias

Various sources of error and bias have been alluded to in the previous sections, of which the most important are:

a. Second and subsequent bottom echoes.
b. Surface reflections.
c. Errors in the phase calibration curve.
d. Sound speed errors at the transducer interface.
e. Water column sound speed errors.
f. Noise.

Some of these effects cause only an increased variance in the depth estimates, but others, notably surface reflections, cause a bias as well. The sound speed errors cause only a bias.

a. The second bottom echo occurs when the pulse has travelled to the seabed and back twice, being reflected by the surface after the first round trip. It coincides (by simple geometry) with the true bottom returns

about 30° below the horizontal. This artefact is much more serious over a flat bottom with a fair degree of coherence in its echo. The specular surface reflection always has a high degree of coherence. A coherent bottom echo (which means in effect a flat sediment seafloor) will normally return a stronger signal than the backscattering at a 30° grazing angle. In GLORIA images the second bottom echo is a persistent feature of flat topography and in places the third, fourth and even fifth bottom echoes can be seen. On the other hand there are good data in the intervening sections, so the multiple bottom echoes have to be edited out. The editing algorithm involves comparing the ordering of the samples in offset from track with the time ordering. A second bottom echo occurs in the time order at 1.732 water depths but in the track offset order they occur much too soon. This is because the bottom reflected energy comes from the nadir and this is reflected in the angle estimates, so there is an apparently enormous depth below the track; thus the bad samples can, mostly, be recognised and weeded out.

b. Some sound reaches the transducer faces by way of a surface reflection as shown in Figure 7. These surface reflections are reduced with respect to the direct returns only by the surface reflection loss and by the vertical directivity of the transducer elements. Since they have a fairly well defined direction associated with them and this is related to the direct arrival angle they cause both noise and bias.

Figure 8 shows the vector model for the surface reflection effect. Since the surface is neither flat nor stationary the instantaneous power level at the transducers will fluctuate. Also since the surface is many wavelengths above the transducers there will be no relation whatever between the absolute phases of the direct and surface reflected energy. The phase difference for the reflected energy between the two rows will be related to that for the direct sound since it is the specular or very nearly specular component which will cause the interference. Because the surface is not flat there will be some variation from the specular direction as shown.

To model the effect we can estimate a surface loss and add the directivity function to get the mean reflected energy. This will fluctuate and the simplest way to model it is to assume the amplitude will be Rayleigh distributed such that the energy comes out right. The absolute phase β is quite indeterminate, so it is uniformly distributed between zero and 2π. The variability in direction due to surface movement is modelled by a Gaussian distribution of the reflection angle about the specular as shown. The phasor diagram in Figure 8 illustrates this, and running a simulation does indeed predict an angle related bias as well as an added variance in the estimated arrival angles. In Figure 8 OP and OQ are the two wanted row signals and it is the phase angle, ϕ, between them we wish to measure. PP' is the surface reflection at the upper row, and QQ' is the corresponding vector for the lower row. If Q'Q" and PP' are parallel the angle Q'QQ" is the rotation of the surface vector in passing from the upper to the lower row of sensors. The angle ϕ' is the one we actually measure. For the GLORIA transducers which have rather little vertical directivity, and for reasonable surface loss, the bias amounts to several degrees and actually changes sign within the quadrant. This diagram is easily modified to take account of random noise. In practice there will be calibration errors and the surface loss is somewhat sea-state dependent so the correction is applied using an empirical correction computed from data taken over a flat portion of seabed.

c. This error arises from the non-ideal acoustic mounting of the arrays, and it can be incorporated in the empirical correction as stated above.

d. Sound speed errors at the transducer face give rise to errors in the phase slope since the kd value for the row spacing depends on sound velocity. Errors in the actual angle can be as much as 1° for a change in water temperature corresponding to going from the tropics to the poles, so it must be factored in. The expression for the change of phase with velocity is :

$$\frac{\delta\phi}{\phi} = -\frac{\delta c}{c}$$

e. Water column errors arise from an incomplete knowledge of the velocity profile. Sufficient accuracy can be obtained by regular use of deep XBTs.

f. Noise actually reduces the bias while as expected increasing the variance of the angle estimates, but only if it is truly random. In the real ocean the predominant source of noise is the ship, and this imposes a

direction on the noise. This will add to both the bias and the variance, but hard figures naturally depend on the ship.

Figure 7 Surface reflection geometry for seabed echoes causing both noise and bias

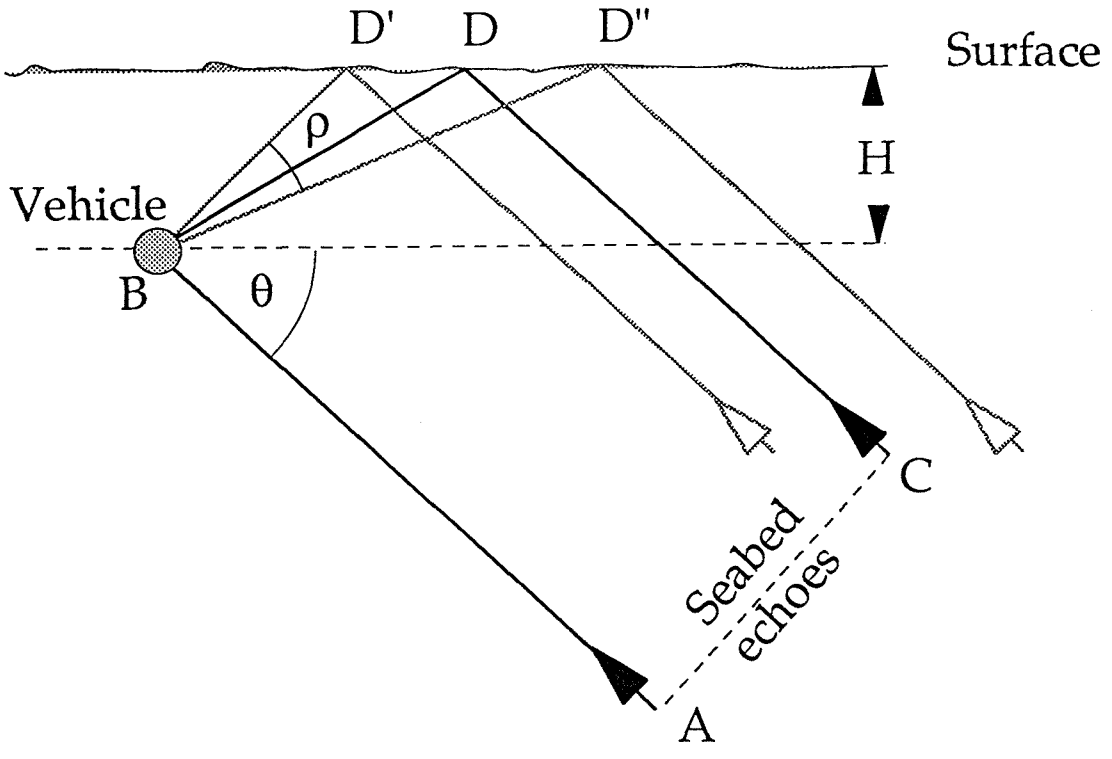
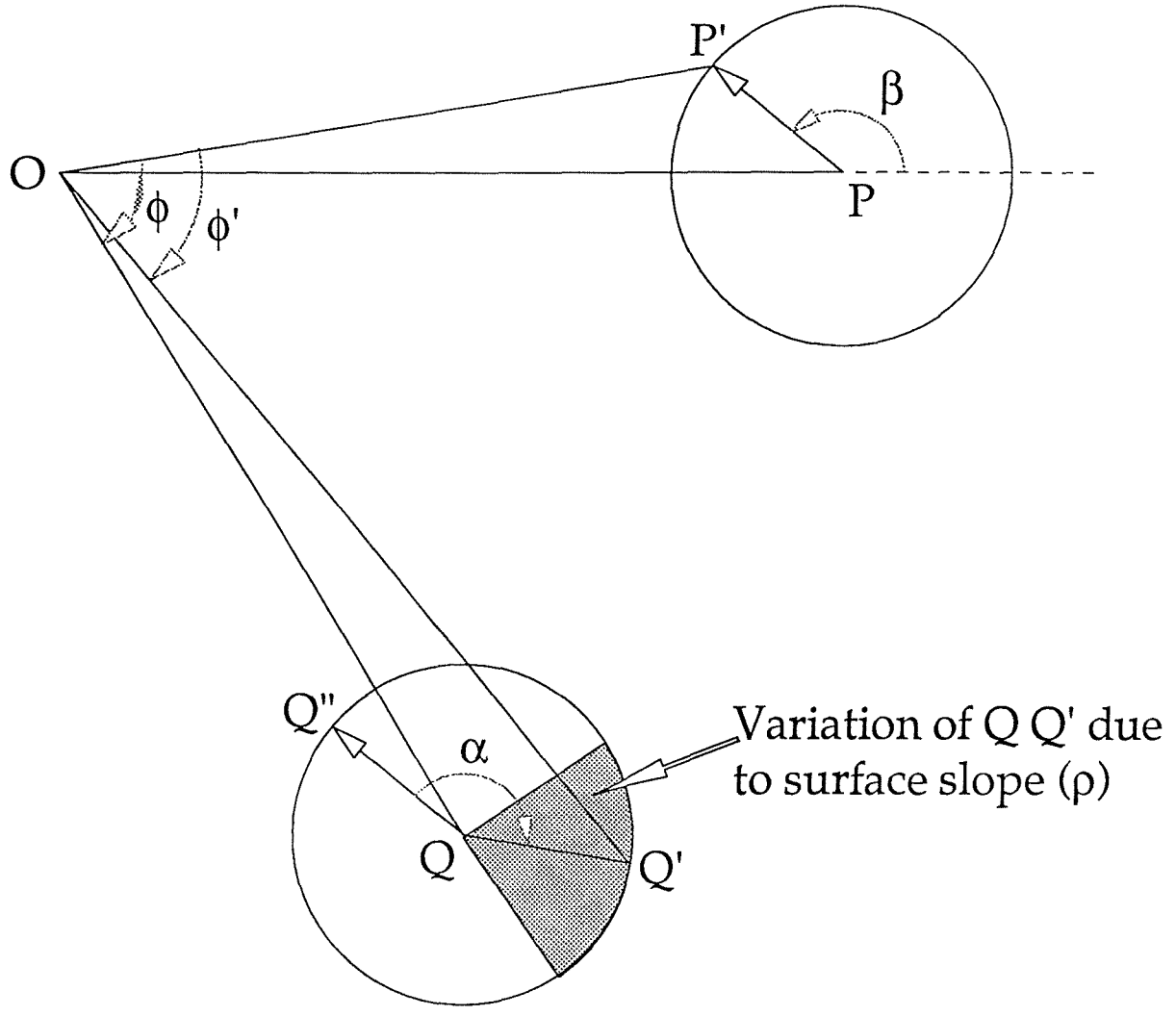


Figure 8 Phasor diagram modelling the effect of surface interference on the mean reflected energy



Software

Five main software packages (NetCDF, GMT, WHIPS, PROJ and SWATH) are used for data processing. A 5Gb Exabyte 8mm tape has been created which contains the five tar files for the software systems together with the installation notes (also included below). The tape itself was written on a SUN classic running SOLARIS 2.3 using the command:

tar cvf /dev/rmt/0 .

The description of the contents of the tape are listed below:

```
drwxr-x---  20024/20024       0 May  5 11:22 1995 ./
-r-xr--r-x  20024/20024 4058595 May 11 13:32 1994 ./GMT2.1.4.tar.Z
-rw-r-----  20024/20024 3090587 Mar 24 14:08 1995 ./whips_v2.2.tar.Z
-rwxrwxr-x  20024/20024 1096340 Mar  6 11:14 1995 ./netcdf-2.3.2.tar.Z
-rw-rw-rw-  20024/20024  190845 Mar  6 12:30 1995 ./PROJ.4.3.tar.Z
-rw-rw-rw-  20024/20024    4649 May  5 11:21 1995 ./install.txt
-rw-r-----  20024/20024 1959905 Mar 24 13:36 1995 ./swath_v1.2.tar.Z
```

To extract these from the tape use a command similar to:

tar xvf /dev/rmt/0

NetCDF is a self-describing /network-transparent data format for data access provided by Unidata. Unidata is a United States national program sponsored by the NSF which is available to all academic communities at no cost. NetCDF files are available on *unidata.ucar.edu* and various Archie-type sites

GMT is a public domain software, so widely used now that it seems redundant to present it in detail. GMT is a set of functions that allow to manipulate, filter, interpolate, and display many types of data. It is based on the NetCDF format, but also accepts ASCII files as input. The output files can be in ASCII, NetCDF format or PostScript. The software is available by anonymous login on *kiawe.soest.hawaii.edu*. In-depth discussion of GMT capabilities can be found in Wessel and Smith (1991). GMT works on a wide range of UNIX workstations. Documentation is available both as a printable manual, in PostScript format, and as on-line help by manual pages.

WHIPS is also public domain software, written to update and refresh the functions provided by MIPS (Chavez, 1986), as well as to port them to other platforms. Its constitutive programs are written mainly in C, and should function under a variety of UNIX environments. Approximately 150 routines are provided at the time of writing, mainly for the processing of GLORIA and TOBI sidescan imagery as well as general image processing. WHIPS uses the NetCDF format for input/output, which provides an additional degree of platform-independence. Some modules are also available for format conversion.

PROJ is a standard UNIX filter function which converts geographic latitude and longitude co-ordinates into Cartesian x and y using a wide spectrum of cartographic projection systems (about 70). The inverse conversion of the projection is also possible if required. The source code in C is also available via the Internet from gie@charon.er.usgs.gov.

The SWATH software has been written to complement the above systems, working either with ASCII data files or NetCDF grids. It is this software suite which will be utilised most for the initial GLORIA swath bathymetry data processing. The krigswath2 program uses a UNIRAS library function for kriging. This is a commercial package which is available at all NERC sites. If this package is not available it is possible for the kriging subroutine to be substituted for another available in other software packages.

For the final output hardcopy two packages have been used xv and ERDAS Imagine 8.1, neither of which are included here. xv is an interactive image display for the X window system written by John Bradley and is shareware. It is available from several sources including over the .

World Wide Web (http://www.hensa.ac.uk/ftp/pub/walnut.creek/FreeBSD-current/ports/distfiles/xv-3.10a.tar.gz). It can import and export images from several formats including PostScript. ERDAS is a commercial raster-based image processing system, with significant cartographic capabilities. It is used extensively in the US, and many universities in the UK. It runs on many different platforms using the X-11 based interface and is very user-friendly. The software presents the classical image processing capacities, as well as excellent map making facilities and links with Arc/Info (ESRI) Geographical Information System software.

Installation

1) NetCDF (version 2.3.2)
2) GMT (version 2.1.4)
3) WHIPS (version 2.2)
4) PROJ (version 4.3)
5) SWATH (version 1.0)

First, define a directory area for the software such as /local/tlb (= $SOFTDIR) and create five directories (e.g.):

£ setenv SOFTDIR /local/tlb

£ mkdir $SOFTDIR/netcdf
£ mkdir $SOFTDIR/gmt
£ mkdir $SOFTDIR/whips
£ mkdir $SOFTDIR/whips/proj
£ mkdir $SOFTDIR/swath

Second, put the tar files in each of these directories, uncompress them (if necessary) and then un-tar them using (version numbers may vary):

£ cd $SOFTDIR/netcdf
£ uncompress netcdf.tar.Z
£ tar xvf netcdf.tar

£ cd $SOFTDIR/gmt
£ uncompress GMT_2.1.4.tar.Z
£ tar xvf GMT_2.1.4.tar

£ cd $SOFTDIR/whips
£ uncompress whips_v2.2.tar.Z
£ tar xvf whips_v2.2.tar

£ cd $SOFTDIR/whips/proj
£ uncompress PROJ.4.3.tar.Z
£ tar xvf PROJ.4.3.tar

£ cd $SOFTDIR/swath
£ uncompress swath_v1.0.tar.Z
£ tar xvf wath_v1.0.tar

Installation of these packages must be done in order, as they are inter-dependant for various library routines.

Login variables:

The following environment variables must be added to the .login file (or equivalents):
£ setenv SOFTDIR /local/tlb

£ setenv GMT_BIN $SOFTDIR/gmt/bin

```
£ setenv PROGS          $home
£ setenv DESTDIR        $SOFTDIR/whips
£ setenv SRCDIR         $SOFTDIR
£ setenv BINDIR         $SOFTDIR/whips/bin
£ setenv WHIPSLIBDIR    $SOFTDIR/whips/llb
£ setenv WHIPSINC       $SOFTDIR/whips/include
£ setenv WHIPSDIR       $SOFTDIR/whips
£ setenv SWATHDIR       $SOFTDIR/swath
```

The following must be added to path:

£ set path = (. /usr/ucb $path $GMT_BIN $PROGS $DESTDIR/bin $DESTDIR/exe $SWATHDIR/bin $SWATHDIR/exe)

The latest versions of the C and FORTRAN compilers are also required:

```
£ setup c
£ setup f77
```

NETCDF installation notes

Change directory to the source directory and configure with the prefix set to the main directory (written out in full):

```
£ cd $SOFTDIR/netcdf/netcdf-2.3.2
£ configure --prefix=/local/tlb/netcdf
£ make all
£ make install
£ make clean
```

GMT installation notes

change directory to the source directory and edit the makefile, setting the 4 directory names appropriately:

```
£ cd $SOFTDIR/gmt/src
£ mkdir $SOFTDIR/gmt/bin
edit the makefile with correct pathnames: e.g.
        NETCDFLIB    = $SOFTDIR/netcdf/lib
        NETCDFINC    = $SOFTDIR/netcdf/include
        BINDIR       = $SOFTDIR/gmt/bin
        LIBDIR       = $SOFTDIR/gmt/lib
        and change install, ranlib, xdr if necessary.
£ make all
£ make install
£ make clean
```

WHIPS installation notes

Create or copy five library files (main utility library, TOBI utility library, NetCDF library, GMT library and postscript library) to the whips library directory:

```
£ mkdir $SOFTDIR/whips/bin
£ cd $SOFTDIR/whips/src/libraries/utilib
£ make
£ make lib
£ cp     libwhipsutil.a     $SOFTDIR/whips/lib
£ make clean

£ cd $SOFTDIR/whips/src/tobi/lib_tobi
£ make
```

£ make lib
£ cp libtobiutil.a $SOFTDIR/whips/lib
£ make clean

£ cp $SOFTDIR/netcdf/lib/libnetcdf.a $SOFTDIR/whips/lib
£ cp $SOFTDIR/gmt/src/libgmt.a $SOFTDIR/whips/lib
£ cp $SOFTDIR/gmt/src/libpsl.a $SOFTDIR/whips/lib

The random function (get_rand.c) is dependant on operating system. The SunOS 4.1 version has caused several problems which requires a different version of get_rand.c subroutine to be available. The subroutine is used in the following programs:

random_dn	$SOFTDIR/whips/src/ios/random_dn
random_img	$SOFTDIR/whips/src/ios/random_img
suppress_glo	$SOFTDIR/whips/src/gloria/suppress_glo
drpout	$SOFTDIR/whips/src/tobi/drpout
suppress_tobi	$SOFTDIR/whips/src/tobi/suppress_tobi

Go to the appropriate directories and copy the alternative subroutine:
£ cp get_rand.c.sun get_rand.c
This is not necessary for SOLARIS.

Finally compile all the programs:

£ cd $SOFTDIR/whips/src
£ make all
£ make install
£ make clean

PROJ installation notes

The SunOS 4.1 version has caused several problems which are not all solved. It requires gcc to be available.
£ setup gcc
Also two files need to be created float.h (for strtod.c) and emess.o (for emess.c)
£ cd $SOFTDIR/whips/proj/PROJ.4/src
£ cp temp_float.h float.h
£ cp temp_emess.o.sun emess.o
This is not necessary for SOLARIS.

Change directory to the main directory and configure with the prefix set to the main directory (written out in full):

£ cd $SOFTDIR/whips/proj/PROJ.4
£ configure --prefix=/local/tlb/whips/proj/PROJ.4
£ sh install
£ cd src
£ make clean

Copy the executable to the correct directory:

£ cp $SOFTDIR/whips/proj/PROJ.4/src/proj $SOFTDIR/whips/bin

SWATH installation notes

Initially setup the environment variables and compilers for compilation (UNIRAS kriging software):

£ setup new uniras
£ setup 1.2.2 motif

£ cd $SOFTDIR/swath/src

£ make all
£ make install
£ make clean

Now that all the software is available a processing chain can be defined.

Processing Chain

To aid the user a processing chain has been created which automatically executes the individual programs which are described below. Figure 9 shows a flow-diagram of the process required to convert raw phase data into bathymetry maps. The processing chain called process_swath is listed in Appendix C. It is written as a C shell script.

Process_swath requires up to 6 user input parameters dependant on the option required:

Option 1 - Processing raw phase data and output bathymetry image and grid
- Map Number (to give map limits)
- Full directory name where the raw phase data are stored
- Cruise name (e.g. mlvl0693)
- Magnitude threshold for cut-off of low magnitude value phase data
- Semi-variance threshold for removal of high semi-variance grid values
- Map grid node spacing (in degrees)

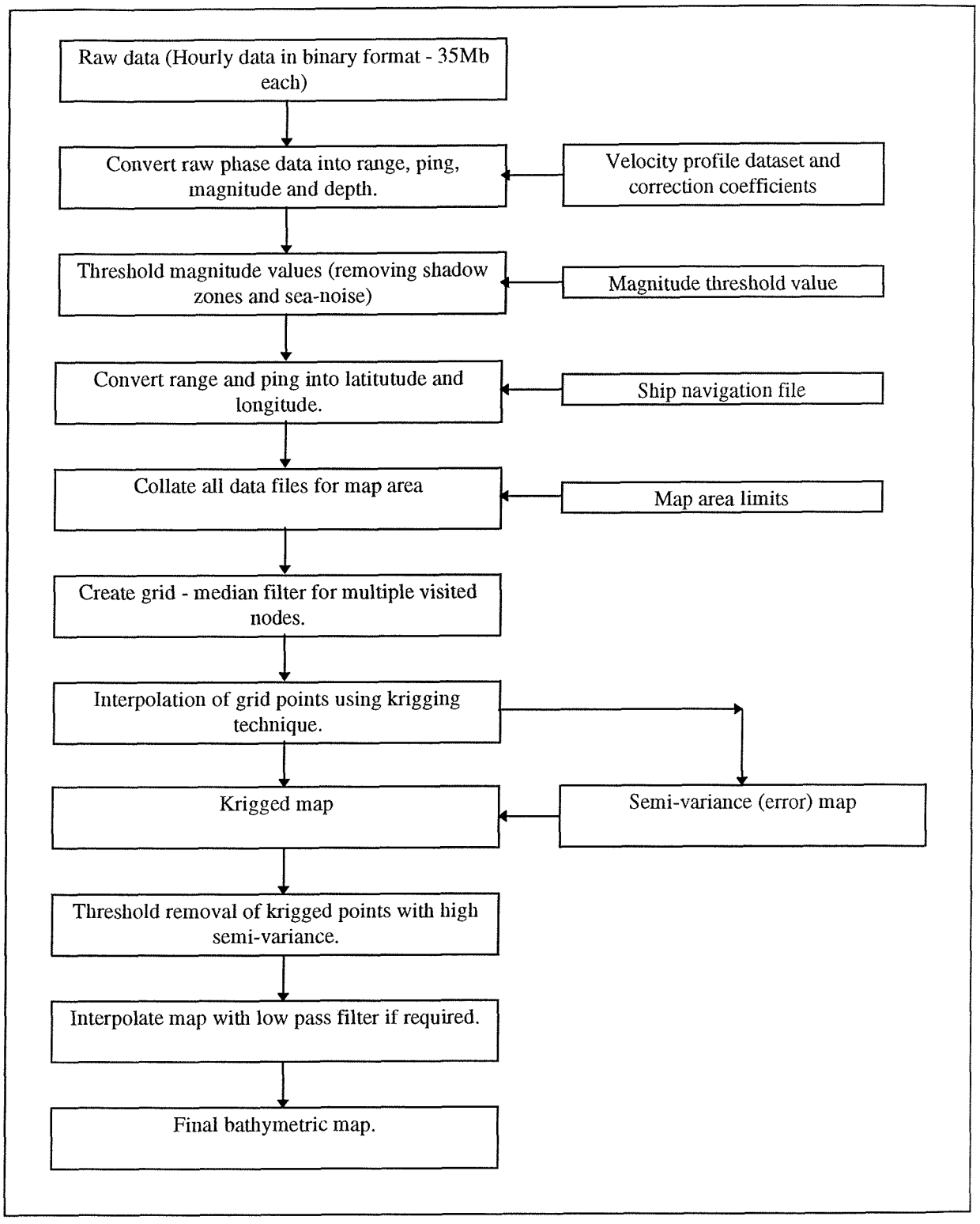
Option 2 - Processing raw phase data and output geographic depth points (in ASCII)
- Map Number (to give map limits)
- Full directory name where the raw phase data are stored
- Cruise name (e.g. mlvl0693)
- Magnitude threshold for cut-off of low magnitude value phase data

Option 3 - Processing geographic depth points and output bathymetry image and grid
- Map Number (to give map limits)
- Full directory name where the raw phase data are stored
- Cruise name (e.g. mlvl0693)
- Magnitude threshold for cut-off of low magnitude value phase data
- Semi-variance threshold for removal of high semi-variance grid values
- Map grid node spacing (in degrees)

Option 4 - Processing bathymetry image grid and output GMT postscript maps
- Map Number (to give map limits)
- Full directory name where the raw phase data are stored
- Cruise name (e.g. mlvl0693)
- Magnitude threshold for cut-off of low magnitude value phase data
- Semi-variance threshold for removal of high semi-variance grid values
- Map grid node spacing (in degrees)

Prerequisites of these processes are generally:
- The raw phase data (each hourly datafile is approximately 35Mb).
- A file containing map limits (in the current directory and called map*n*.dat, where *n* is the map number).
- A directory containing 4 files: Navigation file (called *cruise_name*.nav) and three propagation datafiles (called raypaths.tab pcorrn5.asc and scorrn7.asc).

Figure 9 Flow-diagram of the processes required to convert the raw digital archived phase data into bathymetry maps



Phase program

This program takes the 35Mb raw data files and converts the phase and magnitude data into range and depth. The raw data file consists of a time code and the four data signals from the upper and lower transducer arrays on both port and starboard sides. 384 frames of data were logged for each ping starting from the end of the transmission (time zero). The frames are spaced at one per set of 16 I/Q samples (i.e. one every 70.18ms), thus the total 384 frames represent a time span of 26.95 seconds. Each frame consists of 4 spectra in the following order :- port upper, port lower, starboard upper, starboard lower. Each spectrum has 32 complex values with the I's and Q's being interlaced, and all values are 4-byte floats in IEEE format. Each frame is thus 4x64x4 = 1024 bytes in size. 16 sensor values and a clock reading were also logged for each frame and these are all written in a single 30,720 byte block following the 384 frames of spectral data for each ping. The data for each ping is prefaced by a 512-byte ASCII header containing the ping number and "time zero" time.

The data was logged in hour long files which have names "melDDDHH.dat", where DDD and HH are the day number and hour at the time the file was started. Each file contains 120 pings of data. The total number of bytes per ping is now 424,448 bytes and is comprised of :-

(1) An ASCII header of 512 bytes.

```
char header[512];
```

(2) 384 blocks of FFT data at 70.18ms intervals (393,216 bytes).

```
struct wrt_block {
        float    p_upfreq[64];
        float    p_lofreq[64];
        float    s_upfreq[64];
        float    s_lofreq[64];
        } file_buff[384];
```

(3) 384 blocks of the corresponding sensor and time data (30,720 bytes).

```
struct sens_blk {
        float scaled_vals[16];      sensors & time @ 14.25 Hz.
        byte timedat[16];
        } sensor[384];
```

The sensor data is written as 4-byte floats and is arranged as follows :-

electronics temp	sensor[0]	°C
spare analog chan	sensor[1]	Volts
mag X	sensor[2]	nTesla
mag Y	sensor[3]	nTesla
mag Z	sensor[4]	nTesla
gyro X	sensor[5]	degrees
gyro Y	sensor[6]	degrees
gyro Z	sensor[7]	degrees
sea temp	sensor[8]	°C
depth	sensor[9]	metres
roll	sensor[10]	degrees
pitch	sensor[11]	degrees
computed heading	sensor[12]	degrees magnetic
tx beam offset	sensor[13]	degrees
spare	sensor[14]	
spare	sensor[15]	

This is followed by 16 bytes of ASCII time data :-

timedat[0]	year	10's
timedat[1]	year	1's
timedat[2]	day	100's
timedat[3]	day	10's
timedat[4]	day	1's
timedat[5]	hour	10's
timedat[6]	hour	1's
timedat[7]	mins	10's
timedat[8]	mins	1's
timedat[9]	secs	10's
timedat[10]	secs	1's
timedat[11]	msec	100's
timedat[12]	msec	10's
timedat[13]	msec	1's
timedat[14]	0	
timedat[15]	clock status - 0x33 = OK	

Having this input data the position due to the FM correlator moving over a 2-second window, the results are seen to slant and thus the first recalculation is to align all the 32 FFT bin returns into a vertical pattern so that responses at a single time period can be correlated. A phase difference is calculated for each of the 32 FFT bins on the upper array and the corresponding lower array values and summed thus giving a total phase difference for each time period in the range -360° to 360°. The correlated phase data for all 32 FFT bins are also taken and converted into values of magnitude for each FFT bin and summed, for all four arrays.

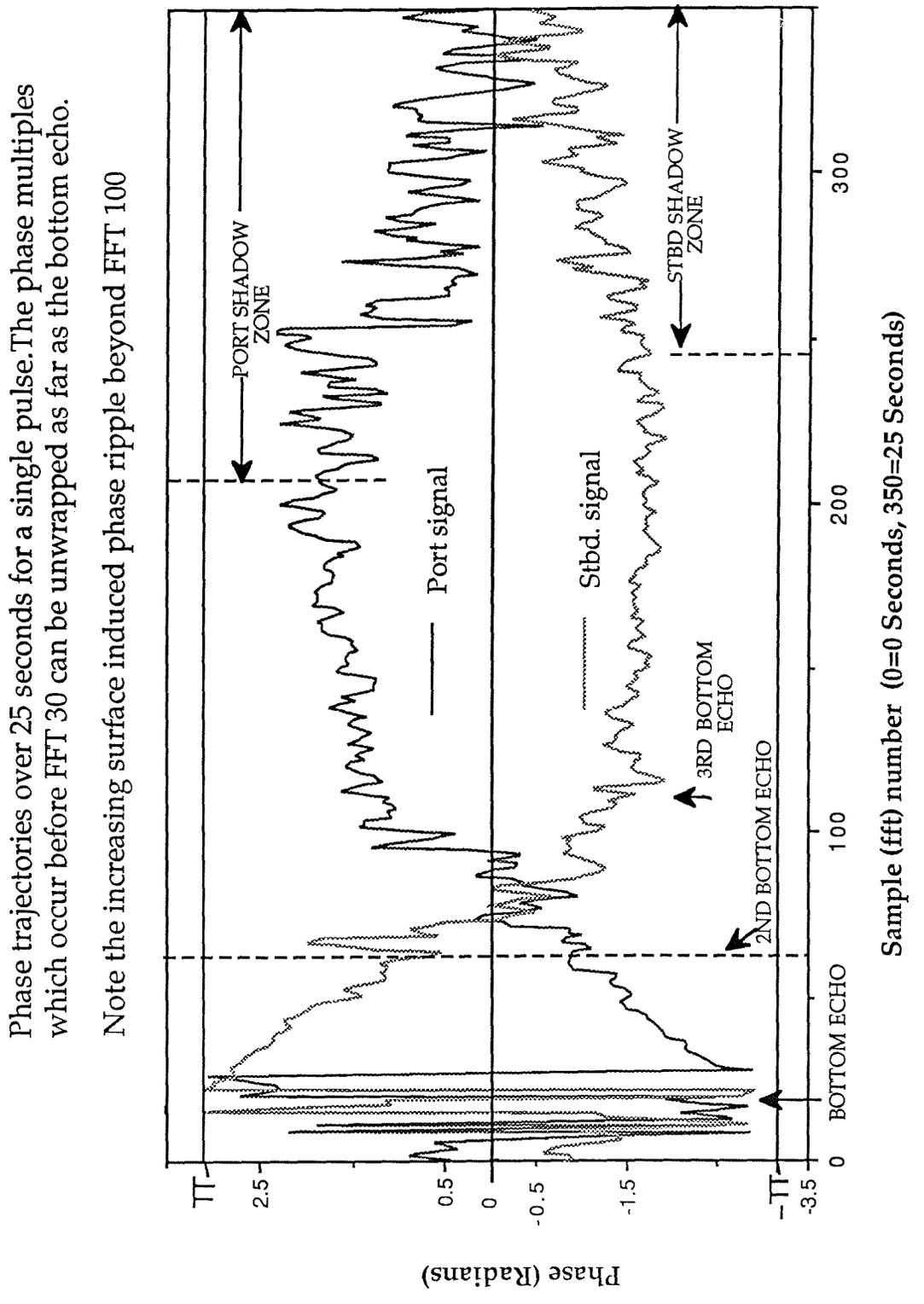
$$\text{Magnitude}_{upper} = \sqrt{\sum_{i=0}^{31} \left(I_{upper} \cdot I_{upper} + Q_{upper} \cdot Q_{upper} \right)}$$ for each time period in the upper array,

$$\text{Phase difference} = \tan^{-1} \left(\frac{\sum_{i=0}^{31} \left(Q_{upper} \cdot I_{lower} - Q_{lower} \cdot I_{upper} \right)}{\sum_{i=0}^{31} \left(I_{upper} \cdot I_{lower} + Q_{upper} \cdot Q_{lower} \right)} \right)$$ for each time period and side.

These values are then adjusted so that a continuous phase difference curve is created along the swath. Firstly the water column noise must be removed from the phase curve by finding the first bottom echo and this is done in two ways: either finding a change in phase sign and then the local maximum power gradient or by solely finding the maximum power gradient. If the port and starboard side differ by more than 280ms then the port bottom echo takes preference as it is usually the stronger reflection. The phase difference values are then unwrapped from an arbitrary point discarding everything before the phase change or bottom echo. Figure 10 shows the unwrapped phase curve.

If the smoothed magnitude values for the whole swath fall below a threshold value then the whole ping is assumed to be sea noise (where transmission was switched off) and thus phase values are meaningless. A second test removes individual phase values if the magnitude values for upper and lower arrays fall below the threshold value. The threshold value is provided by the user, generally obtained from looking at the far range image pixels and/or shadow zones. It was found that swaths with no transmission ping, and thus only low level sea noise, did not provide accurate acoustic reverberation levels to be used for this threshold.

Figure 10 An example of the unwrapped phase curves for port and starboard sides for a single ping against FFT number



Converting from phase difference (α) to angle of insonification from the horizontal (θ) is relatively straight forward using the formula:

$$\alpha = \frac{2\pi \cdot f}{c} \cdot D \cdot \sin(\theta - \theta_0)$$

where $c = 1500\ ms^{-1}$, $f = 6287.5$ hertz (starboard frequency),
$D = 0.1778$ metres (the distance between upper and lower arrays)
and $\theta_0 = 20°$ (the angle of depression of the boresight).

This gives a range of angle of depression of 0° to 90°. Converting from angle of depression and time to depth and range can be done initially by simple geometry. However to improve on this simplistic model a velocity profile of the water column is supplied in file raypaths.tab, which greatly improves the accuracy in estimates of the water velocity (c) and takes into account refraction in the water column, this being extremely important for low angles of depression (far range). Two angle correction files (pcorrn.asc and scorrn.asc) for the port and starboard arrays are also provided for better raypath modelling.

The next stage of this program is to sort the FFT bins by range value to make sure that the range values are continuously increasing within a threshold of 2.5 bins from the original order. This is achieved by smoothing the order values with a window whose full width is 4 bins. A full explanation of the smoothing method is given in Numerical Recipes.

The final step of the program is to output the data to an ASCII data file with range, ping number and depth. Negative ranges represent the port side and the ping count varies from 1 to 120. The first ping time is given in the 512 byte ASCII header. All depths are in metres. This has to be done for each individual hourly datafile. The output file has the type .xyz. A full listing of the program is given in Appendix A.

Merge program

The next stage of the processing is to add geographic co-ordinates to each depth value. To do this the ping time must be correlated with the ship navigation time and consequently the ship's geographic position. The ship's navigation file is provided in the ASCII DXFMT format (also called merge-merge). This gives the date, time, latitude and longitude, every two minutes. It is a fixed format with 80 characters per record and is essentially fixed at:

Cruise_name	8 Characters
Timezone	5 Characters
Year	2 Digit Integer
Month	2 Digit Integer
Day	2 Digit Integer
Space	1 Character
Hour	2 Digit Integer
Minutes	2 Digit Integer
Latitude	9 Digit Real (4 Decimal places)
Longitude	9 Digit Real (4 Decimal places)
Space	10 Characters
Uncorrected depth	5 Digit Integer
Corrected depth	5 Digit Integer
Space	3 Characters
Total Magnetic field	5 Digit Integer
Residual Magnetic field	5 Digit Integer
Free Air Anomaly	5 Digit Integer

As the vehicle travels behind the ship by about 350 metres of cable and the average speed is about 8 knots the vehicle can be said to follow the ship by 1.5 minutes. This time lag is added to the phase data's time. Once the ship navigation time and this new phase data time are correlated (using linear interpolation between data points if required) the position of the vehicle can be found. The heading of the vehicle is then calculated from the ship's track and used to find the azimuth of the swath from which range can be converted into latitude and longitude.

This also has to be done for each individual hourly datafile. The output file has the type .llz and simply consists of a list of latitude, longitude and depth points.

Choosing and creating a grid

Once all the phase data files have been converted into .llz files the next stage of processing is to define a map area, projection and scale. The *createmap* command makes a small datafile consisting of minimum and maximum latitude and longitude, projection code and standard latitude and longitudes. Once the map number is defined (for example 2) the filename would be map2.dat, which could look like for example:

-27.3333	Minimum Latitude (North is Positive)
-26.3333	Maximum Latitude (North is Positive)
-107.5	Minimum Longitude (East is Positive)
-106.5	Maximum Longitude (East is Positive)
5	Projection Code (1=UTM, 5=Mercator)
-30.0	Standard Latitude (North is positive)
0	Standard Longitude (East is positive)

The first line of each .llz datafile is read and if this point is within the map limits or within a third of a degree of any of the limits the filename of the .llz datafile is noted. This therefore creates a list (called map2.cho in this example) of the all the relevant datafiles required for the map area, including points that are near the map limits.

Collation of all the points is executed by the *gridswath* command. This takes the .cho datafile list and reads each .llz file into a dynamic array. The size of this array is defined by the map limits and the grid resolution increment. The increment factor can be defined at any level, for example for approximate 100 metre resolution an increment factor of 0.0009 degrees is defined. It can therefore be seen that this array has equal degree spacing rather than equal ground spacing and thus the map will require map projection at a later stage. If a relatively large increment is defined it is likely that several depth values will fall within the grid point size. When this occurs the datapoints are stacked and the median extracted and a single depth value output.

The output file has the type .xyz and simply consists of a list of pixel row, pixel column and depth points, together with a header consisting of number of rows, number of columns and pixel spacing.

Kriging

The creation of the grid in the previous stage has now produced individual values for each grid point. However due to the yawing of the vehicle it is often possible that some pixels have no depth value and thus some type of interpolation is required. In addition far range pixels on parallel ship tracks may not overlap and thus create a gap between swaths.

The geostatistical method of kriging has become increasingly well known and widely used because of its many desirable features. Unlike most automatic contouring or interpolation methods, kriging is a two stage process. The first stage determines the spatial structure of the data. This is done by calculating the semi-variograph from the data using relative position and value information.

Let $Z(x)$ be the value at position x and $Z(x+h)$ the value at a distance h from x, then the semi-variograph is defined as:

$$\gamma(h) = \sum^{n} \frac{[Z(x) - Z(x+h)]^2}{2n}$$

This produces a graph of h against γ which can give the user a key on the continuity of the data. The figure below shows the shape of a semi-variograph most common in data of this sort. The interpretation is that data values which are close together in space area also close in value, however as the sample positions become more distant, the difference in values increases. At a certain distance, shown as A (Figure 11), the semi-variance remains constant, as the sample values become independent of one another.

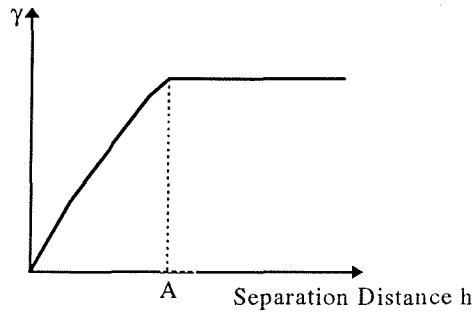


Figure 11 Idealised semi-variance model for kriging process

The second stage of kriging uses the modelled semi-variograph and the original data. The output comes in two forms: a grid of interpolated values and a grid of standard deviation of the interpolation. These are effectively maps of depth and reliability.

The program *krigswath2* reads the gridded data and divides the area into small tiles of 50 pixels square with an overlap of tiles of 6 pixels. Each tile is then kriged independently using the pre-defined semi-variograph. The overlap is required as the kriging method has a search radius and thus edge effects must be removed. The depth and standard deviation values can then be output in exactly the same format as the .xyz grid except the output files have the type .krg and .std respectively. As the standard deviation values give an indication of depth quality these can be used to threshold the depth values. This threshold value can be entered at the *krigswath2* command line, generally to remove all points where depth from calculations using phase data is uncertain, be it due to no data points nearby or high variance.

The kriging algorithm KRIG2 comes from the UNIRAS software package produced by UNIRAS Ltd. and is used under license. Many similar kriging algorithms are available elsewhere and therefore may be needed to replace the functionality. Appendix B lists the krigswath2 program together with the relevant information about KRIG2.

Final filtering and output

Final filtering is only occasionally required or wanted depending on the parameters used in processing. What is required is a quick infill of the blank spaces for easier interpretation. To do this a large smoothing filter is applied using a box car kernel (lpfz) but only changing pixels that have a zero value. In this way any zero pixels are smoothly interpolated between pixels of known depth and removing large gaps in the final output. Contours have not been calculated as these can be plotted using a variety of packages including GMT. A command xyz2grd is included to convert to GMT grid format and similarly xyz2whips converts to WHIPS format.

Once the format is converted into either GMT or WHIPS formats output is relatively straight forward. Using GIS and image processing techniques the data can be further processed, for example true slant-range correction of GLORIA imagery, slope analysis and data co-registration.

Example area:

The swath bathymetry was first successfully deployed in the USGS GLORIA III vehicle by IOSDL staff supporting a pair of back-to-back cruises in the R/V Melville in the area of the East Pacific Rise. The principal investigators were Dick Hey and Dave Naar respectively, supported by the NSF, and the cruises ran from early February 1993 to mid April 1993. The example area is a one degree square. There is a 93% areal coverage with GLORIA swath bathymetry data done in approximately 32 hours. SeaBEAM bathymetry was also collected contemporaneously but coverage was only 34% (47% if including other ships tracks). The data were collected on the second leg (cruise mlvl0693) and this area was chosen as it was the only SeaBEAM data available and being a relatively typical coverage scenario.

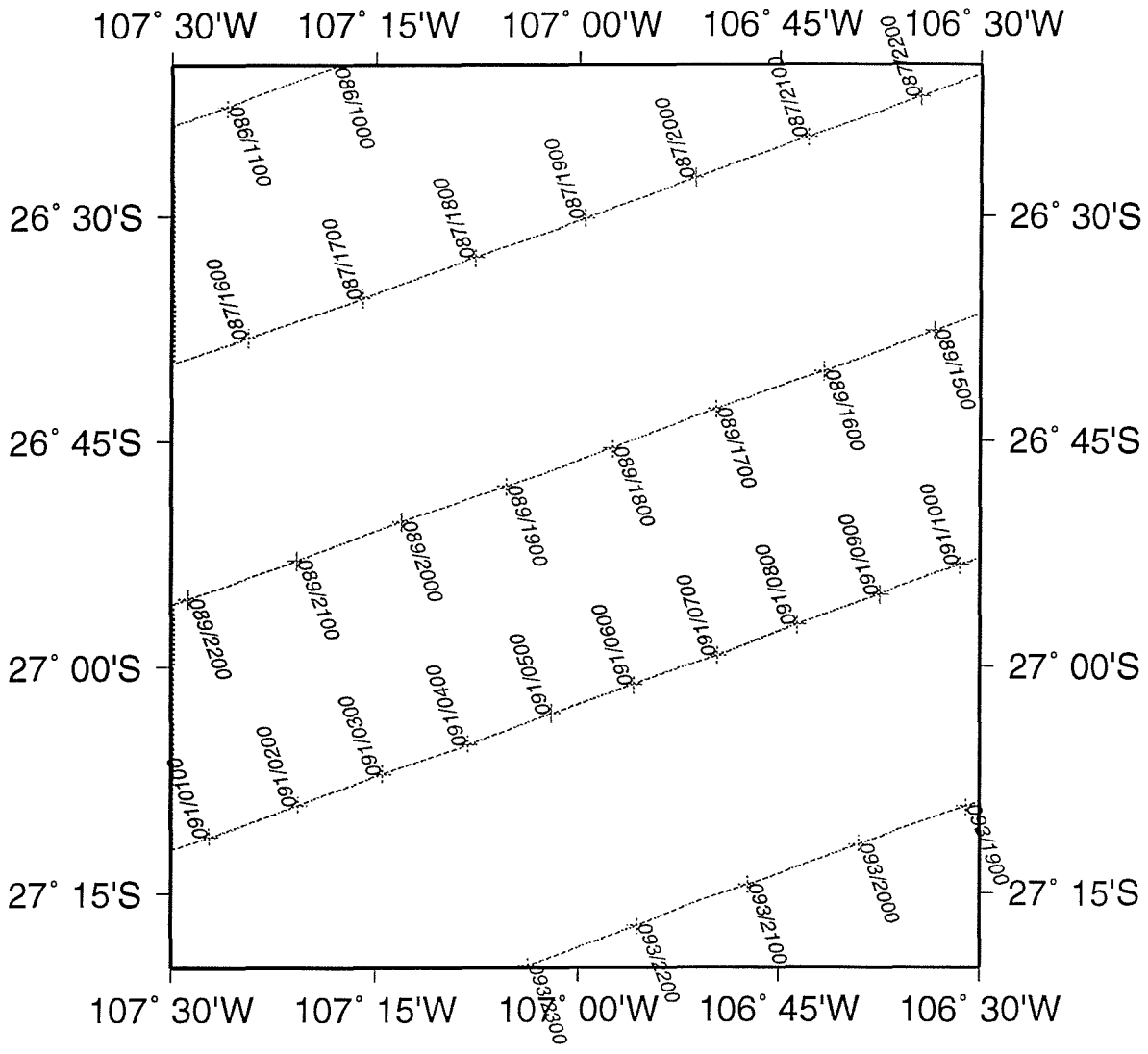
Track Chart

The track chart (Figure12) shows the ship's track with hourly time marks together with Julian day numbers. There are five tracks crossing the area at an azimuth of 070°. This chart was produced using the GMT software and the following commands:

```
psbasemap -R-107.5000/-106.5000/-27.3333/-26.3333 -Jm1:1000000 -B0.25 -X2 -Y1 -K >
        mlvl0693.ps
psxy mlvl0693.gmt_nav -R -Jm -B -M -W1/127/127/127 -O -K -: -V >> mlvl0693.ps
psxy mlvl0693.tim_nav -R -Jm -B -M -W1/127/127/127 -Sx0.1 -O -K -: -V >> mlvl0693.ps
pstext mlvl0693.tim_nav -R -Jm -B -G0/0/0 -O -: -V >> mlvl0693.ps
```

where mlvl0693.gmt_nav contains only the relevant latitude and longitude data for the area, and mlvl0693.tim_nav contains the hour marks' latitude, longitude and ship's heading data.

Figure 12 Track Chart of Melville 0693 showing hourly time and date marks



GLORIA backscatter imagery

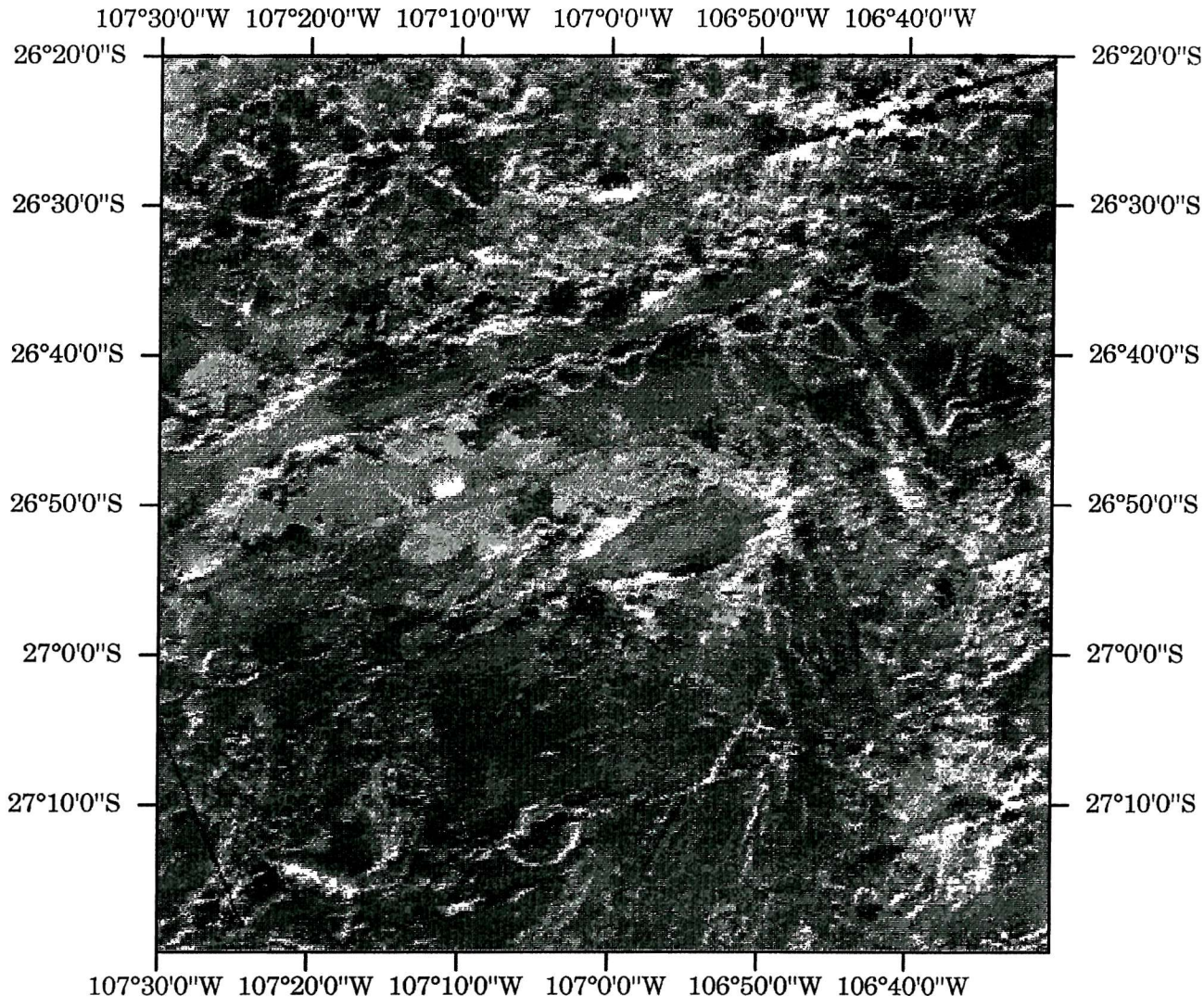
The GLORIA backscatter imagery (Figure 13) has been created using the WHIPS software together with the .cdf raw datafiles. The imagery datafiles are recorded separately from the phase data and are generally held in 6 hour files. The raw imagery has 992 samples (pixels) across-track and one swath every 30 seconds. Each pixel recorded at 8 bit (256 grey levels) with low backscatter having value 0 and printing black and with high backscatter having value 255 and printing white.

The imagery has been processed in the standard way using the automatic processing scripts available in WHIPS. This applies several processing techniques such as slant-range correction, filtering of noise, removal of line dropouts, shading (removal of beam directivity) and geographic registration. Further description of these processes are provided in Chavez (1986) and Searle et al. (1990). Source programs and scripts can be viewed from within WHIPS.

The final backscatter image shows the mosaic of all the GLORIA passes. Unfortunately the water column structure was such that, during acquisition, the maximum range was severely reduced from the 22.5km to about 14km. The ship's tracks were spaced at about 30km and thus a gap is evident between some far range imagery pixels. As the bathymetry alters this gap widens and narrows and in some places completely disappears. This is seen in the image as a smooth grey area which, at first glance to an in-experienced interpreter, looks similar to a ship's track. Fortunately an experienced geological interpreter can produce a coherent interpretation interpolating over the gaps. However the imagery itself does appear more complicated than necessary. If the water column structure had been more favourable coverage would have been better and the image much cleaner. These more favourable conditions were found in this area during the GLORIA imagery cruise Charles Darwin 35 which ran from mid October 1988 to late November 1988, where a full 22.5 km range was achieved.

This water column structure is also a factor to consider when looking at the GLORIA bathymetry data. It is estimated that a greater than 15km range would normally be achieved in favourable water column structure conditions.

Figure 13 GLORIA backscatter imagery created by WHIPS 'process_gloria'



Directory structure and list of data files

The directory structure of the datafiles is simple. Each datatype is stored in separate sub-directories. It is possible to have symbolic links to other directories and disks if required, but this can lead to confusion in not knowing exactly where the data resides. There are three main sub-directories required initially:

- cdf for the raw GLORIA backscatter imagery 6 hourly files (in NetCDF/WHIPS format)
- raw for the raw GLORIA phase data hourly files
- nav for navigation files and water column correction files

During processing the following directories are required and created automatically:

- xyz for the GLORIA phase data of ping, range, depth and magnitude datafiles.
- llz for the GLORIA phase data of latitude, longitude and depth datafiles.

Finally, creation of a map requires its own directory. This directory can be placed anywhere, such as a working area rather than a data area. However again to remove confusion it is suggested that a map sub-directory is created in the same directory as the above sub-directories. Thus the structure may look like this:

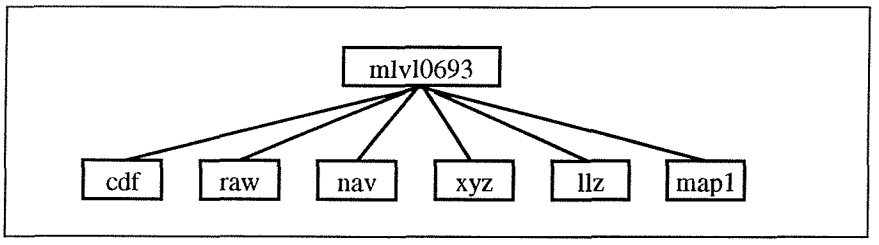


Figure 14 Directory structure of raw and processed data

The contents of the 3 data directories are listed below; the files listed are the ones relevant for the example area. This is because storing the full complement of the cruise data requires several Gigabytes of diskspace.

cdf directory contents:

mlvl0693.index	(list of start and end times of all cdf files)
mlvl0693p17.cdf	
mlvl0693p18.cdf	
mlvl0693p19.cdf	
mlvl0693p23.cdf	
mlvl0693p24.cdf	
mlvl0693p31.cdf	
mlvl0693p32.cdf	
mlvl0693p37.cdf	
mlvl0693p38.cdf	
mlvl0693p47.cdf	
mlvl0693p48.cdf	
mlvl0693p49.cdf	

nav directory contents:

mlvl0693.nav	(navigation file in DXFMT format)
pcorrn5.asc	(port phase correction datafile)
raypaths.tab	(depth/velocity correction table)
scorrn5.asc	(starboard phase correction datafile - version 5)
scorrn7.asc	(starboard phase correction datafile - version 7)

raw directory contents:

mel08219.dat	mel08918.dat	mel09109.dat
mel08608.dat	mel08919.dat	mel09110.dat
mel08609.dat	mel08920.dat	mel09318.dat
mel08610.dat	mel08921.dat	mel09319.dat
mel08611.dat	mel08922.dat	mel09320.dat
mel08714.dat	mel09100.dat	mel09321.dat
mel08715.dat	mel09101.dat	mel09322.dat
mel08716.dat	mel09102.dat	mel09323.dat
mel08717.dat	mel09103.dat	mel09400.dat
mel08718.dat	mel09104.dat	mel09523.dat
mel08719.dat	mel09105.dat	mel09600.dat
mel08720.dat	mel09106.dat	mel09601.dat
mel08721.dat	mel09107.dat	
mel08722.dat	mel09108.dat	

This list is incomplete because of difficulty experienced reading an archive tape (tape error). The second archive tape was not easily accessible and thus the list does not contain the following files:

mel08917.dat
mel08916.dat
mel08915.dat
mel08914.dat

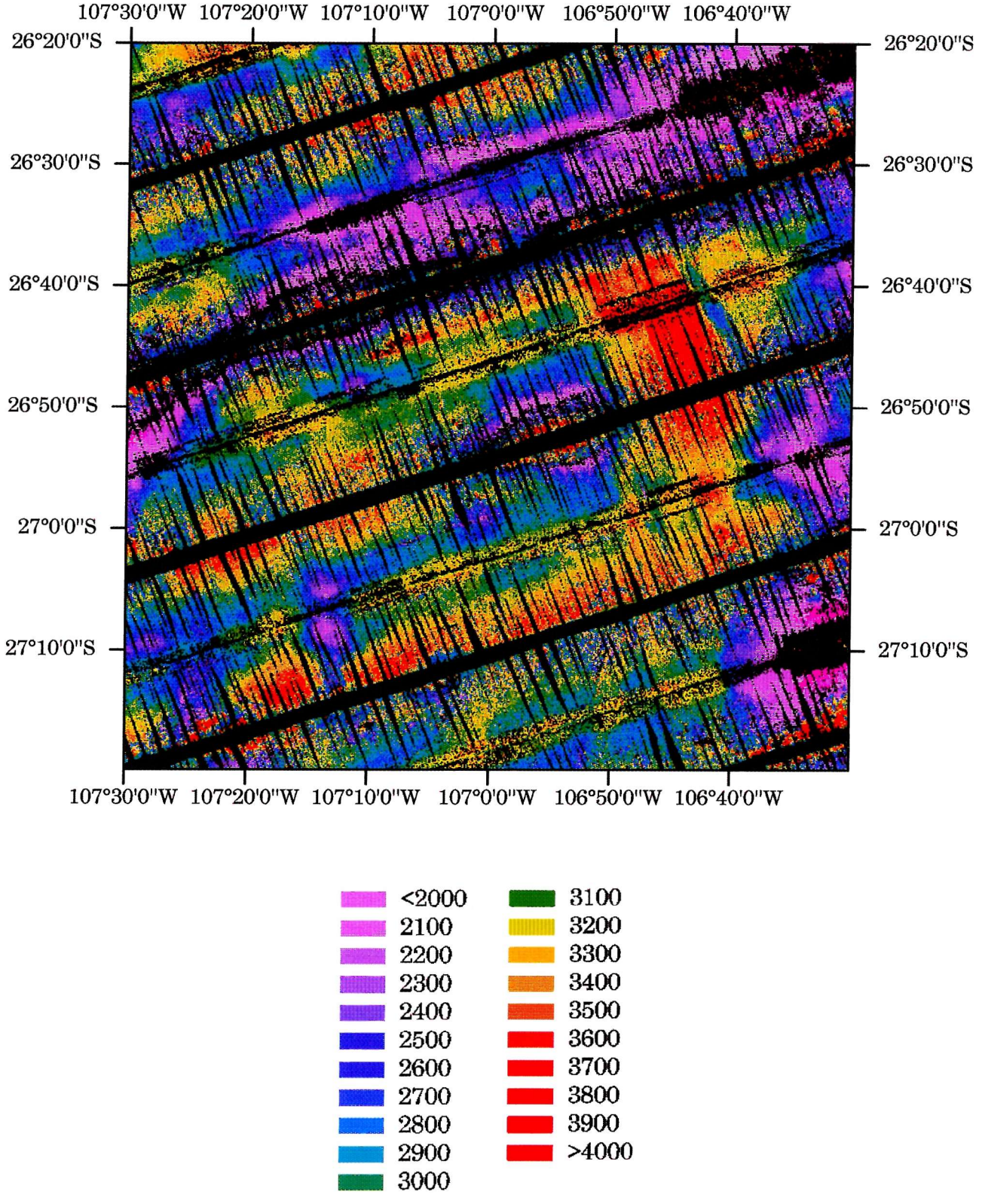
Raw phase bathymetry data

This map (Figure 15) shows the first iteration of the bathymetry. The raw phase data has been converted into times and angles and thus into depth and range. The navigation data has then been added giving ships position and ping azimuth and thus giving geographic position to every depth point. A grid has then been created with a 0.0018 degree spacing (equivalent to 200 metres). Multiple values for grid points have been removed using a median filter. Finally the grid has been imported into ERDAS Imagine which has simple map composing capabilities for final hardcopy.

The bathymetry values vary from less than 2000 metres to over 4000 metres. Several ridges and seamounts are visible rising up to 1000 metres above the relatively flat seafloor (at between 2900 and 3200 metres). A large rhomboidal depression about 800 metres below the average seafloor is seen (centred at 26° 45'S and 106° 45'W) running NNW-SSE. Its edges are distinctly defined and thus it may be tensional rifted block being perpendicular to the local spreading direction and tension field.

The black areas show pixels with no data. Towards far range the bathymetry values tend to have a black zone before reaching maximum range where the values start to increase This black zone is due to the shadow zone created by the water column structure and depth (as mentioned above). At maximum range some depth values are visible and these must be ignored as they are noise. Filtering with respect to low magnitude should remove these values. A good example of this is seen at 26° 43'S and 107° 20'W

Figure 15 Initial colour-coded bathymetry map created from raw phase data - water column corrected but not thresholded or interpolated



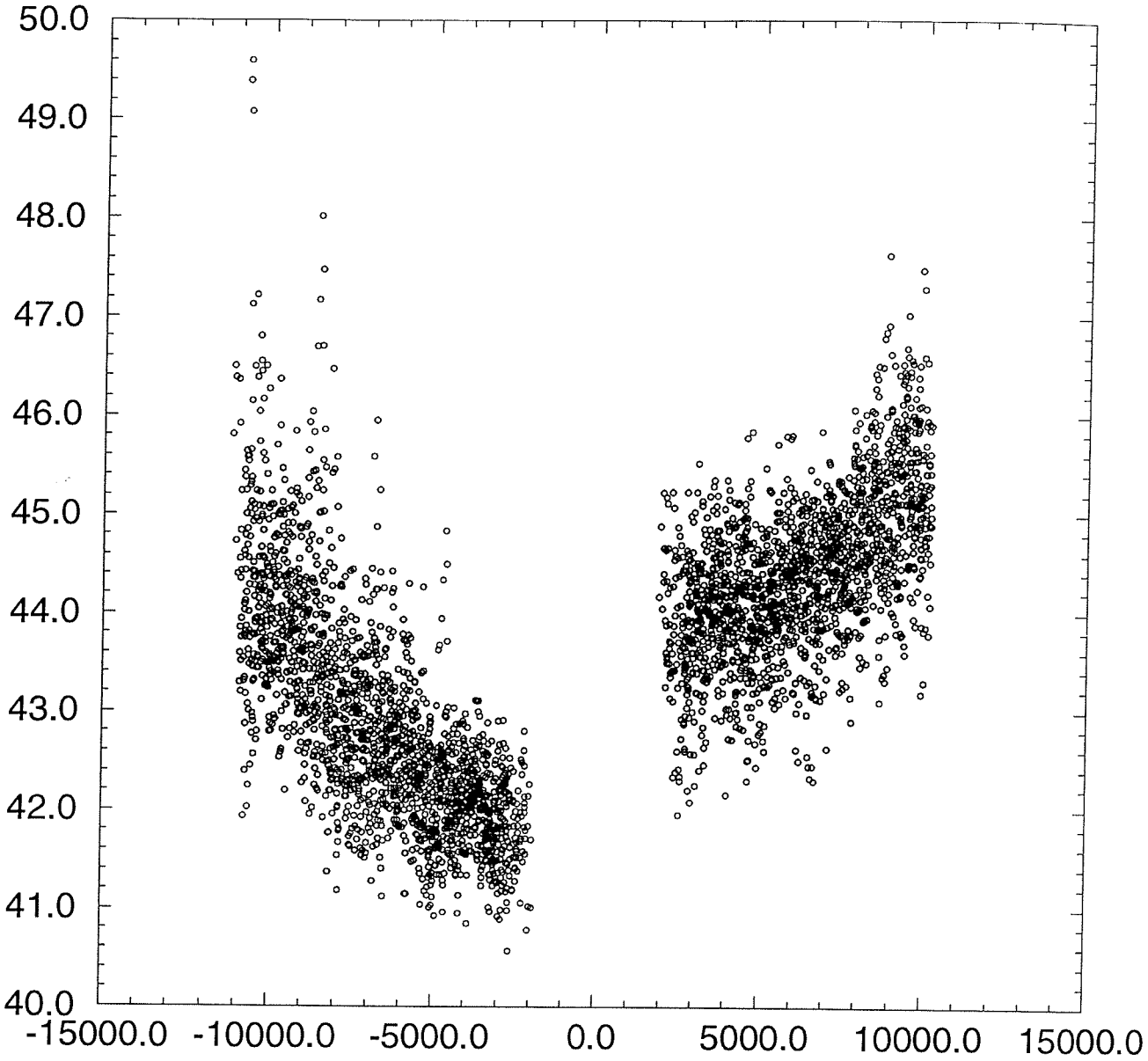
Removal of low magnitude data:

When processing the phase data three values are calculated for each pixel, namely phase, time and magnitude. Phase and time combine to give depth and range. The magnitude data are scaled into a manageable form. As is discussed in the previous section filtering of low magnitude phase data is required to remove shadow zones and noise. On imagery this low magnitude data is generally printed as black and usually self-evident and a threshold easily found. The magnitude data from the phase data appears to be less coherent and thus less easily found. However with careful correlation between the two datasets a phase magnitude threshold can be calculated.

Scattergram of sea-noise magnitude with range

Figure 16 shows the range from the vehicle (metres) against phase magnitude (scaled) when no ping has been transmitted. This therefore gives a measure of the ambient ship and sea-noise received by the arrays. As expected the graph shows increasing noise with range - a factor dependant on the time varied gain applied on the received signal. The data is taken from the mel08219.dat phase datafile for pings 110-120. At this time the vehicle was turning and transmission was turned off. As can be seen values are very small, all less than 50.

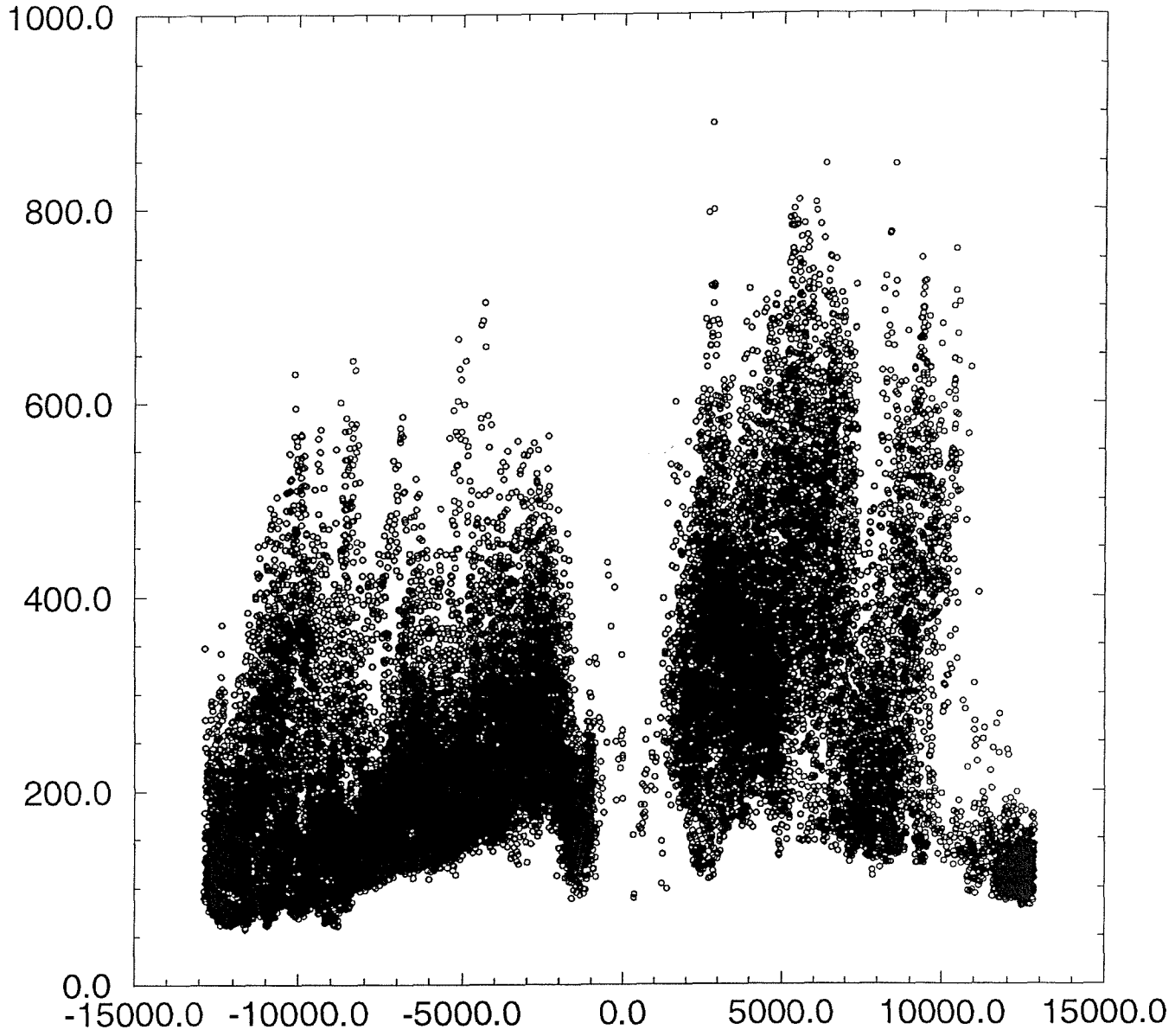
Figure 16 Scattergram of sea-noise magnitude with range (no transmission)



Scattergram of magnitude with range

Figure 17 shows the range from the vehicle (metres) against phase magnitude (scaled) during 120 pings have been transmitted and received. This therefore gives a measure of the sensitivity of the data received by the arrays. The data is taken from the mel08717.dat phase datafile for pings 1-120. It can be seen that the values vary between 50 and 900. Variation across track is 'm' shaped, which mirrors the variation of the GLORIA backscatter imagery envelope, commonly seen in the shading correction. The values are much higher here than in the previous scattergraph as reverberation in the water column increases the magnitude considerably. This therefore removes consideration of ambient sea-noise and moves the threshold into the reverberation levels (about 100 higher).

Figure 17 Scattergram of phase magnitude with range during transmission



Correlation of imagery with phase magnitude

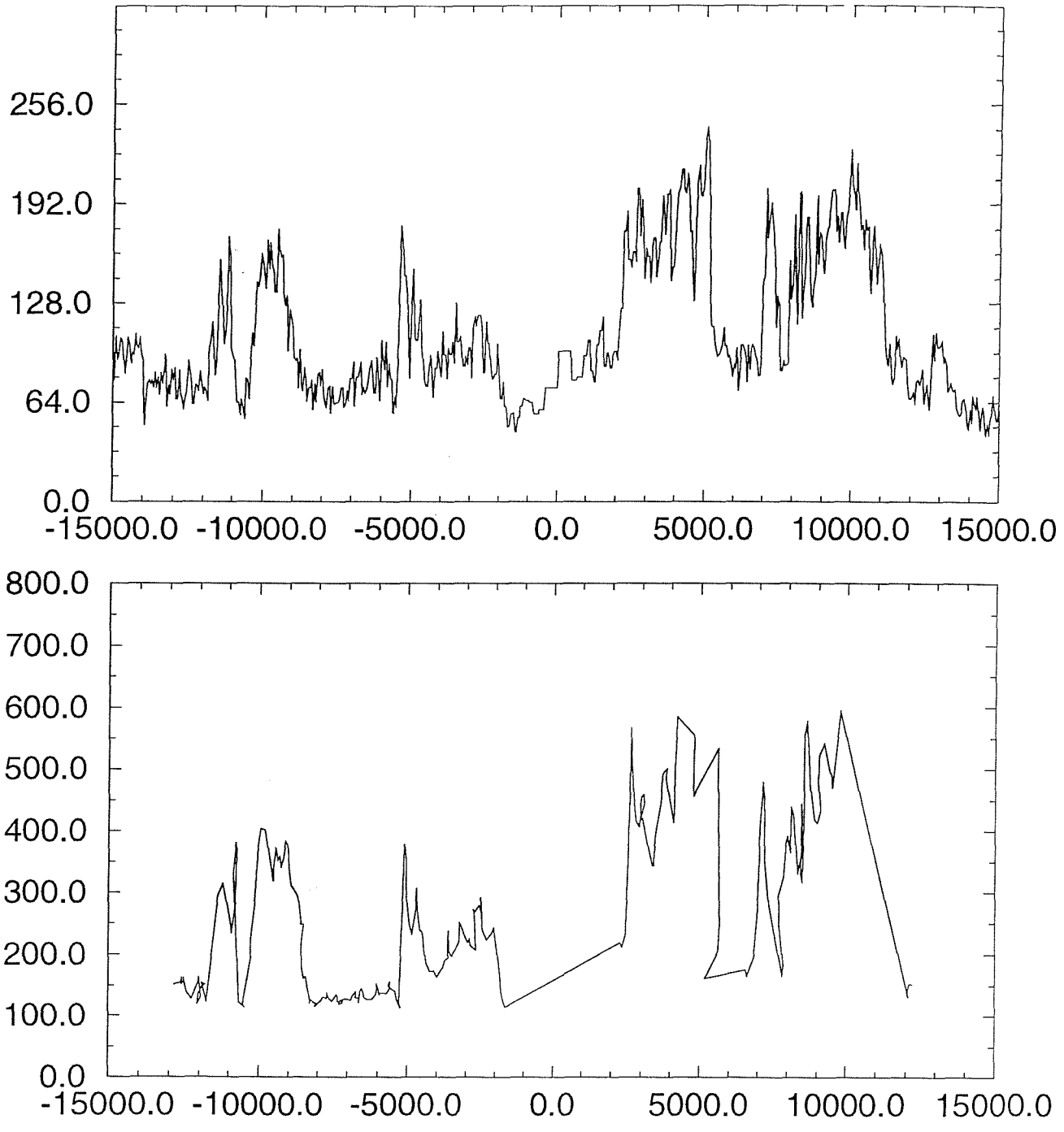
The upper graph on Figure 18 shows one swath of GLORIA imagery (taken from mlvl0693p23.cdf line 573). Magnitudes range from 0 to 255 (being 8 bit imagery, albeit with a logarithmic compression). The lower graph on Figure 18 shows a single line of magnitude values calculated from the phase data for the corresponding time (taken from mel08717.xyz line 80). The magnitudes on this graph vary from 125 to 600. The magnitude (from phase) data has been ordered in increasing time but plotted in increasing range. Overlaps can therefore be seen on the graph but can be ignored.

Viewing the two graphs it is evident that their shape is very similar and the two graphs can be correlated. This has been done and gives a high correlation coefficient of 0.958 and a simple linear conversion equation:

$$\text{Phase Magnitude} = \frac{\text{Imagery Magnitude} - 20.6}{0.354}$$

Using the imagery (before shading correction) a magnitude threshold value of 80 was chosen for shadow detection. This converts into a phase magnitude of 167.8

Figure 18 Correlation of phase magnitude and GLORIA backscatter imagery magnitude for one ping

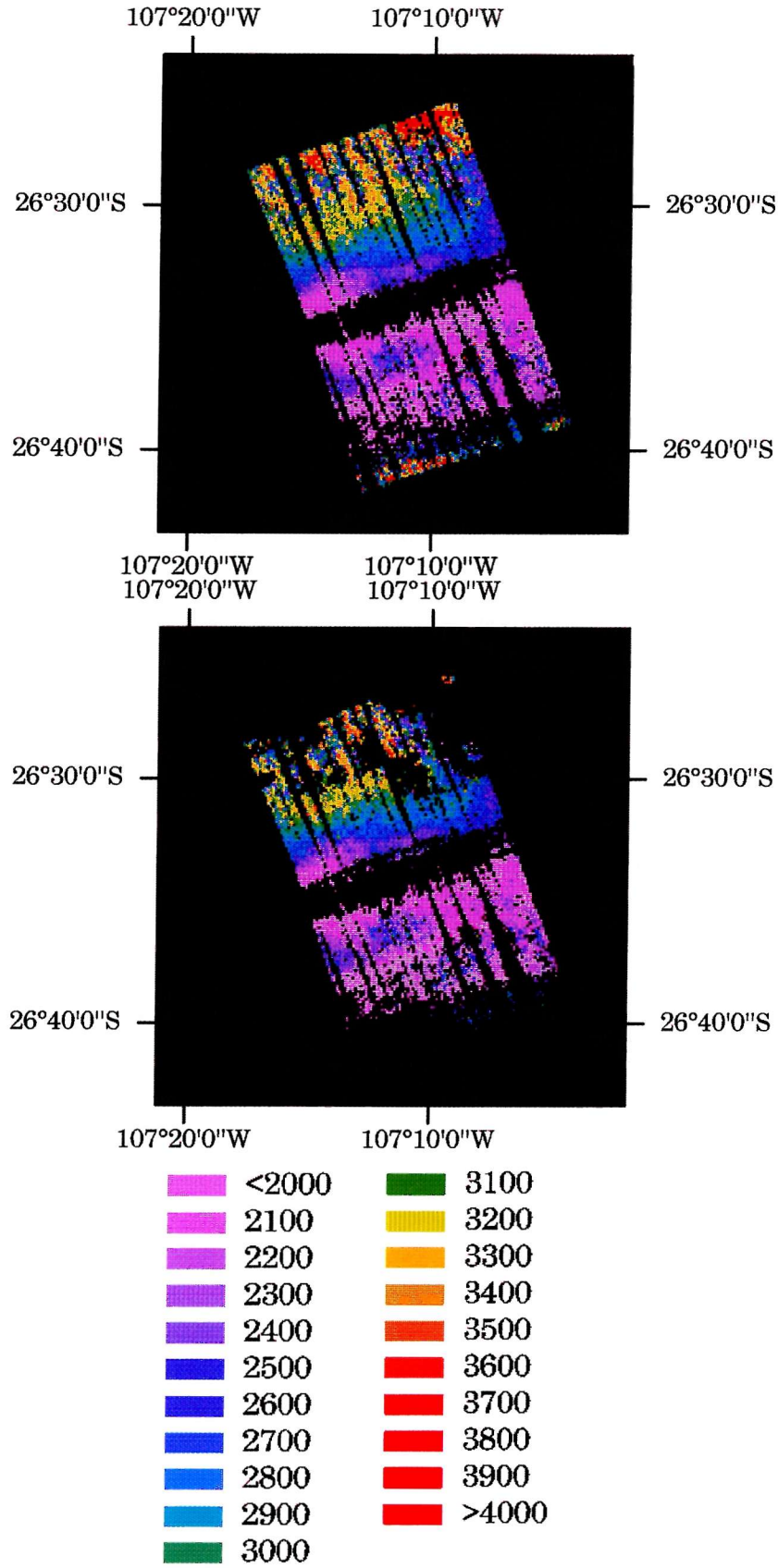


Bathymetry with removal of low magnitude values

The two maps in Figure 19 show the scenarios of the phase bathymetry with and without the removal of low magnitude values on a single hour of phase data (mel08717.dat) with the ships track running 070° across the centre. The magnitude threshold value applied on the lower map is 167.8, as calculated previously. It can be seen that the upper map (before thresholding) has many more bathymetry values than after thresholding. Nearly all the far range depth values have been discarded as being of low magnitude and thus unreliable. The shadow zone (seen on the starboard side as a black line) now represents the maximum achievable range for bathymetry data. All remaining points now have higher confidence and interpolation may now begin.

Some mid-range areas unfortunately now appear to have no bathymetry values as they have been removed by the low magnitude threshold. This is due to the seafloor having a low backscatter coefficient which in turn is due to bottom type, probably silty mud. The GLORIA imagery shows this area to be virtually featureless and thus is indicative of a flat ponding sediment. Interpolation will therefore fill this area with bathymetry values most adequately, without loss of accuracy.

Figure 19 Phase bathymetry before and after the removal of low phase magnitude values on a single hour of phase data



Kriging

The kriging process is in two stages. Firstly the semi-variance must be calculated, this calculates the variation in bathymetry values with distance from the pixel. Secondly using this graph the kriging interpolation calculated producing two outputs:

- The interpolated map of bathymetry values
- A map of corresponding semi-variance values

Semi-variance of bathymetry values

Figure 20 shows several lines of semi-variance for different azimuths between 0° and 90°. The X axis is in units of 200 metres away from the 'central' pixel and the Y axis is the variance of the bathymetry between the 'central' pixel and the 'remote' pixel. It can be seen that the further you go from the central pixel the greater the variance. Also evident is that azimuth direction is not a factor affecting the semi-variance.

The final semi-variance model used was:

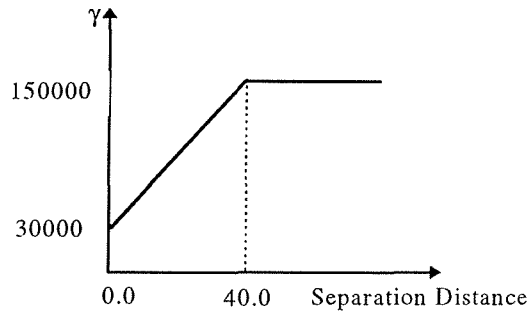
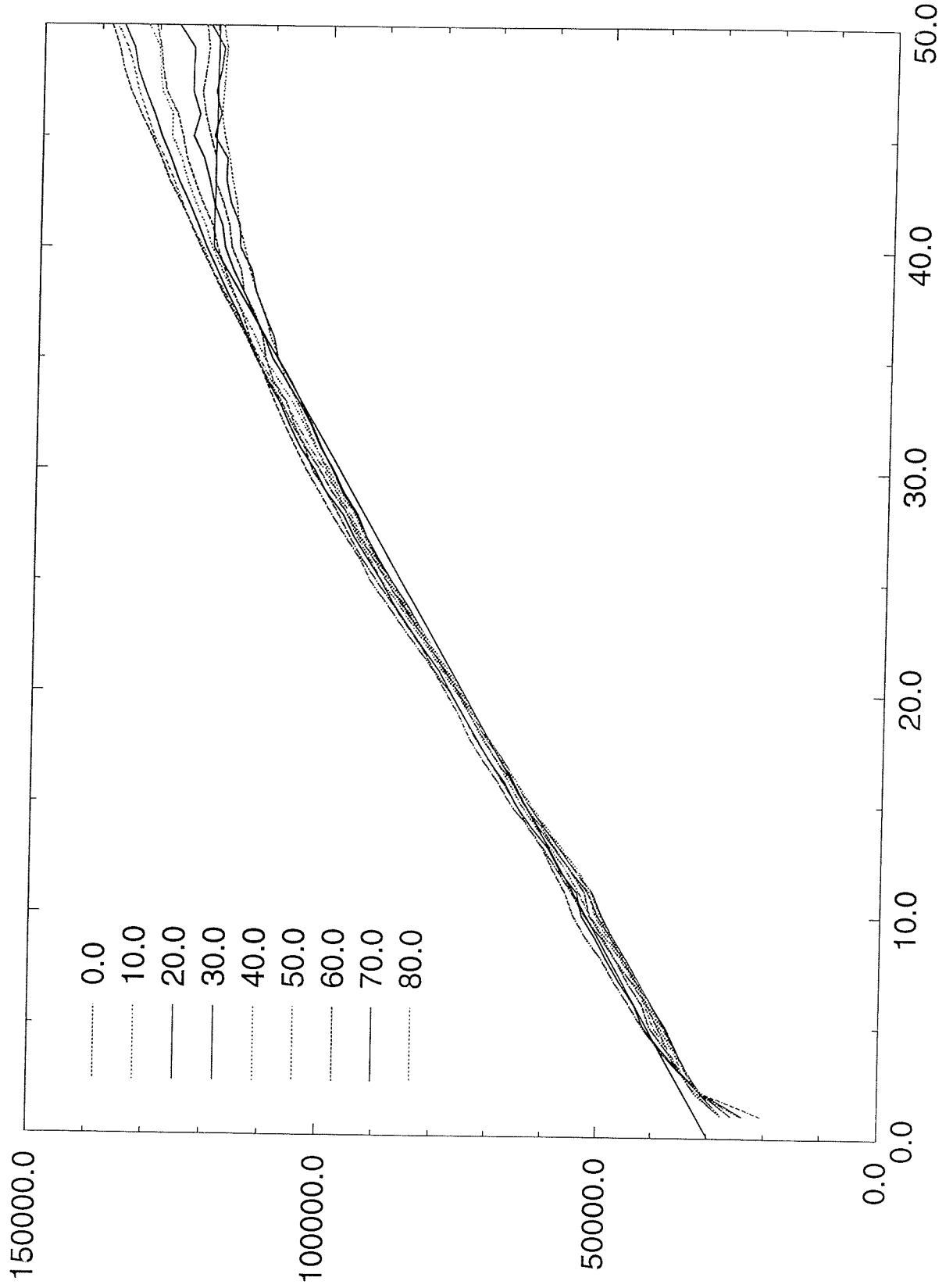


Figure 21 Modelled semi-variance values using in kriging the bathymetry map

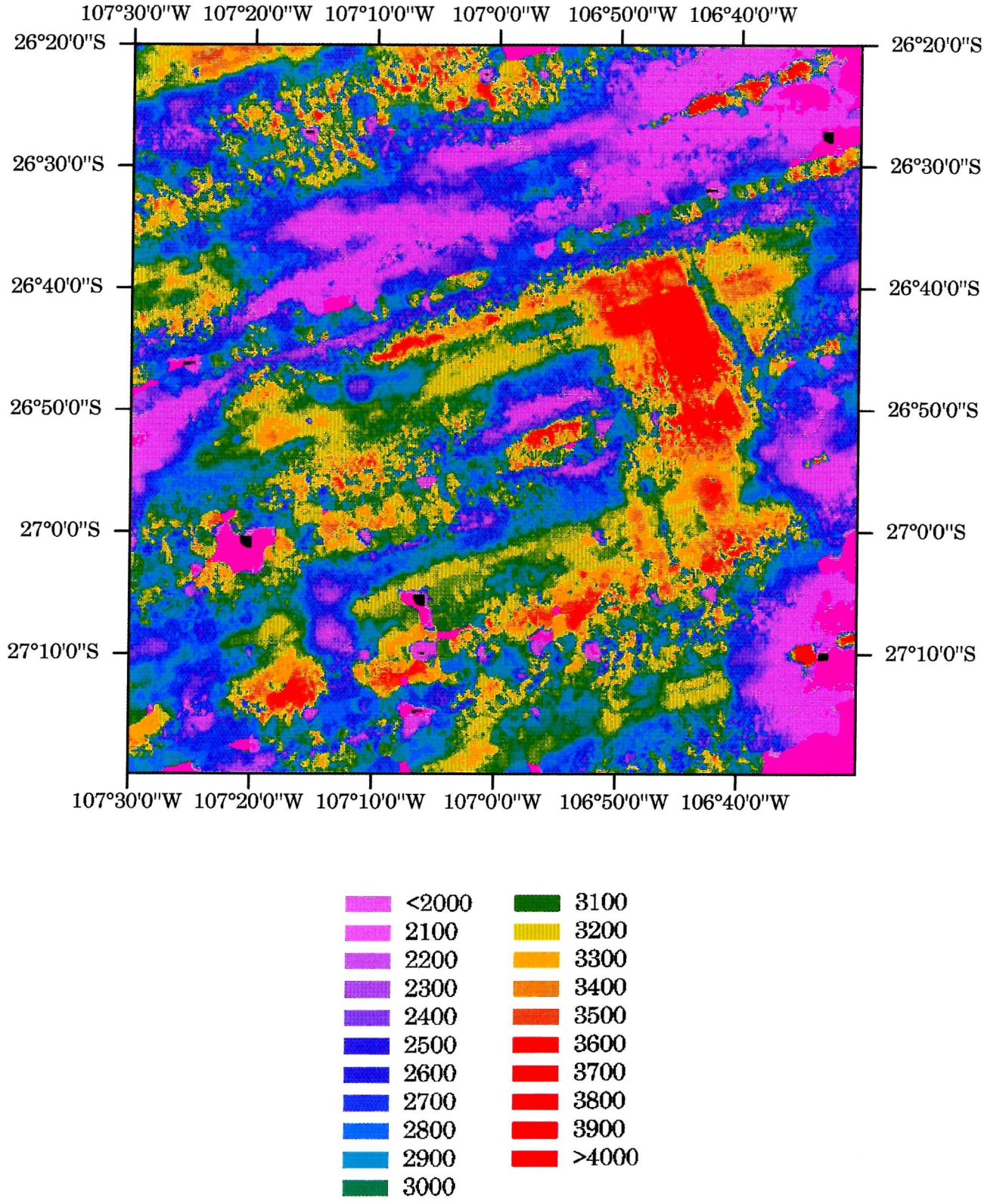
Figure 20 Semi-variance diagram for several different azimuths created from the phase bathymetry map



Kriged map of bathymetry

Using the semi-variance model shown in Figure 21 the new bathymetry map (Figure 22) can now be calculated, to interpolate over any data gaps. The result is excellent, especially with all small gaps between individual swaths are filled. Gaps between adjacent swath, distances of up to 3 km, have also been filled. In most cases this is satisfactory though some seem unduly biased by noisy far range depth values which have crept past the low magnitude threshold. As mentioned in the list of datafiles 4 hours of data could not be retrieved from the archive and have thus had no low magnitude threshold applied and accounts for the distinct line running from 26° 40'S and 107° 10'W to 26° 30'S and 106° 30'W. Reprocessing with the 167.8 magnitude threshold should remove this.

Figure 22 Kriged map of bathymetry



Map of semi-variance values

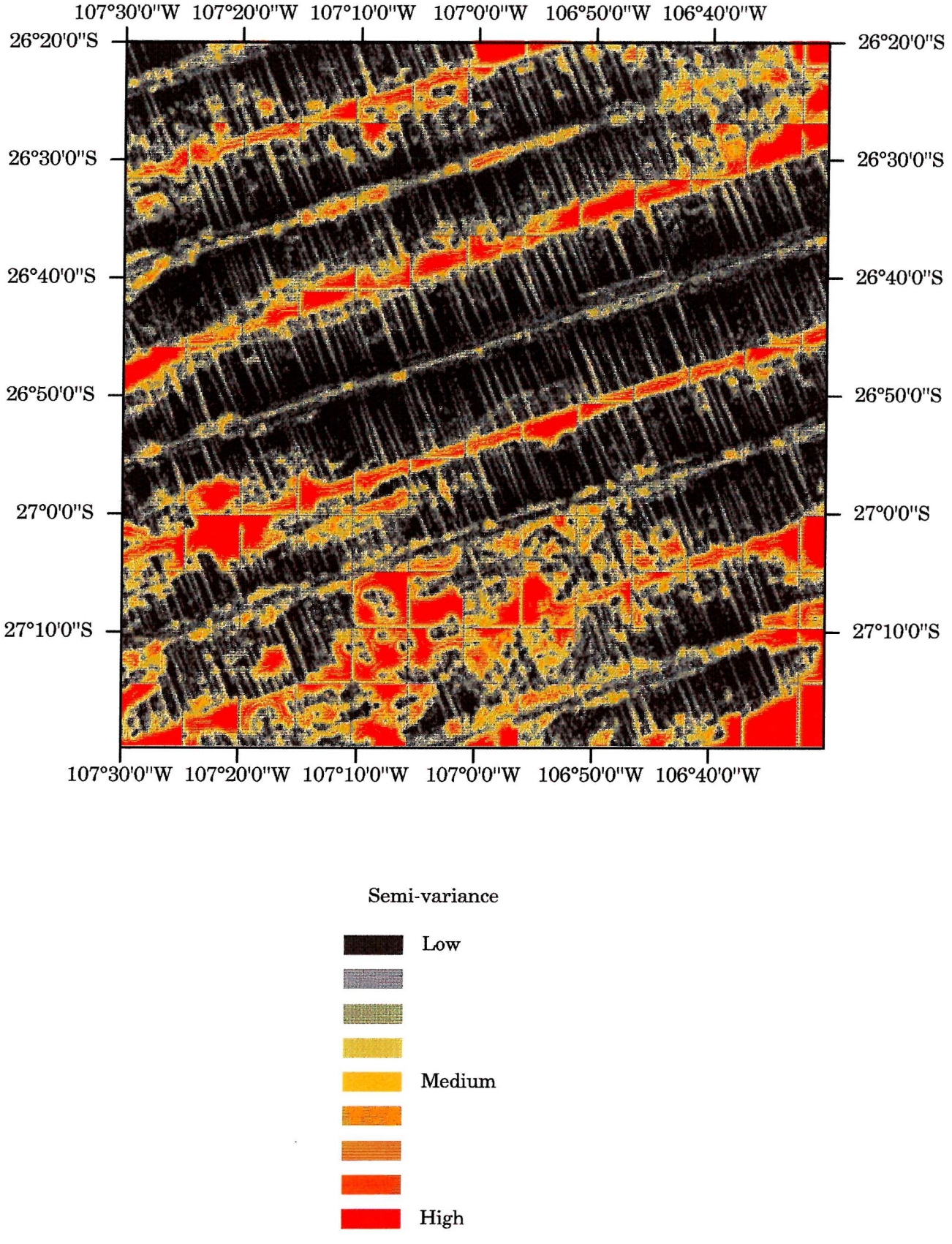
The secondary output from the kriging process is a map of the semi-variance (Figure 23). These can be graded into low, medium and high semi-variance and a threshold chosen to remove poor kriged values on the interpolated bathymetry map. This is subjective by the interpreter. In this case a value of 200^2 (most of the reds and oranges on the map) was chosen. As the semi-variance is a squared function the values are square rooted to provide a better linear scale. However as this threshold is a single value this scaling is not important. It must be stressed however that this thresholding is dependant on the interpreter and the type of geological scenario depicted by the data.

A blocky appearance may be visible in the map. This is due to edge effects of the kriging process. Blocks of 50 pixels are processed individually with an overlap of 5 pixels (equivalent of 10km boxes and 1km overlap). This does not effect the final bathymetry output map as these edge effects only appear when no data is available and thus have relatively high semi-variance which are then thresholded away.

Hysteresis thresholding could have been used to remove poor bathymetry value pixels. Instead of defining a single magnitude threshold, below which all points are discarded, two threshold values are defined. Any point (in the semi-variance image) below the lower threshold value is immediately kept. Points below the higher threshold are also retained if one of its neighbours is below the lower threshold. If the neighbours are between the high and low threshold values the condition is applied iteratively along the edge. Thus each point along a connected edge segment is retained as long as each point it contains lies below the higher threshold and has at least one point below the low threshold. All points above the higher threshold are discarded.

This is done initially by marking each pixel either as 'high', 'potential' or 'low' according to whether the pixel is above the high threshold, between thresholds or below the low threshold respectively. Each 'low' pixel is then passed through the following subroutine. A recursive subroutine function is has been created which looks at the 8 neighbours of a pixel, and if a neighbour is marked 'potential' the marking is changed to 'low'. When the subroutine changes the marking of 'potential' to 'low' the subroutine is called again (recursively). Thus a nest of subroutine calls can be built up as each 'potential' neighbour is converted and the process iterated to the next level down until there are no 'potential' neighbours remaining, at which point the process steps up the iteration levels looking for other 'potential' neighbours. Eventually all 'low' pixels will have been tested and the pixels marked 'low' are output as satisfactory, and any pixels that are left 'potential' or 'high' are output as thresholding fails. The values for this data for the thresholds would be around the medium semi-variance values.

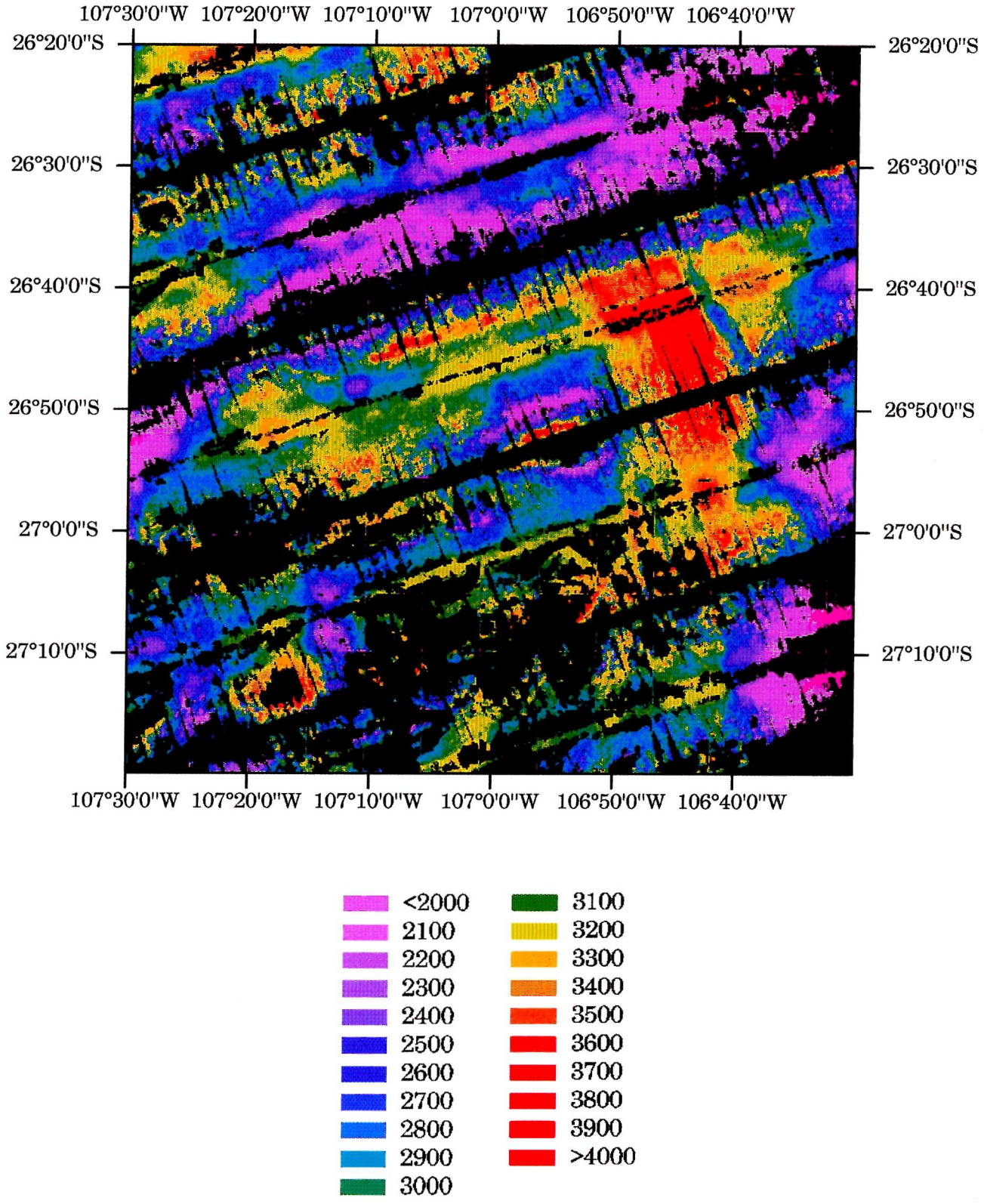
Figure 23 Colour-coded semi-variance value map of area



Thresholded kriged map

Once the semi-variance threshold has been chosen the bathymetry data can be filtered. Any locations failing the threshold test are output as black. Figure 24 now shows the bathymetry points of maximum confidence. Any large black areas can now be interpolated across using simple imagery filters.

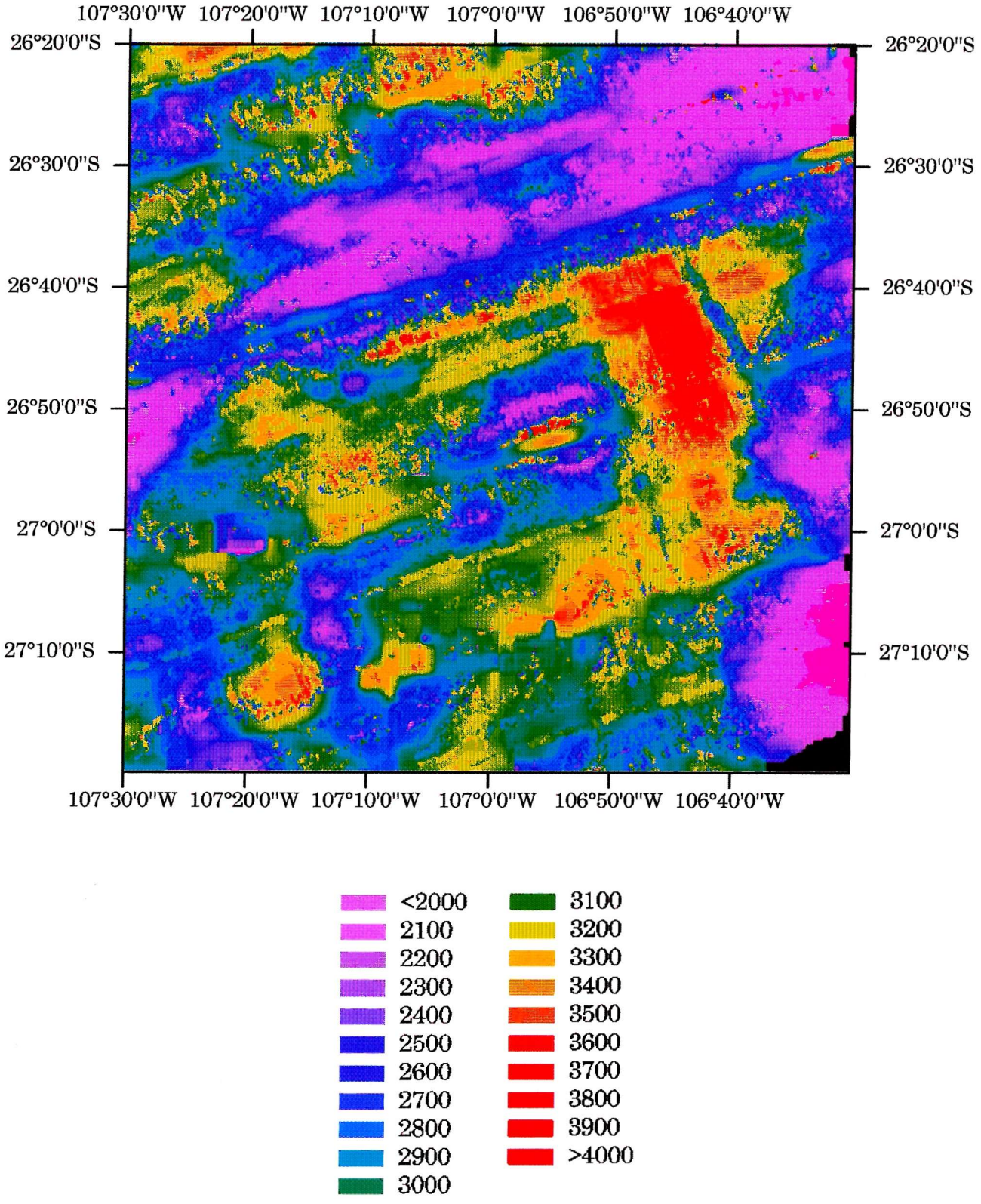
Figure 24 Semi-variance thresholded bathymetry map



Interpolated bathymetry map

The map shown in Figure 25 has been interpolated using a large box-car filter of 31 by 31 pixels which only changes pixels of zero value. This therefore does not affect the good data but merely fills areas of no data value. The resulting map shows excellent small-scale detail in the bathymetry as well as the larger scale shape of the seafloor. The geological interpretation, as was described previously, can be now be supplemented with the smaller scale features such as small ridges seen running parallel along NNW-SSE trend.

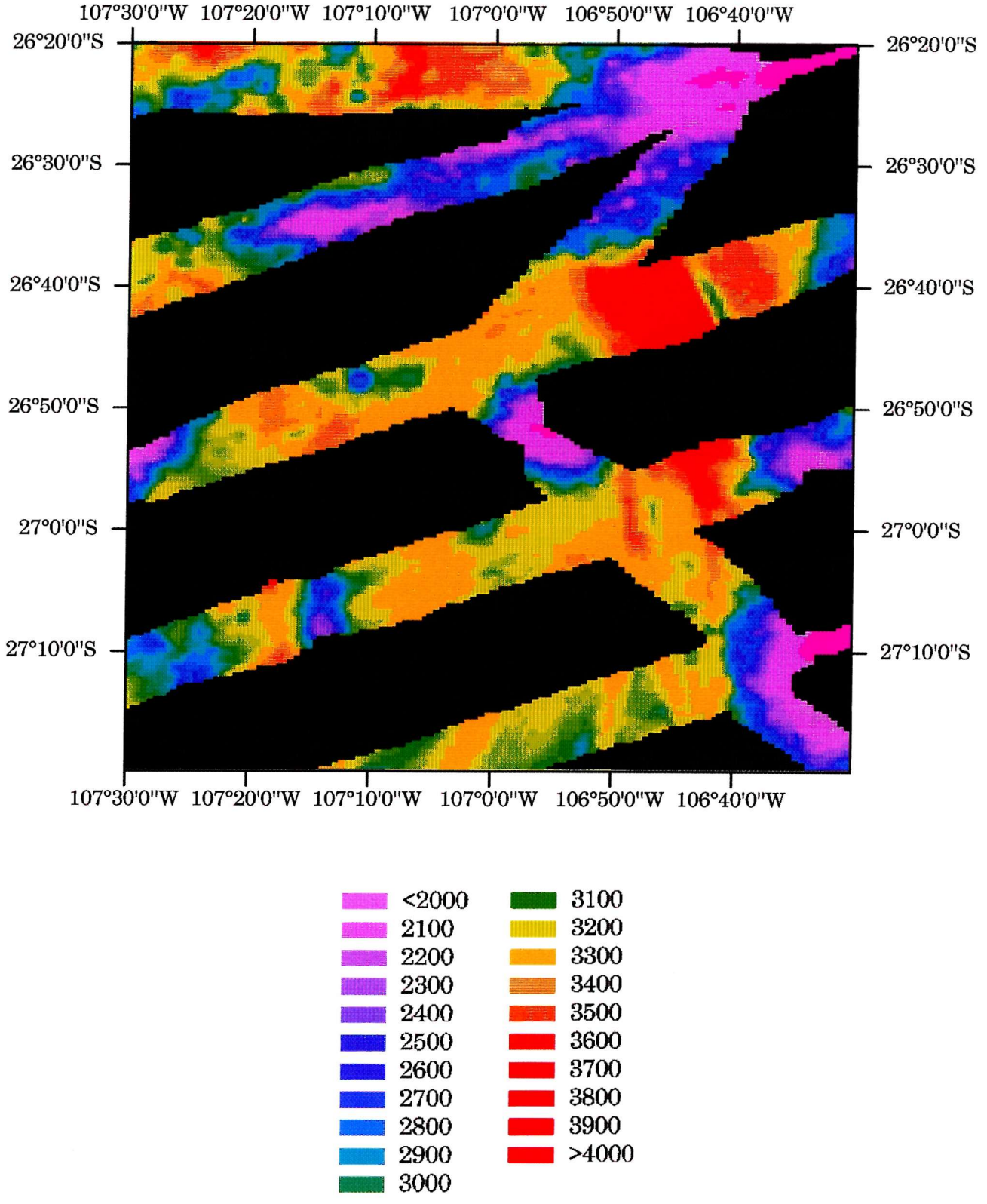
Figure 25 **Interpolated bathymetry map**



Comparison with SeaBEAM

The SeaBEAM data (Figure 26) was collected contemporaneously, apart from one zigzag track from the SE corner to the NW corner which is additional to the dataset. Comparison of the SeaBEAM data and GLORIA swath bathymetry appears to be excellent. Both datasets show the same bathymetric features where data coincide but the GLORIA data has much better coverage. The SeaBEAM data is much smoother and have no small scale features. This is presumably a function of the smoothing process used on the SeaBEAM data and on which we have no control. In conclusion the two datasets are comparable with the GLORIA advantage of greater coverage despite the water column structure producing reduced range capacity.

Figure 26 SeaBEAM bathymetry data

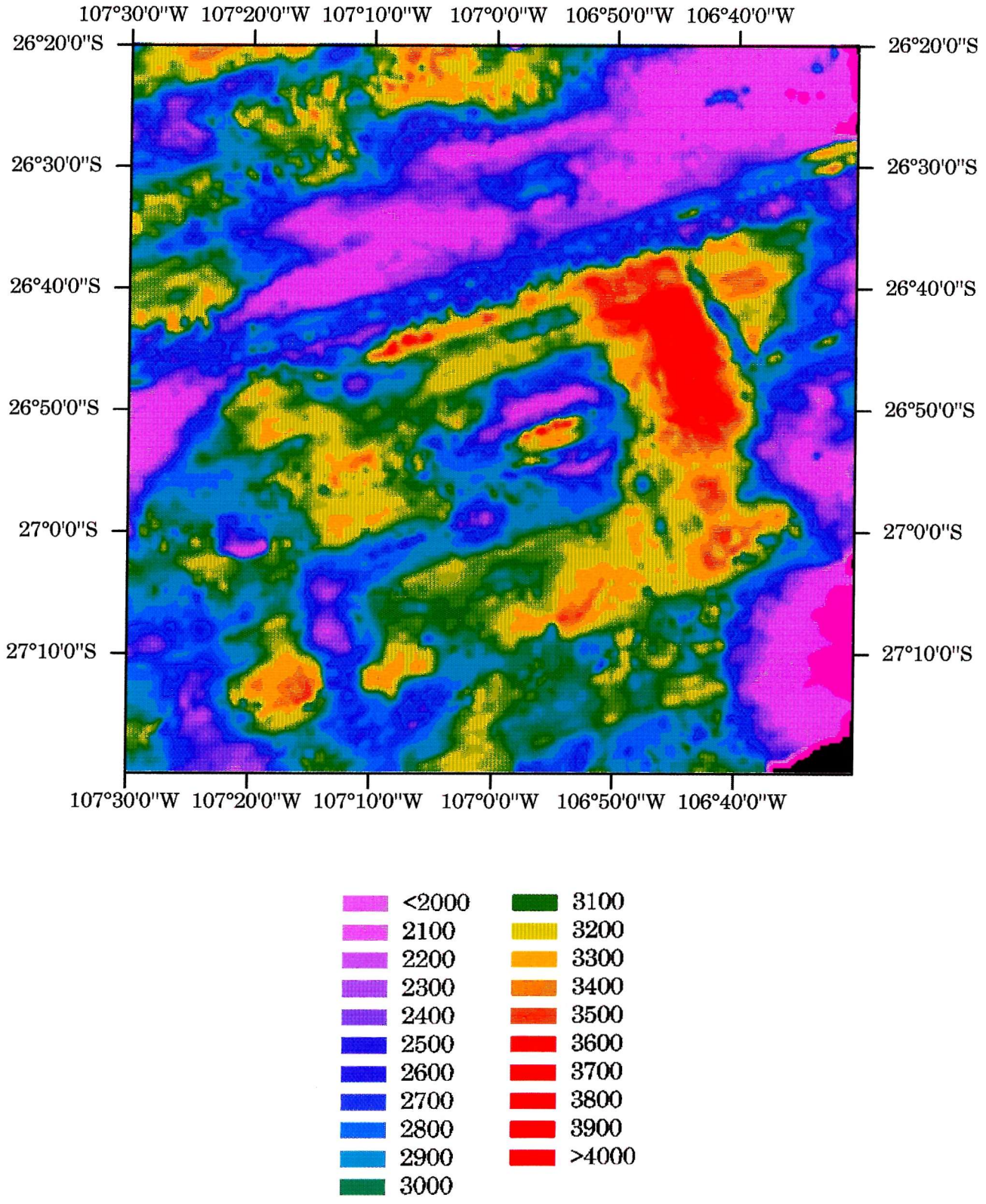


Production of final GLORIA swath bathymetry map

The map shown in Figure 27 presents the final version of the GLORIA swath bathymetry grid. A small 7 by 7 pixel kernel has been used to low pass filter the image to remove noisy pixels and to produce a minimally smoothed map. It is from this version that interpretation and integration with other datasets is possible.

The map has been created from the final grid which has been imported into ERDAS Imagine. This has simple map composing capabilities for final hardcopy. The bathymetry values vary from less than 2000 metres to over 4000 metres. Several ridges and seamounts are visible rising up to 1000 metres above the relatively flat seafloor (at between 2900 and 3200 metres). A large rhomboidal depression about 800 metres below the average seafloor is seen (centred at 26° 45'S and 106° 45'W) running NNW-SSE. Its edges are distinctly defined and thus it may be tensional rifted block being perpendicular to the local spreading direction and tension field.

Figure 27 Final GLORIA swath bathymetry map (with minimal smoothing)



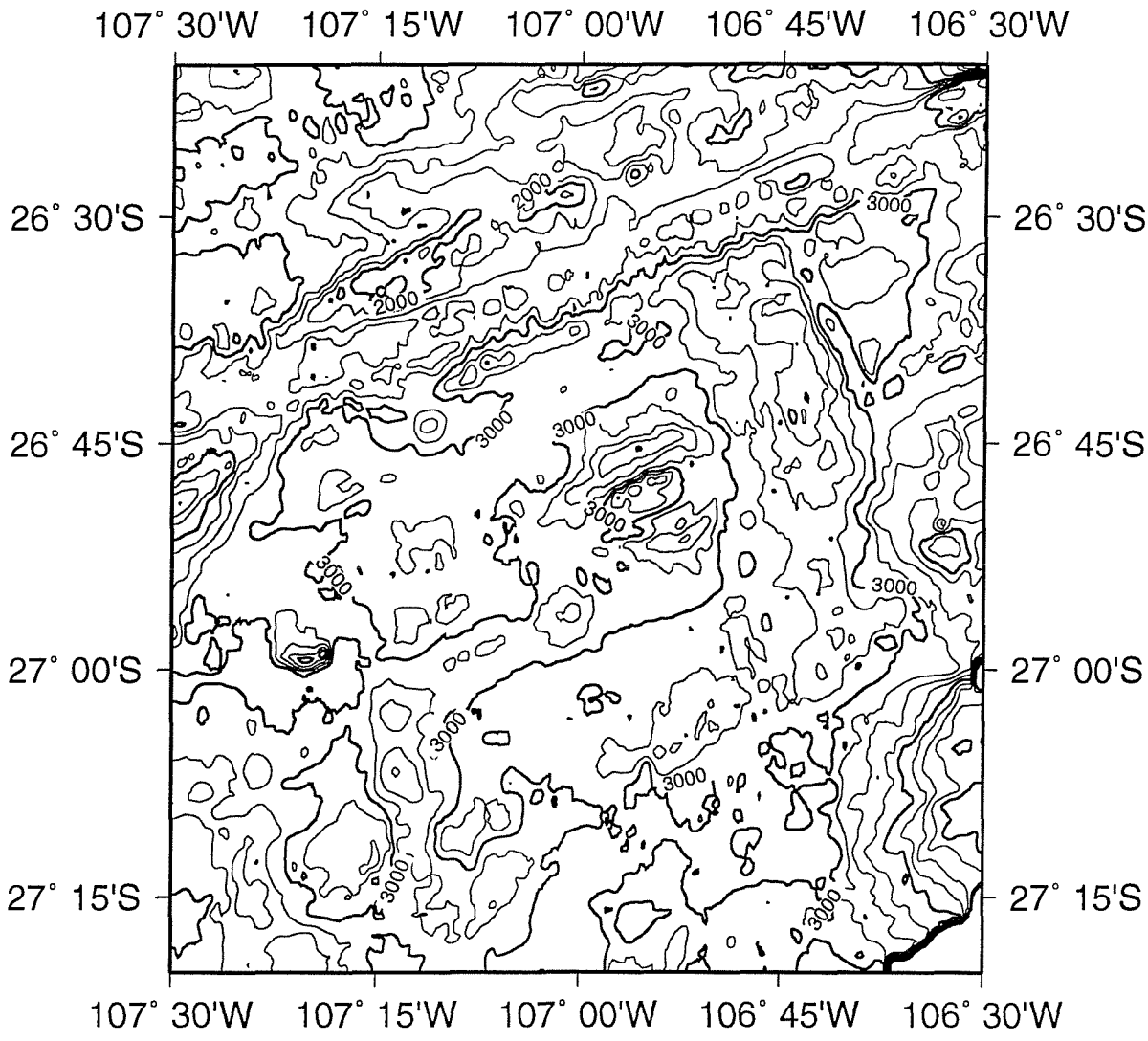
Contouring of GLORIA swath bathymetry data

There are many ways of presenting final bathymetry data of which colour coding has been favoured in this report. This is a matter of taste and can easily be replaced by contouring. Figure 28 shows a map produced with contours at intervals of 250 metres and with contour annotation every 1000 metres. This was created using the GMT commands:

```
whips2grd -i c9.cdf -o c9.grd
grdproject c9.grd -Jm1:10000 -R-107.5/-106.5/-27.3333/-26.3333 -Gc9.prj -I
grdcontour c9.prj -Jm4.374 -L1000/6000 -C250 -A1000f8 -K -X2 -Y2 > Fig28.ps
psbasemap -Jm -R-107.5/-106.5/-27.3333/-26.3333 -O -B0.25 >> Fig28.ps
```

The contours show very little bias to the ship track direction. Much of the data show contours running across-track and correlating well with adjacent swaths. This map does, however, also exhibit some data gaps between adjacent swaths, distances of up to 3 km. The contour lines show these well and thus can be ignored.. As mentioned previously 4 hours of data could not be retrieved from the archive and have thus had no low magnitude threshold applied and accounts for the jagged contour lines seen running from 26° 40'S and 107° 10'W to 26° 30'S and 106° 40'W.

Figure 28 Contour map of final GLORIA swath bathymetry map



Comparison with SeaBEAM

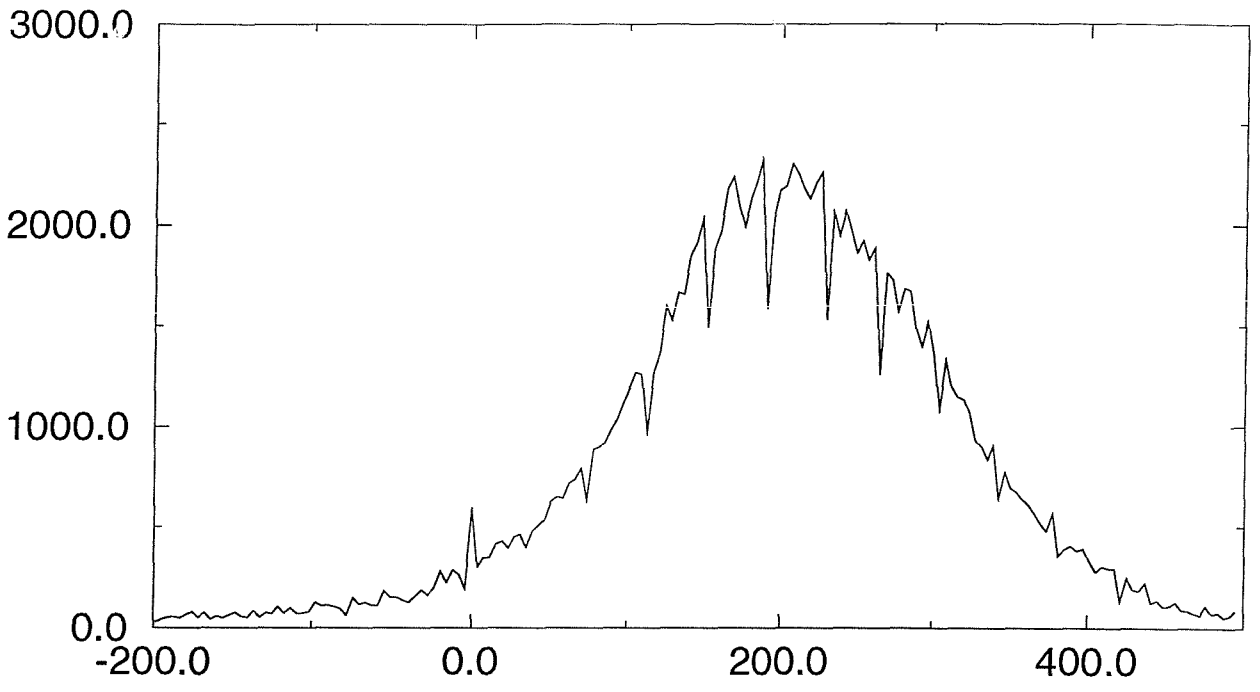
When comparing the SeaBEAM data (Figure 26) and the GLORIA data (Figure 27) the similarity is immediately visible. If the two images are overlaid and subtracted.(where SeaBEAM data exists) a residual image can be made. From this a histogram of the residuals can be extracted. This is shown in figure 29. The horizontal axis represents the SeaBEAM data minus the GLORIA data (in metres). As can be seen the centre of the histogram is about 200 thus showing that the SeaBEAM depths are on average 200 metres greater than the GLORIA depths. The histogram has many jagged points which are an artefact of the binning process (each bin represents 4 metres) and thus can be ignored in all statistical calculations. The peak at zero is due to lack of both GLORIA and SeaBEAM bathymetry data and should also be ignored but will affect the statistical calculation, fortunately only in a very minor way.

The statistical features of the comparison are:

- Median 202
- Mode 187
- Mean 191
- Standard Deviation 129

It is therefore suggested that the bathymetry data is biased 200 metres deeper. The standard deviation is about 4% of water depth.

Figure 29 Histogram of depth differences between SeaBEAM bathymetry and GLORIA swath bathymetry.

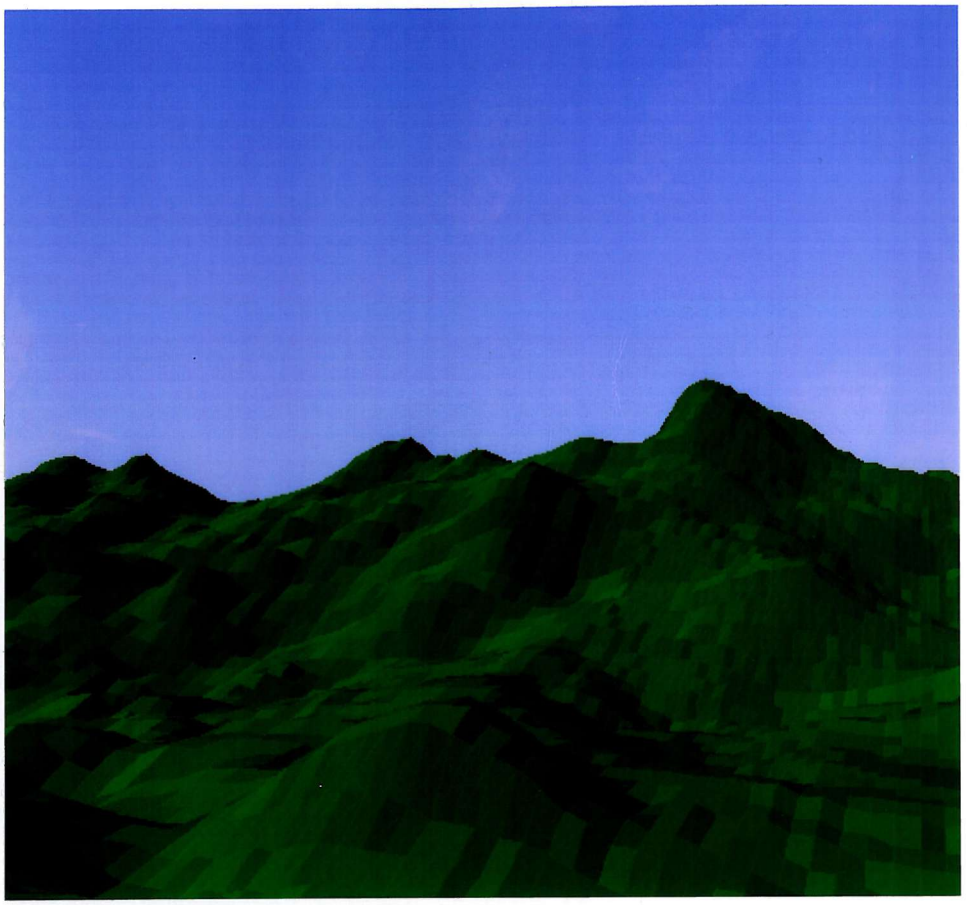


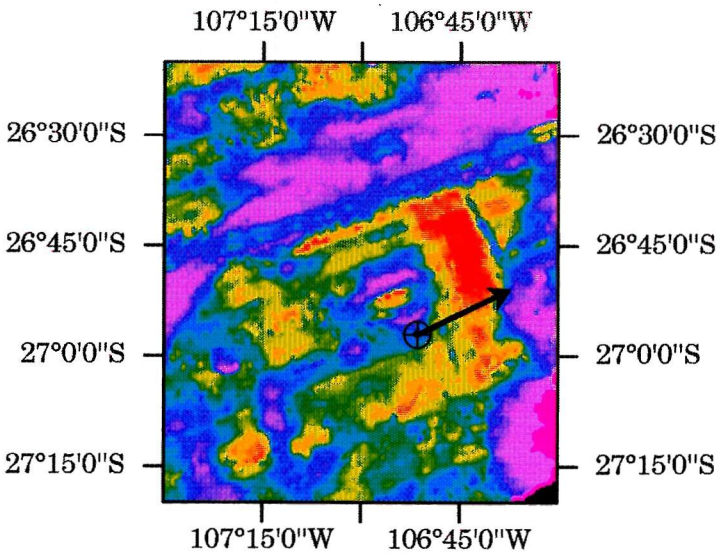
Three dimensional view of GLORIA swath bathymetry

Visualisation of bathymetric datasets is useful for the human understanding of an area. Flying around the area and having a 'feel' for the data is most useful for interpretation, in the same way as a field geologist likes to travel around his fieldwork area before detailed analysis begins. Figure 30 is a three dimensional image which shows the 'view' from a vantage point 2500 metres above seafloor at 26° 58'S and 106° 53'W and looking ENE. A vertical exaggeration of 4 has been used to emphasis the bathymetry. The small map on Figure 30 shows the location and direction for reference. Complete moving fly-through videos can be made from a series of three-dimensional images and if significant computing power is available the process can be made interactive.

Draping of secondary datasets over the three-dimensional model is also possible. These datasets include: the GLORIA imagery, texture analysis classification, or cores (if available).

Figure 30 Three dimensional view of detail within the GLORIA swath bathymetry map





References

Chavez, P. S. [1986] Processing techniques for digital sonar images from GLORIA. Photogrammetric Engineering and Remote Sensing, 52(8) p.1133-1145.

de Moustier, C. [1993] Signal Processing for Swath Bathymetry and Concurrent Seafloor Acoustic Imaging. In: Acoustic Signal Processing for Ocean Exploration. (Moura and Lourtie. eds) NATO ASI Series Kluwer (1993).

Denbigh, P. N. [1989] Swath Bathymetry: Principles of Operation and Analysis of Errors. IEEE Journal Ocean Eng. 14(4) Oct. 1989.

Grace, O. D. and Pitt, S. P.[1970] Sampling and Interpolation of bandlimited Signals by Quadrature Methods. Journal Acoustic Society America 48(6) Part 1.

Hussong, D. M. and Fryer, P.[1983] Back Arc Seamounts and the SeaMARC II Seafloor Mapping System. EOS Trans. AGU 64(45)

Masnadi-Shirazi, M. A., de Moustier, C., Cervenka, P., and Fisk, S. [1992] Differential Phase Estimation with the SeaMARC II Bathymetric Sidescan Sonar System. IEEE Journal Ocean Eng. 17(3) Jul. 1992.

Searle R. C., Le Bas, T. P., Mitchell, N. C., Somers, M. L., Parson, L. M., and Patriat P. [1990] GLORIA Image Processing : The State of the Art. Marine Geophysical Researches. 12, pp21-39.

Somers, M. L., Carson, R. M., Revie, J. A., Edge, R. H., Barrow, B. J. and Andrews, A. G. [1978] GLORIA II - An Improved Long-Range Side-Scan Sonar. In Proc. IEEE/IERE Sub-conference on Offshore Instrumentation, Oceanology International '78, Technical Session J, pp16-24, BPS Publications, London.

Appendix A

Listing of phase.c program:

```
/* phase.c

Written 21/2/93 Russell Beale & Tim Le Bas
(phase_vals.c)
Based on magdisp.c but contains full phase processing to get I, Q values
into format for display with disp.c

Modified 23/2/93
Extends phase_vals.c by correcting for linear FM

Extended 25/2/93
Calculates estimates of actual phase angles.

Adapted 28/2/93
Just produces processed data into binary file, for subsequent processing.
Code optimised.

Altered 2/3/93
Copes with 256 plus n*16 fft bins, n=0, 1, 2,.....
Sensor info skipped for speed as we don't use it

Altered 10/3/93
Added theta correction table, adjusted kd values.

Altered 9,10 /3/93
Adjusted theta correction table to eliminate residual bias

Altered 11/3/93
COmmand line arguments added to allow batch processing

Altered (again!) 17/3/93
new bias curves tried

Altered 12/12/94
remove_low routine added bad_threshold
remove_far routine added bad_threshold

Altered 12/3/95
Finally added all user points

Corrected to full form - new bias correction, full headers etc.
*/
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <math.h>

#define RAD2DEG        180.0/3.14159265
#define PI   (double)3.141593
#define TWOPI   (double)6.283185

FILE *infile;
char inname[50] = { ""};
FILE *outfile;
char outname[50];
char *outname_p;
FILE *rayfile;
char rayendname[50]= "/raypaths.tab";
FILE *pcorrfile;
char pcorrendname[50]="/pcorrn5.asc";
FILE *scorrfile;
char scorrendname[50]="/scorrn7.asc";
char out_ext[5]=".xyz";
char *getenv();
char *outname_start;

typedef unsigned short word;
typedef unsigned char byte;

enum { NO, YES } done, file_ok;
typedef enum { PORT, STBD } side;

struct rd_block {
        float PU_freq[64];
        float PL_freq[64];
        float SU_freq[64];
        float SL_freq[64];
        } fft_blk[256];
```

```
float scale_vals[384][16];
byte timedat[384][16];

float p_theta_correct[900], s_theta_correct[900];

char header[512];

typedef enum {INVALID, VALID} valid_flag;

struct processed_data{
        char header[512];
        float phase_p[223];
        float phase_s[223];
        float power_u_p[223];
        float power_l_p[223];
        float power_u_s[223];
        float power_l_s[223];
        float theta_p[223];
        float theta_s[223];
        int time[223];
        float depth_p[223];
        float depth_s[223];
        float range_p[223];
        float range_s[223];
        valid_flag flag_p[223];
        valid_flag flag_s[223];
        } data_out;


main(argc, argv)
int argc;
char *argv[];
{
register int i;
register int j;
int x, y, y1;
int numread;
int ch;
double PU_I[32][256], PL_I[32][256], SU_I[32][256], SL_I[32][256];
double PU_Q[32][256], PL_Q[32][256], SU_Q[32][256], SL_Q[32][256];
double up_magP[256], lo_magP[256];
double up_magS[256], lo_magS[256];
double alpha, beta, magnit;
int time, angle;
float raytab[901][400][2];
float unused;

float ping_count=0.0;
int tmp_p[223], tmp_s[223];
float orig_p[223], orig_s[223];
int stop_at;

double im_p, im_s, re_p, re_s;
double delta_phi_p[223], delta_phi_s[223];

register int fft_num;
int Idx, Qdx;
double tmp_val, bad_threshold;
char rayname[50],scorrname[50],pcorrname[50];
char envcorr[50];    /*   e.g. /local/tlb/mlv10693/corrfile */

double calc_theta();
double correct_theta();
void sort(), smooft(), realft(), fft_four1();

if (argc < 3) {
        fprintf(stderr, "Usage: phase filename corrfile_dir bad_threshold\n");
        exit(1);
}

strcat(inname, argv[1]);
outname_start=strtok(argv[1], ".");
strcat(outname, outname_start);
strcat(outname, out_ext);

   strcpy(rayname,argv[2]);
   strcat(rayname,rayendname);

   strcpy(pcorrname,argv[2]);
   strcat(pcorrname,pcorrendname);

   strcpy(scorrname,argv[2]);
   strcat(scorrname,scorrendname);

   bad_threshold = 0.0;
   if (argc == 4) bad_threshold = (double) atof(argv[3]);
   bad_threshold=pow(bad_threshold,4.0);
```

```
file_ok = NO;
while ( file_ok == NO) {
        if ( (infile = fopen( inname, "rb")) == NULL )
                {
                fprintf(stderr, "\nUnable to open %s\n", inname );
                fprintf(stderr, "\n\nEnter name of file ->");
                gets( inname);
                fprintf(stderr, "\n\n");
                }
        else{
                file_ok = YES;
        }
}

file_ok = NO;
while ( file_ok == NO) {
        if ( (outfile = fopen( outname, "wb")) == NULL ) {
                fprintf(stderr, "\nUnable to open %s\n", outname );
                fprintf(stderr, "\n\nEnter name of output file ->");
                gets( outname);
                fprintf(stderr, "\n\n");
                }
        else
                {
                fprintf(stderr, "Opened file %s\n", outname );
                file_ok = YES;
                }
}

/* read in raypath table */
file_ok = NO;
while ( file_ok == NO) {
        if ( (rayfile = fopen(rayname, "rb"))==NULL){
                fprintf(stderr, "\nUnable to open %s\n", rayname);
                fprintf(stderr, "\n\nEnter name of raypath file :");
                gets(rayname);
                fprintf(stderr, "\n\n");
        }
        else
                file_ok = YES;
}
fread(raytab, sizeof(raytab), 1, rayfile);

/* read in theta correction table */
file_ok = NO;
while ( file_ok == NO) {
        if ( (pcorrfile = fopen(pcorrname, "r"))==NULL){
                fprintf(stderr, "\nUnable to open %s\n", pcorrname);
                fprintf(stderr, "\n\nEnter name of port theta correction file :");
                gets(pcorrname);
                fprintf(stderr, "\n\n");
        }
        else
                file_ok = YES;
        if ( (scorrfile = fopen(scorrname, "r"))==NULL){
                fprintf(stderr, "\nUnable to open %s\n", scorrname);
                fprintf(stderr, "\n\nEnter name of starboard theta correction file :");
                gets(scorrname);
                fprintf(stderr, "\n\n");
        }
        else
                file_ok = YES;
}
for (i=1;i<900;i++){
        fscanf(pcorrfile, "%f%f", &unused, &p_theta_correct[i]);
        fscanf(scorrfile, "%f%f", &unused, &s_theta_correct[i]);
}

numread = fread( &data_out.header[0], sizeof( char), 512, infile);
fprintf(outfile, "%s", data_out.header);

while ( numread == 512 ){
        ping_count++;
        fprintf(stderr, ".");
        /* Read 4 spectra - 4x64 floats, 256 times */
        numread = byteswap_fread( &fft_blk[0].PU_freq[0], sizeof( float), 65536,
infile);
        if (numread!=65536) break;

        /* get data into correct bins to begin with */
    fft_num=0;
    while(fft_num<256){ /* we are only processing 223 bins, but need 256 for slant
correction */
        Idx = 0;
        Qdx = 1;
        for ( i=0; i< 32; i++ ) { /* 32 frequency bins, I & Q, PU & PL */
                PU_I[i][fft_num] = (double)fft_blk[fft_num].PU_freq[Idx];
                PU_Q[i][fft_num] = (double)fft_blk[fft_num].PU_freq[Qdx];
```

```
                PL_I[i][fft_num] = (double)fft_blk[fft_num].PL_freq[Idx];
                PL_Q[i][fft_num] = (double)fft_blk[fft_num].PL_freq[Qdx];

                SU_I[i][fft_num] = (double)fft_blk[fft_num].SU_freq[Idx];
                SU_Q[i][fft_num] = (double)fft_blk[fft_num].SU_freq[Qdx];
                SL_I[i][fft_num] = (double)fft_blk[fft_num].SL_freq[Idx];
                SL_Q[i][fft_num] = (double)fft_blk[fft_num].SL_freq[Qdx];

                Idx+=2;
                Qdx+=2;
        }
        fft_num++;
}

/* Now read sensor vals...*/
/* we don't use sensor data at present so simply fseek past this for speed.
Leave code here for later use if necessary */
/*
for(i=0;i<384;i++){
        numread = byteswap_fread( &scale_vals[i][0], sizeof(float), 16, infile);
        numread = fread(&timedat[i][0], sizeof(byte), 16, infile);
}
*/
/* 16 floats + 16 bytes times 384 = 5 x 6144 */
fseek(infile, 30720, SEEK_CUR);


/* now recalculate values to correct for slant (linear FM) */
for(fft_num=0;fft_num<256;fft_num++){ /* 223 + 1 + 32 = 256 */
        for(i=0;i<32;i++){
                alpha=0.5*i/32.0;
                beta = 1.0 - alpha;
                x=31-i;
                y=fft_num+i;
                y1=fft_num+i+1;
                PU_I[x][fft_num] = beta* PU_I[x][y] +
                        alpha*PU_I[x][y1];
                PL_I[x][fft_num] = beta* PL_I[x][y] +
                        alpha*PL_I[x][y1];
                PU_Q[x][fft_num] = beta* PU_Q[x][y] +
                        alpha*PU_Q[x][y1];
                PL_Q[x][fft_num] = beta* PL_Q[x][y] +
                        alpha*PL_Q[x][y1];
                SU_I[x][fft_num] = beta* SU_I[x][y] +
                        alpha*SU_I[x][y1];
                SL_I[x][fft_num] = beta* SL_I[x][y] +
                        alpha*SL_I[x][y1];
                SU_Q[x][fft_num] = beta* SU_Q[x][y] +
                        alpha*SU_Q[x][y1];
                SL_Q[x][fft_num] = beta* SL_Q[x][y] +
                        alpha*SL_Q[x][y1];
        }
}

/* now calculate phase values */
/* loop through 223 corrected time bins, and use all 32 frequency bins
for each tan_delta_phi estimate
*/

for (fft_num=0; fft_num<223; fft_num++){ /* only process 223 fft bins */
        im_p=0.0;
        im_s=0.0;
        re_p=0.0;
        re_s=0.0;
        up_magP[fft_num]=0.0;
        lo_magP[fft_num]=0.0;
        up_magS[fft_num]=0.0;
        lo_magS[fft_num]=0.0;
        for(i=0;i<32;i++){
                /* calc. upper and lower P & S mags. */
                tmp_val= PU_I[i][fft_num]*PU_I[i][fft_num]
                                + PU_Q[i][fft_num]*PU_Q[i][fft_num];
                tmp_val=sqrt(tmp_val);
                up_magP[fft_num] += tmp_val;
                tmp_val= PL_I[i][fft_num]*PL_I[i][fft_num]
                                + PL_Q[i][fft_num]*PL_Q[i][fft_num];
                tmp_val=sqrt(tmp_val);
                lo_magP[fft_num]+=tmp_val;
                tmp_val= SU_I[i][fft_num]*SU_I[i][fft_num]
                                + SU_Q[i][fft_num]*SU_Q[i][fft_num];
                tmp_val=sqrt(tmp_val);
                up_magS[fft_num] += tmp_val;
                tmp_val= SL_I[i][fft_num]*SL_I[i][fft_num]
                                + SL_Q[i][fft_num]*SL_Q[i][fft_num];
                tmp_val=sqrt(tmp_val);
                lo_magS[fft_num]+=tmp_val;
```

```
                /* now do least mean squares estimate */

                  /* PORT */
                  im_p += PU_Q[i][fft_num]*PL_I[i][fft_num] -
                      PU_I[i][fft_num]*PL_Q[i][fft_num];
                  /* weight = 1 for now so do nothing */
                  re_p += PU_I[i][fft_num]*PL_I[i][fft_num] +
                      PU_Q[i][fft_num]*PL_Q[i][fft_num];
                  /* STBD */
                  im_s += SU_Q[i][fft_num]*SL_I[i][fft_num] -
                      SU_I[i][fft_num]*SL_Q[i][fft_num];
                  re_s += SU_I[i][fft_num]*SL_I[i][fft_num] +
                      SU_Q[i][fft_num]*SL_Q[i][fft_num];
        }
        delta_phi_p[fft_num] = atan2(im_p,re_p);
        delta_phi_s[fft_num] = atan2(im_s,re_s);
        data_out.power_u_p[fft_num]=up_magP[fft_num];
        data_out.power_l_p[fft_num]=lo_magP[fft_num];
        data_out.power_u_s[fft_num]=up_magS[fft_num];
        data_out.power_l_s[fft_num]=lo_magS[fft_num];

}
adjust_vals(delta_phi_p, up_magP, lo_magP, PORT);
adjust_vals(delta_phi_s, up_magS, lo_magS, STBD);


/* TLB added to remove low magnitude data lines*/
if (bad_threshold !=0.0) remove_low(up_magP, lo_magP, bad_threshold, PORT);
if (bad_threshold !=0.0) remove_low(up_magS, lo_magS, bad_threshold, STBD);

/* TLB added to remove low magnitude data far range pixels*/
if (bad_threshold !=0.0) remove_far(up_magP, lo_magP, bad_threshold, PORT);
if (bad_threshold !=0.0) remove_far(up_magS, lo_magS, bad_threshold, STBD);

for (fft_num=0; fft_num<223; fft_num++){ /* don't go beyond bin 223 */
        data_out.phase_p[fft_num]=delta_phi_p[fft_num];
        data_out.phase_s[fft_num]=delta_phi_s[fft_num];
        data_out.theta_p[fft_num]=calc_theta(delta_phi_p[fft_num], PORT,
                &data_out.flag_p[fft_num]);
        data_out.theta_s[fft_num]=calc_theta(delta_phi_s[fft_num], STBD,
                &data_out.flag_s[fft_num]);
        time=(int) ((2.07 + fft_num*0.070176) * 20);
        data_out.time[fft_num]=time;
        if(data_out.flag_p[fft_num]!=INVALID){
                angle = (int) (data_out.theta_p[fft_num]*RAD2DEG *10);
                data_out.depth_p[fft_num]=raytab[angle][time][0];
                if(data_out.depth_p[fft_num]==0.0)
                        data_out.flag_p[fft_num]=INVALID;
                else data_out.range_p[fft_num]=raytab[angle][time][1];
        }
        if(data_out.flag_s[fft_num]!=INVALID){
                angle = (int) (data_out.theta_s[fft_num]*RAD2DEG *10);
                data_out.depth_s[fft_num]=raytab[angle][time][0];
                if(data_out.depth_s[fft_num]==0.0)
                        data_out.flag_s[fft_num]=INVALID;
                else data_out.range_s[fft_num]=raytab[angle][time][1];
        }
}

/* now to fft bin number smoothing */
/* sort data by range */
sort(data_out.range_p, tmp_p);
sort(data_out.range_s, tmp_s);
/* port */
for(i=0;i<223;i++){
        if(data_out.range_p[tmp_p[i]]>0.0)
                orig_p[i]=(float)tmp_p[i];
        else{
                stop_at=i;
                break;
        }
}
smooft(&orig_p[0], i-1, 4.0);
for(j=0;j<stop_at;j++){
        if((fabs(tmp_p[j]-orig_p[j]))>2.5)
                data_out.flag_p[tmp_p[j]]=INVALID;
}
/* starboard */
for(i=0;i<223;i++){
        if(data_out.range_s[tmp_s[i]]>0.0)
                orig_s[i]=(float)tmp_s[i];
        else{
                stop_at=i;
                break;
        }
}
smooft(&orig_s[0], i-1, 4.0);
for(j=0;j<stop_at;j++){
```

```
        if((fabs(tmp_s[j]-orig_s[j]))>2.5)
                data_out.flag_s[tmp_s[j]]=INVALID;
}
/* now print out results */
for(i=222;i>=0;i--){
        magnit =  pow((up_magP[i]*up_magP[i] + lo_magP[i]*lo_magP[i])/2.0,0.25);
        if(data_out.flag_p[i]!=INVALID)
          fprintf(outfile, "%f\t%f\t%f\t%f\n",
                -data_out.range_p[i], ping_count, data_out.depth_p[i],magnit);
                  }
for(i=0;i<223;i++){
        magnit =  pow((up_magS[i]*up_magS[i] + lo_magS[i]*lo_magS[i])/2.0,0.25);
        if(data_out.flag_s[i]!=INVALID)
          fprintf(outfile, "%f\t%f\t%f\t%f\n",
                 data_out.range_s[i], ping_count, data_out.depth_s[i],magnit);
                  }

/* read next values - should be header */
numread = fread( &data_out.header[0], sizeof( char), 512, infile);
while(data_out.header[0]!='G'||data_out.header[1]!='L'){
        /* not header - read in 16 more fft bins (-512 bytes cos we've already
        read those) */
        if(fseek(infile, (sizeof(struct rd_block))*16 - 512, SEEK_CUR)) break;
        numread = fread( &data_out.header[0], sizeof( char), 512, infile);
}
}

fclose(outfile);
fclose(infile);
printf("\n");
}


/* calc_theta - calculates theta values from delta_phi's */
double
calc_theta(d_phi, wot_side, flag)
double d_phi;
side wot_side;
valid_flag *flag;
{
        double theta;
        static double theta0=0.34906585;         /* 20 degrees in rad */
        static double thetaR=0.0034906585;  /* 0.2 degrees in rad */
        double theta_correction;
        double kd;
        if(wot_side==PORT){
                theta_correction = theta0 + thetaR;
                kd = -4.88;
        }
        else{
                theta_correction = theta0 - thetaR;
                kd = 4.6;
        }
        if(*flag==VALID){
                theta= asin(d_phi/kd) + theta_correction;
                /*if theta >90-i.e.adjusts nadir*/
                if (theta>3.14159265/2.0)
                        theta=3.14159265-theta;
        }
        if(theta<=0.0)
                *flag=INVALID;

        if(*flag==VALID)
                theta = correct_theta(theta, wot_side);
        return(theta);
}

adjust_vals(phi, up_mag, lo_mag, wot_side)
double *phi;
double *up_mag, *lo_mag;
side wot_side;
{
        register int i;
        int index;
        int sign;
        static int port_bottom_echo;
        double cutoff;
        int discard, before, after;
        int max=0;
        register int tmp;
        double val;
        double grad;
        int phase_flag = 0;
        int first_phase_change = 0;
        int phase_change=0;
        double grad_max=0;
        valid_flag *flag;
```

```
        if (wot_side==PORT){ /* phi should be -ve */
                cutoff=4.88;
                sign=1;
                flag=data_out.flag_p;
        }
        else{
                cutoff=4.6; /* phi should be positive */
                sign= (-1);
                flag=data_out.flag_s;
        }

        /* make all values valid to begin with */
        for(i=0;i<223;i++)
                flag[i]=VALID;

        /* now have to remove water column noise - detect bottom echo:
        2 approaches - find where phase changes sign, find max in
        power */
        for(tmp=0;tmp<45;tmp++){
                if(sign*phi[tmp]>0.0)
                        phase_change=tmp;
        }

        /* look for max. gradient of power, some (arbitary number) of
        bins before and after phase change.  Since phase_change is last time
        of occurance, bias towards earlier bins */
        /* no phase sign change, or very early bin # - just look for max. power */
        if (phase_change<10){
                before=10;
                after=30;
        }
        else{
                before=phase_change-10;
                after=phase_change+8;
        }
        for(i=before;i<after;i++){
                grad = up_mag[i+1]*lo_mag[i+1] - up_mag[i]*lo_mag[i];
                if(grad>grad_max){
                        grad_max=grad;
                        max=i;
                }
        }
        /* Is starboard bottom echo close to port? */
        if (wot_side==PORT)
                port_bottom_echo=max;
        else{
                if(max<port_bottom_echo-4||max>port_bottom_echo+4){
                        /* stbd very different - set = port */
                        max=port_bottom_echo;
                }
        }

        /* now unwrap from arbitrary point */
        for(tmp=60;tmp>=0;tmp--){
                if (fabs(phi[tmp] -phi[tmp+1])>3.141592649){
                        phi[tmp] -= TWOPI*sign;
                        if(fabs(phi[tmp])>cutoff)
                                flag[tmp]=INVALID;
                }
        }

        /* now discard everything before phase change or bottom echo */
        if (phase_change<10||max<phase_change)
                /* use bottom echo */
                discard = max;
        else
                discard=phase_change;
        for (i=0;i<discard;i++)
                flag[i]=INVALID;

}

double
correct_theta(theta, wot_side)
double theta;
side wot_side;
{
        int index;
        /* Theta correction table is in degrees, not radians */
        index=(int) (theta*RAD2DEG*10 +0.5);
        if (wot_side==PORT)
                theta = (double) p_theta_correct[index];
        else
                theta = (double) s_theta_correct[index];
        theta/=RAD2DEG;
        return(theta);
}
```

```
/* same format as fread but byteswaps input */
int
byteswap_fread(data, size_of_data, number, fp)
float *data;            /* pointer to data to be obtained*/
int size_of_data;       /* size of data items in bytes */
int number;             /* number of things to read */
FILE *fp;               /* file pointer */

{
        register int count = 0;
        float *data_start;
        union u{
                float f;
                unsigned char c[4];
        } in;
        unsigned char cin[4];
        register int i;
        number*=size_of_data/sizeof(float);
        data_start=data;
        for(i=0;i<number;i++){
                if(fread(cin, sizeof(char), 4, fp)!=4)
                        return(count);
                in.c[0]=cin[3];
                in.c[1]=cin[2];
                in.c[2]=cin[1];
                in.c[3]=cin[0];
                *(data++)=in.f;
                count++;
        }
        data=data_start;
        return(count);
}

/* sort: takes pointer to range array and sorts into order, returns order in
index array - e.g. index[3]=200 means data point 3 is 200th in the list
*/
void
sort(data, index)
float *data;
int *index;
{
int i, j;
float f;
for (i=0; i<223;i++)
        index[i]=i;
for(i=222;i>=0;i--){
        for (j=0;j<i;j++){
                if(data[index[j]]<data[index[i]]){
                        f=index[i];
                        index[i]=index[j];
                        index[j]=f;
                }
        }
}
}


/*************************************************************************
                void smooft( float *yptr, int n, float pts)
*************************************************************************
From Numerical Recipes p514.  Smooths an array y[] of n data points, with a window
whose full width is of order pts neighbouring points.
y[] is used as a temporary working array to avoid confusion over the
intial index of 1 rather than 0 used by the NR routines.
Data is copied into this temp array at the start and the filtered results
are returned to the original location at the end.
*/


void smooft(dptr, n, pts)
float *dptr;
int n;
float pts;
{

float y[4096];    /* temp work array */
int i, nmin, m=2, mo2, k, j;
float yn, y1, rn1, fac, cnst;
float *tptr;


tptr = dptr;
for ( i=1; i<=n; i++ )
        y[i] = *tptr++;

nmin = n + (int)(2.0*pts + 0.5);
while ( m < nmin )
```

```
        m *= 2;
cnst = pts/m;
cnst = cnst*cnst;
y1 = y[1];
yn = y[n];
rn1 = (float)1.0/(n-1);
for ( j = 1; j<= n; j++)
        y[j] += (-rn1*(y1*(n-j)+yn*(j-1)));
for ( j = n+1; j<= m; j++)
        y[j] = (float)0.0;
mo2 = m >> 1;
realft( &y[0], mo2, 1);
y[1] /=  mo2;
fac = (float)1.0;
for ( j = 1; j<mo2; j++)
        {
        k = 2*j + 1;
        if (fac)
                {
                if ( (fac = ((float)1.0 - cnst*j*j)/mo2) < (float)0.0)
                        fac = (float)0.0;
                y[k] = fac*y[k];
                y[k+1] = fac*y[k+1];
                }
        else
                y[k+1] = y[k] = (float)0.0;
        }

if ( (fac = ((float)1.0 - 0.25*pts*pts)/mo2) < (float)0.0)
        fac = (float)0.0;
y[2] *=  fac;
realft( &y[0], mo2, -1);
for ( j = 1; j<= n; j++)
        y[j] +=  rn1*(y1*(n-j) + yn*(j-1));

tptr = dptr;
for ( i=1; i<=n; i++ )
        *tptr++ = y[i];
return;
}
/*************************************************************************
                void realft( float *dptr, int n, int isign)
*************************************************************************
From Numerical Recipes p417. Calculates the Fourier transform of a set of 2n
real-valued data points. Replaces the data pointed to by dptr by the positive
frequency half of its complex Fourier transform.
*/


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


theta = 3.141592653589793/(double) n;
if ( isign == 1)
        {
        c2 = (float)-0.5;
        fft_four1( dptr, n, 1);
        }
else
        {
        c2 = (float)0.5;
        theta = -theta;
        }
wtemp = sin(0.5*theta);
wpr = -2.0*wtemp*wtemp;
wpi = sin(theta);
wr = 1.0 + wpr;
wi = wpi;
n2p3 = 2*n + 3;
for ( i=2 ; i<= n/2; i++ )
        {
        i4 = 1+(i3 = n2p3-(i2 = 1+(i1 = i+i-1)));
        h1r = c1*(*(dptr + i1) + *(dptr + i3));
        h1i = c1*(*(dptr + i2) - *(dptr + i4));
        h2r = -c2*(*(dptr + i2) + *(dptr + i4));
        h2i = c2*(*(dptr + i1) - *(dptr + i3));
        *(dptr + i1) = h1r + wr*h2r - wi*h2i;
        *(dptr + i2) = h1i + wr*h2i + wi*h2r;
        *(dptr + i3) = h1r - wr*h2r + wi*h2i;
        *(dptr + i4) = -h1i + wr*h2i + wi*h2r;
        wr = (wtemp = wr)*wpr - wi*wpi + wr;
```

```
        wi = wi*wpr + wtemp*wpi + wi;
        }
if (isign == 1)
        {
        *(dptr + 1) = (h1r = *(dptr + 1)) + *(dptr + 2);
        *(dptr + 2) = h1r - *(dptr + 2);
        }
else
        {
        *(dptr + 1) = c1*((h1r = *(dptr + 1)) + *(dptr + 2));
        *(dptr + 2) = c1*(h1r - *(dptr + 2));
        fft_four1( dptr, n, -1);
        }

return;
}

/*************************************************************************
                void fft_four1( float *dptr, int nn, int isign)
*************************************************************************
From Numerical Recipes. Replaces the data pointed to by dptr by its
discrete Fourier transform, if isign is 1, or replaces the data with
nn times its inverse discrete Fourier transform if isign is -1.
nn is the number of complex data points. Thus the data array must have
space for 2*nn floats.
*/

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

void fft_four1(dptr, nn, isign)
float *dptr;
int nn;
int isign;
{
int n, mmax, m, j, istep, i;
double wtemp, wr, wpr, wpi, wi, theta;
float tempr, tempi;

n = nn << 1;
j = 1;
for ( i=1; i<n; i+=2 )
        {
        if (j > i)
                {
                SWAP( *(dptr + j), *(dptr + i));
                SWAP( *(dptr + j+1), *(dptr + i+1));
                }
        m = n >> 1;
        while ( m >= 2 && j > m )
                {
                j -= m;
                m >>= 1;
                }
        j += m;
        }
mmax = 2;
while ( n > mmax)
        {
        istep = 2*mmax;
        theta = 6.28318530717959/(isign*mmax);
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for ( m=1; m<mmax; m+=2 )
                {
                for ( i=m; i<=n; i+=istep )
                        {
                        j = i + mmax;
                        tempr = (float)(wr*(*(dptr + j)) - wi*(*(dptr + j+1)));
                        tempi = (float)(wr*(*(dptr + j+1)) + wi*(*(dptr + j)));
                        *(dptr + j) = *(dptr + i) - tempr;
                        *(dptr + j+1) = *(dptr + i+1) - tempi;
                        *(dptr + i) += tempr;
                        *(dptr + i+1) += tempi;
                        }
                wr = (wtemp=wr)*wpr - wi*wpi + wr;
                wi = wi*wpr + wtemp*wpi + wi;
                }
        mmax = istep;
        }

return;
}


/* low magnitude removal of complete swaths */
```

```
remove_low(up_mag, lo_mag, bad_threshold, wot_side)
double *up_mag, *lo_mag;
double bad_threshold;
side wot_side;
{
        int i,j,chunk;
        double sum;
        double grid[224];
        valid_flag *flag;
        enum { TRUE, FALSE } tx_on;

        if (wot_side==PORT){
                flag=data_out.flag_p;
        }
        else{
                flag=data_out.flag_s;
        }

        for(i=0;i<223;i++){
                grid[i] =  (up_mag[i]*up_mag[i] + lo_mag[i]*lo_mag[i])/2.0;
                }

        tx_on = FALSE;
        chunk = 0;
        j=50;
        while(tx_on == FALSE && chunk < 10) {
                sum = 0.0;
                for (i=0;i<10;i++)
                        sum += grid[j++];

                if ( (sum/10.0) > bad_threshold )
                        tx_on = TRUE;
                chunk++;
                }

        if (tx_on == FALSE) {
                for(i=0;i<223;i++)
                        flag[i] =  INVALID;
                }

}
/* low magnitude removal of individual pixels */

remove_far(up_mag, lo_mag, bad_threshold, wot_side)
double *up_mag, *lo_mag;
double bad_threshold;
side wot_side;
{
        int i;
        valid_flag *flag;
        enum { TRUE, FALSE } bot_on;

        if (wot_side==PORT){
                flag=data_out.flag_p;
        }
        else{
                flag=data_out.flag_s;
        }


                for (i=223;i>0;i--) {
                        if ( ((up_mag[i]*up_mag[i] + lo_mag[i]*lo_mag[i])/2.0)
                                < bad_threshold) {
                                        flag[i] =  INVALID;
                                          }
                                }

}

#undef SWAP
```

Appendix B

Listing of krigswath2.f program:

```
      parameter (iblock=50)
      parameter (iovrlap=6)
      parameter (iblock2=iblock*iblock)
      real xarr(iblock2),yarr(iblock2),value(iblock2)
      real zest(iblock,iblock),zstd(iblock,iblock)
      real radius,spacing,zr,x,y
      integer np,xc,yc,i,j,k,nl,ns,thresh
      integer*2 z(600,600)
      integer*2 z1(600,600)
      character*15 inname,outname,outname2,argv

      external gam

      i = iargc()
      argv ="1"
      if (i.ge.1) call getarg(1,argv)
      write(inname,  '(''map'',a1,''.xyz'')') argv
      write(outname, '(''map'',a1,''.krg'')') argv
      write(outname2,'(''map'',a1,''.std'')') argv
      thresh=15000
      if (i.eq.2) then
            call getarg(2,argv)
            read(argv,*) thresh
            endif
      write(6,*) i,thresh

      call groute('s ldummy;e')
      call gopen
      call kopen

      open(1,name=inname,status='old')
      open(2,name=outname,status='unknown')
      open(3,name=outname2,status='unknown')

      read(1,*) spacing,ns,nl
      do 5 j=nl,1,-1
      do 6 i=1,ns
      read(1,*) y,x,zr
      if (zr.gt.20000.0) zr=0.0
      z(i,j) =int(zr)
      z1(i,j)=0
 6    continue
 5    continue
      rewind(1)
      read(1,*) spacing,ns,nl

      do 30 yc=1,nl,iblock-iovrlap
      do 33 xc=1,ns,iblock-iovrlap

      k=1
      do 10 j= 0,iblock-1
      do 13 i= 0,iblock-1
            xarr(k)=real(i+1)
            yarr(k)=real(j+1)
            value(k)=real(z(i+xc,j+yc))
            if (value(k).gt.0.1) k=k+1
 13   continue
 10   continue
      np=k

c     write(6,*) np
      radius=10.0

      call kneq(5)
      call krig2(xarr,yarr,value,np,radius,iblock,iblock,zest,zstd,gam)

      do 20 j= iovrlap,iblock-1-1
      do 23 i= iovrlap,iblock-1-1
      z(i+xc,j+yc)=int(zest(i+1,j+1))
      z1(i+xc,j+yc)=int(zstd(i+1,j+1))
 23   continue
 20   continue

      do 25 j= 0,iblock-1-1
      do 27 i= 0,iblock-1-1
      if (z1(i+xc,j+yc).eq.0)  z(i+xc,j+yc)=int(zest(i+1,j+1))
      if (z1(i+xc,j+yc).eq.0) z1(i+xc,j+yc)=int(zstd(i+1,j+1))
 27   continue
 25   continue
```

```
 33     continue
 30     continue

        write(2,40) spacing,ns,nl
        write(3,40) spacing,ns,nl
 40     format(f7.4,2i10)

        do 50 j=nl,1,-1
        do 55 i=1,ns
        read(1,*) y,x,zr
        if (z1(i,j).gt.thresh) z(i,j)=0
        write(2,42) y,x,real(z(i,j))
        write(3,42) y,x,real(z1(i,j))
 42     format(2f10.4,f7.1)
 55     continue
 50     continue

        close(1)
        close(2)
        close(3)

        call gclose

        end

        real function gam(h)
        real h
        a=40.0
        b=0.0
        c1=120000.0
        c=30000.0
        if(h.ge.a) goto 1
        if(h.le.b) goto 2
        gam=((h-b)*c1+(a-h)*c)/(a-b)
        return
 1      gam=c1
        return
 2      gam=c
        return
        end
```

KRIG2 UNIRAS REFERENCE

NAME KRIG2 - Gridding in 2D by kriging

SYNOPSIS (FORTRAN)

SUBROUTINE KRIG2 (XARR, YARR, VALUE, NP, RADIUS, NX, NY, ZEST, ZSTD, GAM)

```
INTEGER  NP,NX,NY
REAL  RADIUS, XARR(NP), YARR(NP), VALUE(NP), ZEST(NX,NY), ZSTD(NX,NY)
EXTERNAL  GAM
```

ARGUMENTS

XARR	In	X-coordinates of samples
YARR	In	Y-coordinates of samples
VALUE	In	Sample values
NP	In	Number of samples
RADIUS	In	Search radius
NX	In	Number of grid nodes in X direction
NY	In	Number of grid nodes in Y direction
ZEST	Out	Estimated values at grid nodes
ZSTD	Out	Standard deviations at grid nodes
GAM	In	Semi-variogram function

DESCRIPTION

Estimates the value of a bivariate random function at all grid nodes belonging to a 2D regular grid by the method of kriging.

NP samples of the random function have been taken and (XARR(I), YARR(I)) is the position and VALUE(I) is the value of the I'th sample. The random function estimate and the standard deviation of the estimate are calculated at NX x NY grid nodes with indices (I, J), where I varies between 1 and NX and J between 1 and NY. ZEST(I,J) is the estimate of the function at the node with indices (I, J) and ZSTD(I,J) is the corresponding standard deviation. The position of the grid node with indices (I, J) is

(XMIN + (I - 1) x DX, YMIN + (J - 1) x DY)

where DX = (XMAX - XMIN)/(NX - 1)
and DY = (YMAX - YMIN)/(NY - 1)

Here XMIN, XMAX, YMIN, and YMAX are the world coordinatelimits.

GAM is a theoretical semi-variogram in the form of a real function which must be declared as follows:

```
REAL FUNCTION GAM(H)
REAL H
```

GAM must be supplied by the user and linked with the user's program.

For a specific grid node, only samples within a circle centered on the node and of a radius equal to the value of the parameter RADIUS, contribute to the estimate at that node. If no samples are fouind inside this circle the estimate at the node is set to the "undefined value." There is an upper limit M on the number of samples, which can contribute to an estimate. If more than M samples are found within the circle, excess samples are ignored.

M can be set calling KNEQ. The default value of M is 49, which is also the largest possible value. The world coordinate limits are set by calling GLIMIT. When GLIMIT has not been called, the coordinate limits are calculated from the input data according to:

XMIN = min(XARR(I), I =.1,, NP)
XMAX = max(XARR(I), I =.1,, NP)
YMIN = min(YARR(I), I =.1,, NP), etc.

FUNCTIONAL INTERACTIONS

This routine is affected by the following routine:
KNEQ Set number of kriging equations

NOTES

1. The arrays XARR, YARR, VALUE, ZEST, ZSTD must be declared:
   DIMENSION XARR(NP), YARR(NP), VALUE (NP), ZEST(NX,NY), ZSTD(NX,NY)

Appendix C

Listing of process_swath script:

```
#/bin/sh
#
#   process_swath:
#
#
#       This is a command file for automatically processing GLORIA swath bathymetry
data.
#       map_no      - is the map number
#       option      - values of 1 - 4
#       cruise      - is the name of the cruise eg mlv10593
#       images      - is the dircetory of the raw data files (35Mb)
#
#   VERSION: 1.0
#
#   DATE: 18 January 1995
#
#   WRITTEN BY: Tim Le Bas
#               I.O.S.D.L.
#               Brook Road
#               Wormley
#               SURREY. UK.
#               GU8 5UB.
#
#   MODIFIED:  None.
#
# ##############################################################################

# check if we have arguments
if ($#argv < 1  || $#argv > 7) then
  echo ""
  echo "There are upto 7 possible arguments to this command:-"
  echo ""
  echo " 1)  Map Number                                          (no default) "
  echo " 2)  Processing option                                   (default = 1)"
  echo "          Option 1: Raw (.dat) to map(map_no).grd & .cdf"
  echo "          Option 2: Raw (.dat) to .llz"
  echo "          Option 3: .llz to map(map_no).grd & .cdf"
  echo "          Option 4: map(map_no).grd to map(map_no).ps"
  echo " 3)  The Full directory name of the raw data files  (default is current dir"
  echo "                                                       - opts 1 2 3)"
  echo " 4)  The name of the cruise                          (no default - opts 1 2)"
  echo " 5)  The bad-data threshold x=(y-20.6)/0.354         (default = 167.8 "
  echo "                                                       - opts 1 2)"
  echo " 6)  Bathymetry semi-variance threshold              (default = 200 "
  echo "                                                       - opts 1 3)"
  echo " 7)  The spacing of grid nodes in degrees            (default = 0.0018018 "
  echo "                                                       - opts 1 3 4)"
  echo ""
  echo "   e.g.  process_swath 3 1 /local2/tlb/mlv10693/raw mlv10693 200.0 250
0.0009009"
  echo ""
  echo "  Please note the following files are required by this program:"
  echo "     map(map_no).dat"
  echo "     cruise_name.nav  (in options 1 & 2 only)"
  echo "     raypaths.tab     (in options 1 & 2 only)"
  echo "     pcorrn5.asc      (in options 1 & 2 only)"
  echo "     scorrn7.asc      (in options 1 & 2 only)"
  echo ""
  exit
endif

# Setup the variables using the args.
set map_no = `expr $1`

# Check if default selected
if ($#argv >= 2) then
  set option = `expr $2`
else
  set option = "1"
endif

# Check if default selected
if ($#argv >= 3) then
  set images = `expr $3`
else
  set images = $cwd
endif

# Check if default selected
```

```
if ($#argv >= 4) then
  set cruise = `expr $4`
else
  set cruise = "not-specified"
endif

# Check if default selected
if ($#argv >= 5) then
  set badthresh = `expr $5`
else
  set badthresh = "167.8"
endif

# Check if default selected
if ($#argv >= 6) then
  set var_thr = `expr $6`
else
  set var_thr = "200"
endif

# spacing of 200 metres = 0.0018018 degrees
# Check if default selected
if ($#argv == 7) then
  set spacing = `expr $7`
else
  set spacing = "0.0018018"
endif

# check file map(map_no).dat exists
set file = "map"$map_no".dat"
if (! -f $file) then
  echo " Cannot proceed.....File $file not found "
  exit
endif

# Set up the name of the program run log file
set log_file = ""$cruise"map"$map_no".log"

# Write out to the screen user-variables
echo " Map Number         "$map_no""
echo " Option             "$option
echo " Cruise name        "$cruise
echo " Images             "$images
echo " Spacing            "$spacing
echo ""

# ############################################################################

if ($option == "1" || $option == "2") then

# ############################################################################

echo ""
echo "Please enter the path name of where navigation files are kept:"
echo "   eg"
echo "       /data/nav      ../nav"
echo ""
echo "Current directory is:"
pwd
echo ""
echo "(Press RETURN to select current directory)"
set input = $<

# Check if current directory selected
if ($input == "") then
 set nav = "."
else
  set nav = ""$input""
endif

# check file cruise.nav exists
set file = ""$nav"/"$cruise".nav"
if (! -f $file) then
  echo " Cannot proceed.....File $file not found "
  exit
endif

echo ""
echo "Please enter the path name of where correction files are kept:"
echo "   eg"
echo "       /data/corrfiles       ../corrfiles"
echo ""
echo "Current directory is:"
pwd
echo ""
echo "(Press RETURN to select current directory)"
set input = $<
```

```
# Check if current directory selected
if ($input == "") then
 set corrfile = "."
else
  set corrfile = "$input"
endif

# check file pcorrn5.asc exists
set file = "$corrfile"/pcorrn5.asc"
if (! -f $file) then
  echo " Cannot proceed.....File $file not found "
  exit
endif

# check file scorrn7.asc exists
set file = "$corrfile"/scorrn7.asc"
if (! -f $file) then
  echo " Cannot proceed.....File $file not found "
  exit
endif

# check file raypaths.tab exists
set file = "$corrfile"/raypaths.tab"
if (! -f $file) then
  echo " Cannot proceed.....File $file not found "
  exit
endif

echo ""
echo " Correction files  "$corrfile
echo ""

# check directory ../xyz exists
set file = $images"/../xyz"
if (! -d $file) then
  mkdir $file
endif

# check directory ../llz exists
set file = $images"/../llz"
if (! -d $file) then
  mkdir $file
endif

# ##############################################################################

# Phase process the 35Mb files

set com = "/bin/ls -r "$images""
$com | grep .dat > temp.in
set com = "do_phaseall "$images" "$corrfile" "$badthresh
$com < temp.in
echo $com "< temp.in"
rm temp.in

# ##############################################################################

# Merge the navigation into the xyz files

set com = "/bin/ls -r "$images""
$com | grep .xyz > temp.in
set com = "do_mergeall "$images" "$nav" "$cruise""
$com < temp.in
echo $com "< temp.in"
rm temp.in

# ##############################################################################

# Move the phase processed xyz files to their own directory

set com = "mv "$images"/*.xyz "$images"/../xyz/."
echo " mv xyz files to separate directory"
$com

# ##############################################################################

# Move the merge processed llz files to their own directory

set com = "mv "$images"/*.llz "$images"/../llz/."
echo " mv llz files to separate directory"
$com

endif
# end of $option 1 and 2

# ##############################################################################

if ($option == "1" || $option == "3") then
```

```
# ##############################################################################

# Choose the relavent llz files

set com = "/bin/ls -r "$images"/../llz"
$com | grep .llz > tempallllz.dat
set com = "choose "$map_no" "$images"/../llz tempallllz.dat"
echo $com
$com
/bin/rm tempallllz.dat

# ##############################################################################

# Grid the relavent llz files (median grid points and output as xyz
#  (This one takes a long time!)

set com = "gridswath -M"$map_no" -I"$spacing" -imap"$map_no".cho -omap"$map_no".xyz"
echo $com
$com

# ##############################################################################

# extra bits and pieces

setup v1.2.2 motif
setup new uniras

# ##############################################################################

# Krig the map(map_no).xyz file (median grid points and output as map(map_no).krg
# and map(map_no).std (error map)
#  (This one takes a long time!)

set com = "krigswath2 "$map_no" "$var_thr
echo $com
$com

# ##############################################################################

set lim = "`cat map"$map_no".dat`"
set west = `expr $lim[3]`
set east = `expr $lim[4]`
set south = `expr $lim[1]`
set north = `expr $lim[2]`

# ##############################################################################

set com = "xyz2grd map"$map_no".krg -Gmap"$map_no".grd -I"$spacing" -:  -
R"$west"/"$east"/"$south"/"$north
echo $com
$com

# ##############################################################################

set com = "xyz2whips -i map"$map_no".krg -o map"$map_no".cdf"
echo $com
$com

# ##############################################################################

set com = filter -i map"$map_no".cdf -o map"$map_no"fil.cdf -z -b31,31"
echo $com
$com

endif
# end of $option 1 and 3

# ##############################################################################

if ($option == "4") then

# ##############################################################################

set lim = "`cat map"$map_no".dat`"
set west = `expr $lim[3]`
set east = `expr $lim[4]`
set south = `expr $lim[1]`
set north = `expr $lim[2]`

# ##############################################################################

# check file tempbathcol.dat exists
set file = "tempbathcol.dat"
if (! -f $file) then
    set com = "makecpt -S12c -M3000 -C500"
    echo $com "> tempbathcol.dat "
    $com  > tempbathcol.dat
```

```
  endif

# ##############################################################################

set out = "map"$map_no".ps"
# check if ps exists and delete if necessary
if (-f $file) then
    set com = "/bin/rm "$out
    $com
endif

# ##############################################################################
# Mercator scale 6 inches = 1 degree  == 1:729002  = (111.1*100000)/(6*2.54)
# 1:1000000 = 4.37 inches/degree = 4374015.748/1000000
# Contours limits 500/6000  Contours every 500
# Contours annotated every 1000 (font 8)

set com = "grdimage map"$map_no".grd -Jm4.374 -Ctempbathcol.dat -K -X2 -Y2"
echo $com ">" $out
$com > $out

set com = "grdcontour map"$map_no".grd -Jm -L500/6000 -O -K -C500 -A1000f8"
echo $com ">>" $out
$com >> $out


set com = "psbasemap -Jm -R"$west"/"$east"/"$south"/"$north" -O -K -B0.25"
echo $com ">>" $out
$com >> $out

set com = "psscale -Ctempbathcol.dat  -D7/2.5/5/0.5 -L -O"
echo $com ">>" $out
$com >> $out

/bin/rm tempbathcol.dat

# ##############################################################################

endif
# end of $option 4

# ##############################################################################

exit
```