Rasterizing vector and discrete data with the Woods Hole Image Processing System Software

U. S. Geological Survey Open-File Report 93-530

UNITED STATES DEPARTMENT OF THE INTERIOR
GEOLOGICAL SURVEY

Rasterizing vector and discrete data with the Woods Hole Image Processing System Software

by

Valerie Paskevich[1]

Open-File Report 93-530

August 1993

[1]Woods Hole, MA. 02543

CONTENTS

FIGURES

Abstract

The Branch of Atlantic Marine Geology has been involved in collecting, processing and digitally mosaicking high and low resolution side-scan sonar data. Recent development of a UNIX-based image-processing software system referred to as the Woods Hole Image Processing System (WHIPS), includes a series of task specific programs for processing and enhancement of side-scan sonar data. To extend the capabilities of the UNIX-based programs, development of digital mapping techniques have also been developed. To more fully understand the geologic processes and information contained within the side-scan sonar mosaics, additional data need to be registered and synthesized with the sonar mosaics. These additional data sets may be vector related, such as contour lines or map grid lines, or discrete single point sounding data such as bathymetry or magnetic values obtained along a ship's track. To accomplish this, a series of task specific programs along with various cookbook procedures have been developed and are described in this report.

Introduction

To process, map and digitally mosaic side-scan sonar data, the Branch is currently utilizing a series of task specific programs and procedures. The programs have been developed to run in a UNIX environment utilizing the UNIDATA NetCDF (Unidata, 1991) data access software and are referred to as the Woods Hole Image Processing System (WHIPS) software (Paskevich, 1992). The programs and procedures described in this report have been developed and tested on a Data General AViiON series workstation utilizing the DG/UX operating system. In addition to the AViiON, a large portion of the software has been ported and run on other systems including a Digital Equipment Corporation (DEC) DECstation 3100 and 5000, and a SUN Sparcstation.

To provide additional insight and to aid in the interpretation of the side-scan sonar mosaics, it often is necessary to create additional datasets for comparison. For example, an additional image illustrating the seafloor bathymetry for the sonar mosaic area may be desired. Once the bathymetry image is created, the user may wish to drape the side-scan sonar data over the bathymetry for a given area and produce a perspective image of the data for clearer interpretation of the sonar image.

Essentially there are two forms of rasterizing data to be used in creating an image file. The first is the rasterizing of discrete data values as in single point data values, and is accomplished using program **pointpic**. These data values are identified by a geographic coordinate with an associated value. The coordinate value may be recorded as integer or floating point and may represent any one of a variety of data types such as gravity, magnetics, bathymetry, or grain-size analysis from a laboratory. When rasterizing this type of data, the geographic coordinates are used to place the value in it's proper map and, subsequently, image space. Each data record, or point, is placed individually and the initially produced image may resemble nothing more than dot's scattered about.

The second procedure involves the rasterization of vector data, and is accomplished using program **linepic**. In this manner, the user supplies a series of geographic coordinates that comprise a line along with a value to assign to the line. When rasterizing these data, a series of points are computed to connect the supplied coordinates and thereby produces a line within the image. The user supplied value is then placed at the image coordinate. As the value is placed at adjacent coordinates between the user supplied coordinates, a line is completed and is represented by the user's value. This method may be used in a variety of ways such as rasterizing contour line information, ship's track-lines or coastline information. Once this type of image has been created, the user may wish to combine the image with an existing image as a bathymetry contour line overlay.

Detailed examples utilizing these programs and procedures are presented here. Regardless of the rasterization technique employed by the user, it may be desirable to apply additional processing or enhancement to the image. This report will present some alternative processing and enhancement techniques the user may experiment with. It is assumed that the reader has some knowledge of image processing techniques.

The procedures for rasterizing data described in this report utilize the UNIX program **proj** (Evenden, 1990) version 3.1. Depending on the type of data being rasterized (i.e. discrete or vector), the user will pass the output from program **proj** to either **pointpic** or **linepic**. Program **proj** provides the cartographic scaling and projection of the geographic coordinate data and allows the user to select from approximately 75 map projections for their final map product. The program is fully documented. The source code and documentation for **proj** may be obtained via anonymous ftp at charon.er.usgs.gov (128.128.40.24). A new version of **proj** (proj4.1) may also be obtained from charon. Though the procedures described in this report have not been tested utilizing the new version, there are no changes anticipated in the execution of the scripts described in this report.

Program **proj** operates as a standard UNIX filter utility. Input data may be piped directly into **proj**. Output of the program is to std_out and would generally be piped directly to a final program in the mapping/rasterizing procedure. The user must specify the proper **proj** parameters to define the image map to be created.

Program PROJ

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

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

Program **proj** supports approximately 75 cartographic projections. The user must properly specify any of the projection specific parameters that may be required by the selected projection. It is essential that the user have a good understanding of the available map projections and the required projection parameters to create the desired map. For more detailed cartographic characteristics of the projections, the user may refer to additional documentation (Snyder, 1987 or Snyder and Voxland, 1989).

Before actual data processing begins, both **pointpic** and **linepic** start by reading the first four pairs of coordinates passed to it from program **proj** and creates the WHIPS netCDF image. **Pointpic** and **linepic** assumes that these initial pairs of values are the map boundaries in cartesian coordinates. With the four map corners known, the program computes the size of the image area. Since the image must be defined as a rectangle, the maximum coordinates are used to compute the image size. For some map projections (i.e. conic projections), the desired map area will be smaller than the actual image size because of the arc of the longitude lines. Figure 1 shows the typical placement of a Universal Transverse Mercator (UTM) map in a defined image area and the map corners are designated as points **A**, **B**, **C** and **D**.

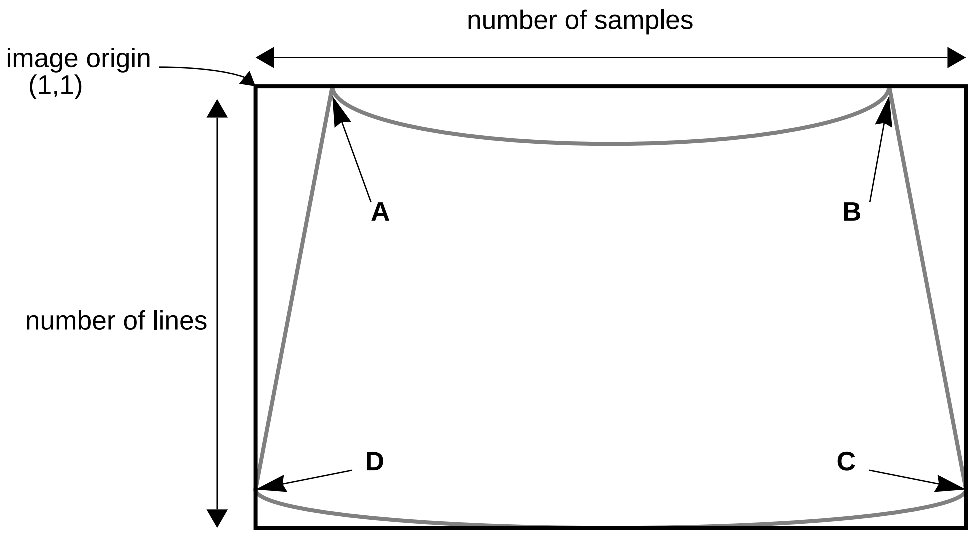

Figure 1 - UTM map placed in image space

WHIPS Programs

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

derivative	computes first order difference of an image
dk2dk	creates a new WHIPS netCDF image file by extracting a sub-area from an existing WHIPS image.
filter	applies a low-pass, high-pass, zero replacement or divide filter to an image.
grid_data	computes vector coordinates to draw map area grid lines.
linepic	takes a vector file with geographic coordinates which make up a line and rasterizes the line data in a WHIPS netCDF image file that represents a defined map space.
lowpass2b2	applies a 2-by-2 low-pass filter to a WHIPS image.
median3	applies a 3-by-3 median filter to a WHIPS image.
mode3	applies a 3-by-3 mode filter to a WHIPS image.

mode5	applies a 5-by-5 mode filter to a WHIPS image.
pointpic	take a file that contains projected geographic coordinates and an associated data value and places the data value in a WHIPS netCDF image file that represents a defined map space.
qmos	Mosaics (overlays) the specified input file over the specified output file. The program will either overlay where the input file has priority over the existing output file, where the output file has priority over the input file, or average non-zero pixel values together from the input and existing output file.
quickview	displays an 8-bit WHIPS netCDF image in an X-11 window.
raw2whips	converts a raw image data to a netCDF file for use with WHIPS
whips2raw	converts a WHIPS netCDF image file to a raw image file

Rasterizing Discrete Data

Rasterizing discrete data is perhaps the most common way to synthesize data for analysis and is, in it's general usage, a two-step procedure. In most cases various geophysical data (gravity, magnetics and bathymetry) collected as single point sources along a ship's cruise track provide the source of the data. The steps to produce this example are summarized in Appendix B, cookbook example 1.

To begin, the user must create the data file for processing. The *x* and *y* coordinates must be the geographic coordinates of the data value and must be recorded as signed decimal degrees with west longitude and south latitude recorded as negative values. The *z-value* will represent the user's data and may be recorded as an integer or floating point value.

As a practical example, assume the user wishes to process the bathymetric values recorded on a past cruise and create an image of the data for a specific map area, scale and projection. The user must begin by collecting the necessary data. Within the Branch, data has been collected and archived in various data formats over the years. Some UNIX utilities exist within the Branch for reading and re-formatting these various data formats. However, if a utility does not exist, it is the user's responsibility to properly format the data. In this example, the data file contains multiple records each with a geographic coordinate and an associated data value (*z-value*). The data fields must be separated by one or more spaces or tabs. A sub-set of the example data file, *farn3.bt,* is shown below. In this example the *z-value* represents bathymetry data.

```
...
...
26.88940   -85.00830    1348
26.89260   -85.01040    1353
26.89550   -85.01250    1358
26.89860   -85.01460    1325
26.90170   -85.01690    1292
26.90460   -85.01900    1259
```

```
26.90770   -85.02110    1244
...
...
```

Once the data has been accumulated, the second step is for the user to select the **proj** parameters to define the map area. For this example, a Universal Transverse Mercator (UTM) map will be created. The map scale will be 175 meter pixel and the map boundaries will be 24^o to 26^o north latitude and 85^o to 83.5^o west longitude. At this point the user should create a data file which contains the map corner coordinates. The map corner coordinates must be specified in the following order:

upper left
upper right
lower right
lower left

To illustrate this example, the file *wfla_bounds.dat* was created and contains the following information:

```
% more wfla_bounds.dat
26 -85
26 -83.5
24 -83.5
24 -85
```

Once the map boundaries are defined, the user specifies the **proj** parameters. As stated earlier, there are some optional **proj** parameters that must be specified for proper processing to take place. In addition to the required parameters, the user must specify any projection specific values. For this example map projection, we must additionaly specify a central longitude. For the UTM projection, the **proj** command line would be as follows:

```
proj +proj=utm +lon_0=-86 -r -s -m 1:175 -f '%.0f'
```

In addition, the input files to be processed must be included on the program run-line. The file which contains the map corners must be specified before any data files. Once the boundary data file has been specified any number of data files may be specified. This allows the user to produce a composite image of multiple cruise data. The actual **proj** run-line may then be specified as:

```
proj +proj=utm +lon_0=-86 -r -s -m 1:175 -f '%.0f' \
wfla_bounds.dat farn3.bt
```

The data contained in the input file will be processed by **proj** and output to the UNIX std_out device. As the data files are processed, the geographic coordinates are converted to meter coordinates. The data may be saved by specifying a data file or piped directly into program **pointpic**.

Directing the output of **proj** to **pointpic** is recommended. In this manner, only temporary intermediate data files are created which will help reduce the amount of disk space required for the processing. Because the output image is to contain bathymetry values, the **pointpic** option to create a 16-bit output image must also be specified. By defining the output image as 16-bit, the program is able to store pixel values that are within the range of -32768 to 32767 in the output image. The **pointpic** run-line, as it should be appended to the **proj** run-line, would look as follows:

```
|pointpic -b 16 -o wfla_bt.cdf
```

As program **pointpic** is executing, it accepts the output from program **proj** and places the data value (e.g. pixel values) in their proper image location. The program begins by reading the first four pairs of coordinates passed to it from program **proj** and creating the WHIPS netCDF image. **Pointpic** assumes that these values are the map boundaries in cartesian coordinates. With the four map corners known, the program then computes the size of the image area.

After the image size has been computed and the image has been created, the program continues by placing the data values in their proper map space. Pixel cartesian coordinates which are outside of the defined map area and yet fall within the defined image area are placed in the output image. The image is filled on a pixel-by-pixel basis as the pixels are individually processed. Pixel values are placed in the image regardless of any previous pixel that has been placed at the given coordinate. This results in coordinates with multiple values being overwritten regardless of any previous value placed at the position.

The WHIPS netCDF image created now contains the bathymetry values placed in their proper map space. A portion of the image, *wfla_bt.cdf*, created in this example is shown as Figure 2. The

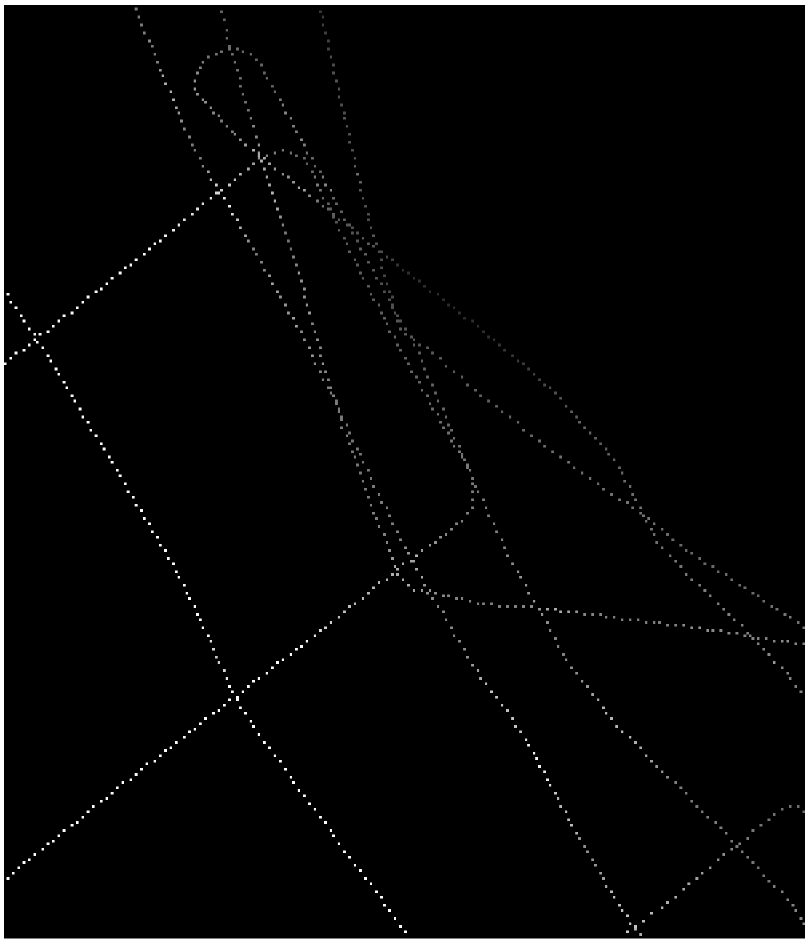

Figure 2 - Portion of *wfla_bt.cdf* with rasterized bathymetry.

bathymetry values that were placed within the image are seen as dots. As illustrated, the majority of the image is still empty since the data coverage for the desired area is not complete. The bright values contain the larger dn_values or, in other words, represents the deeper ocean while the lighter values represent the shallower ocean. At this point, the user may wish to apply additional processing to the image. One option is to apply a series of filters to fill the image. The user may choose from several available filters. Those filters include: **filter**, **mode3 mode5**, **mean3** or **lowpass2b2**.

By applying a series of filters to the image, the user may fill the entire map area with data. Depending on what filter is chosen and how it is applied by the user, the final result is an image that represents a gridded distribution of the data. Though the filters do not statistically weight the data as would be found in an actual gridding program, the user may still produce an image that represents a fairly accurate depiction of the data distribution over the map area.

The image created in this example was filtered repeatedly to fill the entire map area with data. Once the entire image area was filled, a final low-pass filter was applied to smooth the data. The filters chosen were applied arbitrarily to fill the image as quickly as possible. In selecting the filters, little concern was given to the actual data distribution and creation or placement of the new data values. The user may wish to give more thought to the filters and filter sizes applied to the image. The completed image is shown as Figure 3. A diagonal derivative computed from Figure 3

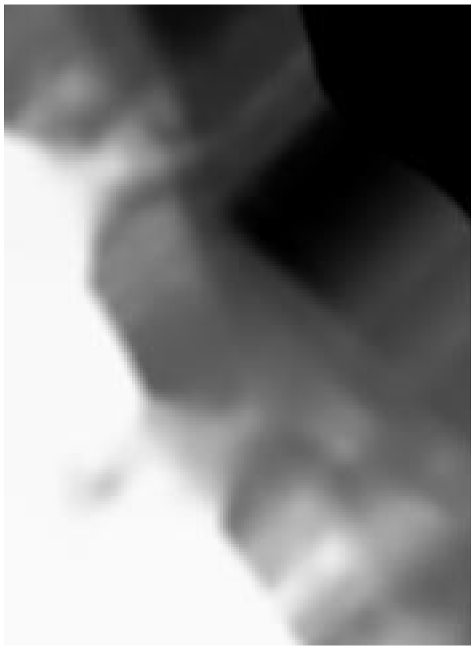
Figure 3 - Filtered image to produce complete coverage.

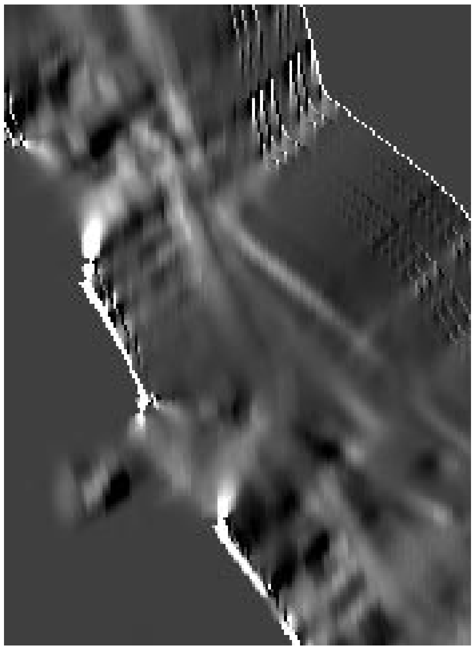
Figure 4 - Diagonal derivative of image.

is shown as Figure 4. The images displayed as Figures 2, 3 and 4 have been converted from a 16-bit data range of 600 to 3500 to a an 8-bit data range for display purposes within this report.

Rasterizing Vector Data

Procedures to rasterize vector data are similar to those for rasterizing discrete data. However, there are two major differences in rasterizing vector data as opposed to rasterizing discrete data. The first and obvious difference is the desire to draw lines to connect coordinates rather than simply placing pixel values at a specified map coordinate within the image. A second major difference in processing a vector data file, is that the user must specify the value that is to be placed at each pixel location and therefore define the dn value that the line will represent.

The user should begin by compiling the data to be rasterized. Once the data has been collected and properly formatted, the user should define the map area. When the preliminary steps have been completed, the user may continue with the execution of the processing steps and creation of the image.

The input data to be rasterized must contain a series of geographic coordinates. The coordinates may be in the DMS format acceptable by MAPGEN (Evenden and Botbol, 1985) though a simple signed decimal degree specification is recommended. Consecutive pairs of geographic coordinates are processed to produce a series of points and generate a line between the input coordinates. A file may contain several line segments. The start of each new line segment must be indicated by a record containing the MAPGEN command # -b.

As an example of rasterizing a vector data set, assume the user wishes to create an image which contains the coastline of a given area. The desired map area is 41^o to 43^o north latitude and 72^o to 69.5^o west longitude and the final map is a UTM projection. The map scale is 500 meter pixels. Utilizing the MAPGEN program **getcoast** and the available coastline files, we are able to extract the vector information that will be used to draw the coastline. An excerpt of the example data set is shown below:

```
...
...
# -b
41.461536       -70.824844
41.459483       -70.816043
41.455376       -70.813403
41.451269       -70.818097
41.449215       -70.823964
41.448922       -70.830711
41.449509       -70.838338
41.453322       -70.842152
41.458896       -70.838925
41.461243       -70.832765
41.461536       -70.824844
# -b
41.448335       -70.853593
41.442468       -70.853299
...
...
41.448335       -70.853593
# -b
41.429854       -70.924584
41.423694       -70.927518
41.420467       -70.932798
41.420173       -70.940132
41.424867       -70.942479
41.429561       -70.938666
41.431321       -70.932505
41.435134       -70.928105
41.438361       -70.922531
41.433374       -70.920771
41.429854       -70.924584
```

Once again utilizing **proj** to define the desired map area and convert the geographic coordinates to their cartesian coordinates, the vector data file is input for processing. The data file containing the map boundaries must be specified prior to the actual data file. As **proj** outputs the cartesian coordinates in meters, the data are directed to program **linepic**.

Program **linepic** is similar to program **pointpic** as it processes the first four pairs of coordinates it receives. These coordinates must be the cartesian coordinates, in meters, of the map boundaries. From these four pairs of coordinates, the size of the image file is computed and the file is created on disk. Subsequent data records are assumed to be the actual data file.

As each record of the data file is obtained, it is scanned to see if it contains any MAPGEN command. A record with a MAPGEN command is any record that contains a pound sign (#) as the first position of the record. If the record is determined to be MAPGEN command, it is then further scanned to determine whether it contains the MAPGEN command -b to flag the start of a new line segment. If the *-b* is found in the record, the program flags the next pair of coordinates processed as the beginning of a line segment. In plotting terms, the *-b* flags the program to pick up the pen and proceed to the next coordinate before continuing to draw.

If the data record does not contain a MAPGEN command, it is assumed the data record contains a pair of vector coordinates. The program reads the data file until it has obtained two consecutive pairs of vector coordinates. From these two pairs of coordinates, program **linepic** computes the points necessary to generate a line connecting the supplied coordinates. The user specified dn_value is then placed at the computed coordinates. Coordinates that fall outside of the image area are dropped producing a line clipped at the map boundaries.

To produce the example image described, the **proj** and **linepic** run-lines are summarized below. It is assumed that the coastline vector data has been previously retrieved and is stored in the file *ne_coast.dat*.

```
%  proj +proj=utm +lon_0=-69 -r -s -m 1:250 -f ’%.0f’ \
   bounds2.dat ne_coast.dat | linepic -o ne_coast.pic
```

If the user does not have access to the Branch’s coastline files, they may easily create their own files by map digitization and conversion of the data file with MAPGEN program **mcoast**. Program **mcoast** is available as part of the MAPGEN system and is documented via a UNIX man page distributed with the software. Even if it is not the user’s intent to create a MAPGEN coastfile, easy creation of a vector file is all that is required to prepare a vector data file for rasterization.

The following image, Figure 5, was created using the same cartographic specifications and was created using the standard Branch coastline information from World Data Bank-II. Because the image is significantly reduced for presentation here, it does not adequately illustrate the detail contained in the coastline file. It is the users decision whether or not he or she wish to convert his or her vector data to a coastline file, though conversion may help to reduce the impact on disk storage requirements while still providing platform independence access.

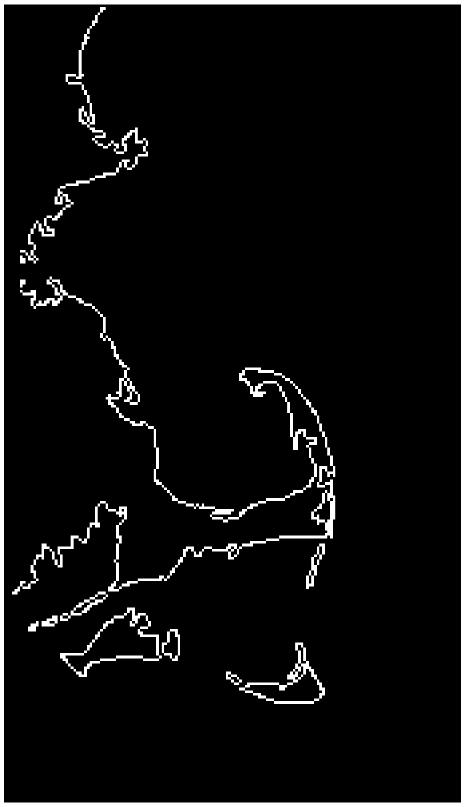

Figure 5 - Rasterized vector coastline data for eastern Massachusetts.

The vector rasterization procedure may also be applied to data such as contour line and grid line information. Separate images containing such information may then be combined together or with other image files to create a more complete map. To illustrate this type of procedure, an image containing coastline, bathymetry contour and grid line information was created and is shown as Figure 6.

The final figure builds on the previous example. The map projection information remains the same. The scale of the new map is 1000 meter pixels, and the map bounds of the area are 39^o to 43^o north latitude and 72^o to 68^o west longitude. A new coastline image for the enlarged area is created, as well as, the additional images with bathymetry contours and grid line information. The coastline information selected is the World Data Bank-II file and is accessed in the same manner as previously described in Figure 5.

The bathymetry contour line information is retrieved from the Branch *coastline* file, *neatl_cnt*, using the MAPGEN program **getcoast**. The user should be familiar with the available *coastfiles*. An important feature of these files is the assignment of feature codes which permits the selective retrieval of various information. The user should also be familiar with the content of the selected *coastfiles* since the use of feature codes is significantly different for those files containing coastline information as compared to those files containing bathymetry contour information. For those *coastfiles* containing bathymetry contour information, the feature codes (**-f**), when utilized, are assigned to various contour levels. It is important to note that for the individual bathymetry *coastfiles* the feature codes and contour levels differ from file to file and the user should check the local site documentation for specific codes and feature assignments. When utilizing the feature code on the **getcoast** runline, specifying a single feature code or a range of feature codes, allows the user the ability to extract selected information from the file. Not specifying a feature code on the **getcoast** runline will result in all the information contained in the file being extracted.

For convenience and the use of illustrative purposes here, all contour line data, regardless of feature codes, will be extracted and combined as one image with a single dn value of 250 will be assigned to the lines. This contour file included the 20 through 200 meter contours at 20 meter intervals, and the 300 through 3100 meter contour at 100 meter intervals. The run-line to extract the vector data, project the information, and create the image is shown below.

```
% getcoast -72 -68 39 43 -r \
  /usr/coast/bathy/neatl_cnt | \
  proj +proj=utm +lon_0=-71 \
  -r -s -m 1:1000 -f '%.0f' bounds3.dat - | \
  linepic -o neatl_cnt.pic -d 250
```

Generally it would be recommended to extract the individual contour line data separately for a single contour line. Extracting the vector information for each bathymetry contour level provides the ability to assign a unique dn value to a specific contour line. Below is an example selecting only the 500 meter contour line information (*-f 13*) from the *neatl_cnt* file and assigning the dn value of 25 to the line in the image created by **linepic**. By designating a unique dn value to each contour line, the user will have the option of later color coding the individual contour lines or interpolating between the lines to produce a bathymetric map of the area. Though creating individual images may provide more options for combining various data sets, producing the individual images requires multiple **getcoast**, **proj** and **linepic** executions.

```
% getcoast -72 -68 39 43 -r -f 13\
  /usr/coast/bathy/neatl_cnt|
  proj +proj=utm +lon_0=-71 \
  -r -s -m 1:1000 -f '%.0f' bounds3.dat - | \
  linepic -o neatl_cnt0500.pic -d 25
```

The next procedure creates an image with grid lines for the map area. The map boundaries will be provided by the *bounds3.dat* file. Rather than simply drawing a line from corner to corner of the map area, we will create a series of coordinates to draw the vector grid lines. In many cases, drawing a straight line between map corners is not appropriate because the grid line would not accurately represent the curvature of the selected projection. By producing a series of vector coordinates along the desired grid line, the line drawn will more accurately represent the projection. To simplify the creation of the vector coordinates a program, **grid_data**, has been developed. The user may utilize this program to automatically generate the vector coordinates as well as the intermediate grid lines between the map bounds. By default, **grid_data** will generate grid lines for every degree of latitude and longitude between the map bounds.

```
% grid_data <bounds3.dat | proj +proj=utm +lon_0=-71 \
  -r -s -m 1:1000 -f '%.0f' bounds3.dat - | \
  linepic -o neatl_grid.pic -d 250
```

The coastline, bathymetry contour and grid line images were combined to create the image and is shown as Figure 6. The image shows the limits of the selected bathymetric file. The entire processing script to produce this image can be found in Appendix B.

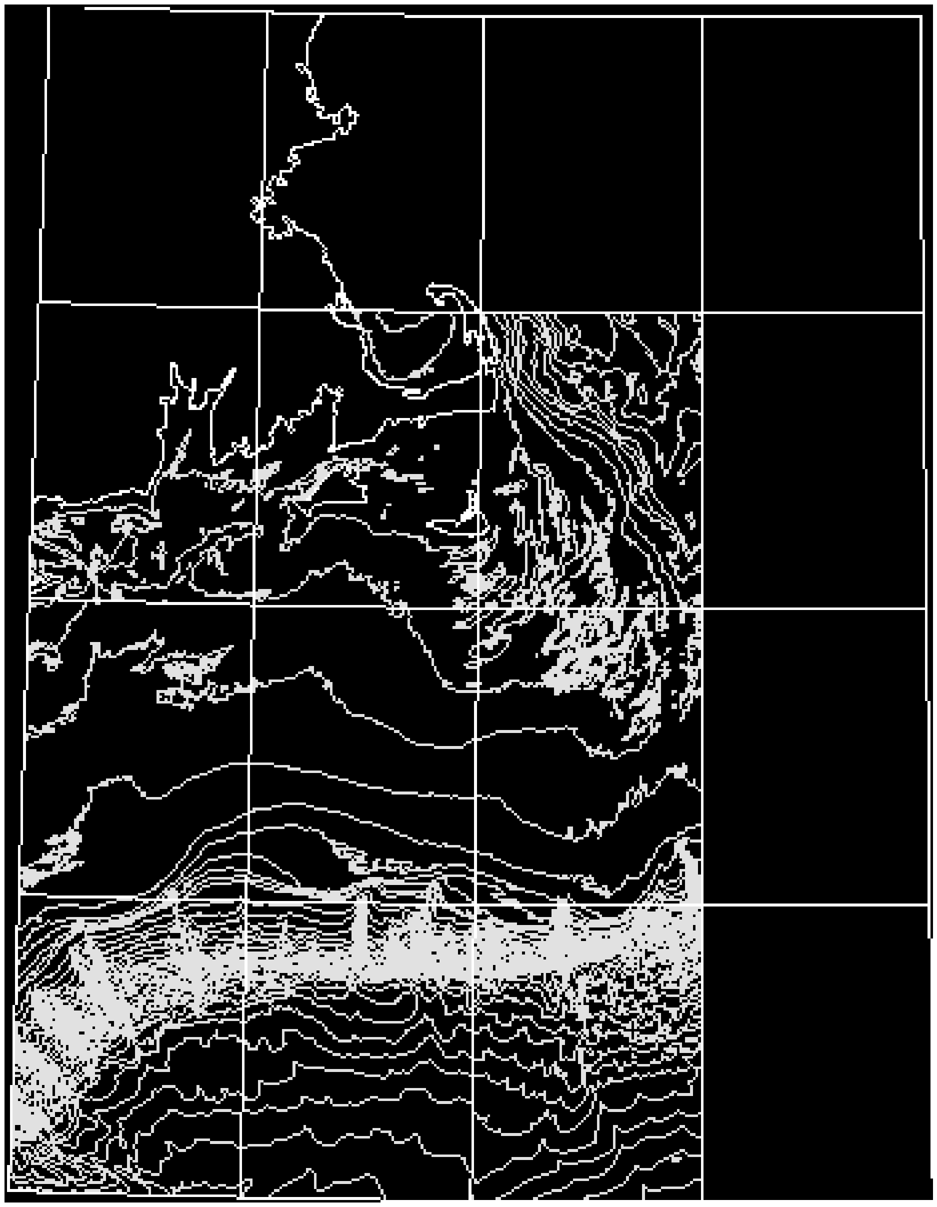

Figure 6 - Combined coastline, bathymetry contours and map graticules.

Summary

Programs and procedures presented here provide a simple method for rasterization of vector and discrete data to create an image that represents the data in a specific map scale and projection. The examples illustrate various options available to the user to create images that may be integrated with other images or easily imported to other software packages.

APPENDIX A

Woods Hole Image Processing System (WHIPS) Program Documentation

WHIPS Programs

derivative	computes first order difference of an image
dk2dk	creates a new image file by extracting a sub-area from an existing image
filter	applies a low-pass, high-pass, zero replacement or divide filter to an image
grid_data	computes vector coordinates to draw map area grid lines.
linepic	takes a vector file with geographic coordinates which make up a line and rasterizes the line data in a WHIPS netCDF image file that represents a defined map space.
lowpass2b2	applies a 2-by-2 low-pass filter to a WHIPS image
median3	applies a 3-by-3 median filter to a WHIPS image
mode3	applies a 3-by-3 mode filter to a WHIPS image.
mode5	applies a 5-by-5 mode filter to a WHIPS image.
pointpic	takes a file that contains projected geographic coordinates and an associated data value and places the data value in a WHIPS netCDF image file that represents a define map space.
qmos	mosaics (overlays) the specified input file over the specified output file. The program will either overlay where the input file has priority over the existing output file, where the output file has priority over the input file, or average non-zero pixel values together from the input and existing output file.
quickview	displays an 8-bit WHIPS netCDF image in an X-11 window
raw2whips	converts a raw image data file to a netCDF file for use with WHIPS
whips2raw	converts a WHIPS netCDF image file to a raw image file

NAME

derivative - compute first order difference of an image

SYNOPSIS

derivative -i *input* **-o** *output* [**-h** | **-v** | **-d**] [**-a** *addback*] [**-H**]

DESCRIPTION

The **derivative** program computes a simple difference between adjacent pixels in an image. The derivative may be computed in one of three directions: horizontal, vertical or diagonal. The user must specify the direction of the derivative to be performed by selecting **-h**, **-v** or **-d** as part of the run-line.

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

-h | **-v** | **-d**

specifies which type of derivative to perform. Valid selections are horizontal (**-h**), vertical (**-v**) or diagonal (**-d**).

The *horizontal derivative* is processed on a line by line basis where the adjacent pixels of one single line are subtracted from each other across the image line. The image lines are processed one at a time starting with the first pixel of the image line to the last pixel of the image line with the following equation where ix is the sample of the image being processed.

line1[ix] = (line1[ix] - line1[ix+1]) + iaddback

A special case exists for the last sample of the image line because it has no adjacent value to be used in the computation. The last sample is computed as:

hder[ix] = hder[ix-1]

The *vertical derivative* is computed using two consecutive image lines. The sample locations of the image are subtracted from line to line. The first input image line is a special case scenario. This line is computed by subtracting the individual sample from itself and is computed as:

line1[ix] = (line1[ix] - line1[ix]) + iaddback

or more simply as:

line1[ix] = iaddback

The remaining image lines of the input file are processed as:

line2[ix] = (line1[ix] - line2[ix]) + iaddback

The diagonal derivative is computed by subtracting the offset sample of one line from the sample of another line. The first line of the image file is a special circumstance and is computed as a horizontal derivative. The remaining image lines are computed as:

line2[ix] = (line1[ix] - line2[ix+1]) + iaddback

A special circumstance exists for the last sample of each line. Those samples are computed as:

line2[ix] = line2[ix-1]

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

-a *addback*

specifies the addback value to be used by the program when computing the derivative. The default value is 127 for an 8-bit image and 4000 for a 16-bit image.

-H

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

RESTRICTIONS

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

The output file to be created must not currently exist.

SEE ALSO

WHIPS(5)

DIAGNOSTICS

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

BUGS

The 16-bit option of the program has not been completely tested.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

-i i*nput_file*

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

-o *output_file*

specifies the output file to be created.

-a *sl,ss,nl,ns*

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

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

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

-l *linc*

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

-s *sinc*

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

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

NOTES

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

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

EXAMPLE

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

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

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

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

SEE ALSO

ssdk2dk(1)

WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

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

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

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

-b *nl,ns*

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

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

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

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

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

(i,j) - The image coordinate of the pixel being computed. For line 29 and sample 12, the image coordinate would be (29,12).

P(i,j) - The original pixel value of the input image at coordinate i,j.

S(i,j) - The sum of the points over the boxcar centered at i,j.

N(i,j) - The number of valid points within the boxcar surrounding the pixel being processed.

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

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

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

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

$$X(i,j) = COEF*LPF(i,j)$$

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

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

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

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

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

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

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

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

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

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

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

-a *addback*

specifies the addback value which is used by the *high-pass* and *divide filters* only.

-v *minval,maxval*

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

-c *coef*

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

-n *norm*

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

-m *mingood*

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

-f *fraction_good*

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

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

SEE ALSO

/usr/include/limits.h

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

WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

grid_data - compute vector coordinates to draw map area grid lines

SYNOPSIS

grid_data <std_in >std_out [**-P** *lat_intv*] [**-M** *lon_intv*] [**-C** c*urve_intv*] [**-H**]

DESCRIPTION

The **grid_data** program will compute a series of geographic coordinates at a specified interval between a set of user supplied map corners. These map corners are used as end-point coordinates to compute a series of geographic vector coordinates that represent the grid lines for the map area. The computed coordinates are then used as vector coordinates to produce a grid line. The program computes additional coordinates between the supplied map corners to produce a more accurate representation of the grid line for the selected map projection. Additionally, a series of vector coordinates are computed for each of the latitude (**-P**) and longitude (**-M**) intervals specified. Each grid graticule line output is indicated at the start of the line segment by a comment line identifying the grid line, and the **MAPGEN** command line **#-b**.

The program input and output is via *std_in* and *std_out*. This option was implemented to allow for piping of the data through a series of steps to facilitate a stream processing. The main goal of the program and the proposed processing are to generate a series of geographic vector coordinates for a given grid line, project those coordinates into a specified map space and rasterize the grid line(s) into a WHIPS netCDF image. Once the WHIPS image containing the grid has been created, it may be overlayed atop any other WHIPS netCDF image representing the identical map area by program **qmos**.

Program input is through std_in. As stated above, the input to program **grid_data** is four pairs of geographic coordinates that represent the map corners. The map corner coordinates must be specified in the following order:

upper left
upper right
lower right
lower left

A typical input file may look like:

```
42 -71     # upper left corner
42 -69     # upper right corner
41 -69     # lower right corner
41 -71     # lower left corner
```

By default, the grid line intervals (**-M** and **-P**) are every 1^o and the *curve_intv* (**-C**) is every .01^o. The *curve_intv* helps produce a series of vector coordinates to better approximate the grid line. Assuming an example with map bounds of 41^o to 42^o latitude and -71^o to -69^o longitude, vector coordinates for every latitude and longitude within the map corners will be produced. This grid interval would include 41^o and 42^o latitude and -71^o, -70^o and -69^o longitude. Along each grid line, the vectors would be computed at a .01^o interval. This results in a series of 201 points between the -71^o and -69^o longitude coordinates being produced for every latitude line output. A portion of the line information for latitude 41^o is shown below.

```
# Latitude: 41.000000
# -b
 41.000000      -71.000000
 41.000000      -70.990000
 41.000000      -70.980000
 41.000000      -70.970000
 41.000000      -70.960000
 41.000000      -70.950000
 41.000000      -70.940000
 41.000000      -70.930000
 41.000000      -70.920000
 41.000000      -70.910000
 41.000000      -70.900000
 41.000000      -70.890000
 41.000000      -70.880000
 41.000000      -70.870000
 41.000000      -70.860000
 41.000000      -70.850000
 41.000000      -70.840000
 41.000000      -70.830000
 . . .
 . . .
 . . .
 41.000000      -69.090000
 41.000000      -69.080000
 41.000000      -69.070000
 41.000000      -69.060000
 41.000000      -69.050000
 41.000000      -69.040000
 41.000000      -69.030000
 41.000000      -69.020000
 41.000000      -69.010000
 41.000000      -69.000000
# Latitude: 42.000000
# -b
 42.000000      -71.000000
 42.000000      -70.990000
 42.000000      -70.980000
 . . .
 . . .
```

There are no required run-line options to be specified other than the UNIX std_in and std_out specifications.

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

-P *lat_intv*

specifies the interval (*lat_intv*) between the map latitude boundaries (parallels) at which to compute and output the latitude grid line vector coordinates. The value must be specified in decimal degrees. The default value is 1. The grid line vector coordinates are processed starting with the map's minimum latitude boundary. The grid line vectors for the remaining lines are then calculated at the user requested *lat_intv* until the computed line is greater than the maximum map latitude boundary.

-M *lon_intv*

specifies the interval (*lon_intv*) between the map longitude boundaries (meridians) at which to compute and output the longitude grid line vector coordinates. The value must be specified in decimal degrees. The default value is 1. The grid line vector coordinates are processed starting with the map's minimum longitude boundary. The grid line vectors for the remaining lines are then calculated at the user requested *lon_intv* until the computed line is greater than the maximum map longitude boundary.

-C *curve_intv*

specifies the interval (curve_intv) at which to output line vector coordinates. The value may be specified as decimal degrees. The default *curve_intv* value is .01.

As the geographic vector coordinates for a given grid line are being computed and output, additional coordinates between the end points are output. This is done to better represent the map grid line in various projections. Though some projections produce a square or rectangular map area (e.g. Mercator) where the graticule lines would be straight lines, simply drawing a connecting line between the map corner coordinates would be acceptable. However, in the case of a conic or cynlindrical projection, drawing a straight line to connect a pair of map coordinates which are the upper left and right corner would be incorrect since the grid line would not represent the true curveture of the graticule lines. By producing a line with additional geographic coordinates between the end-points, the grid line, when rasterized, will more accurately represent the map graticule lines.

See further explaination of specifying the *curve_intv* value in the **NOTES** section.

-H

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

RESTRICTIONS

none known

NOTES

When specifying the *curve_intv* (**-C**), it is important to specify a value that will evenly divide the latitude and longitude intervals (**-P** and **-M**) to assure that the map corner boundaries are output. The user should specify a *curve_intv* (**-C**) which is an even increment of the map boundaries. One or two oddities may occur if the *curve_intv* is not an even increment of the map boundaries. One oddity which might occur is that the right longitude and top latitude may not be computed properly and therefore a line for these graticules may not be drawn. A second oddity might be that latitude and longitude grid lines may not be drawn to intersect their respective map boundary counterpart. It may also be a possibility that the lines may be drawn to intersect and, possibly, extend beyond their boundary counterpart. To obtain the best drawn grid lines, the curve_intv value should be at some interval such as .1, 01., .001 or .02.

EXAMPLE

The example below shows a simple application of the program to compute the default vector grid line coordinates for a given map area. The input file, *bounds.dat*, is shown below.

```
% grid_data <bounds.dat >grid.dat
```

```
% more bounds.dat
42 -71
42 -69
41 -69
41 -71
```

The following example illustrates how to utilize **grid_data** along with programs **proj** and **linepic** to generate a WHIPS netCDF image file which contains an image of the grid line. The *bounds.dat* file in this example is the same file as used in example 1.

```
% grid_data <bounds.dat | proj +proj=utm +lon_0=-69 -r -s \
  -m 1:100 -f ’%.0f’ bounds.dat - | linepic -o grid.pic
```

SEE ALSO

linepic(1), proj(1), qmos(1)

WHIPS(5)

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

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

DIAGNOSTICS

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

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

linepic - take a vector file with geographic coordinates which make up a line and rasterize the line data in a WHIPS netCDF image file that represents a defined map space.

SYNOPSIS

linepic -o *output* < std_in [**-b** *bittyp*] [**-d** *line_value*] [**-H**]

DESCRIPTION

Linepic will process the projected geographic coordinates of a line and produce an image which contains the rasterized line information. This program and associated procedures provide a simple way to rasterize various vector line data (e.g. contours, coastlines, data interpretations) into an image. The output image file may be 8, 16 or 32-bit image and can be specified by the user by selecting the **-b** option on the program run-line. The default is to create an 8-bit image.

The line represented by the vector data file may be one continous line segment or consist of two or more line segments. If the line consists of more than one line segment, each line segment must begin with the **MAPGEN** command line *# -b*. The program will compute the points necessary to connect the vector line coordinates and produce a continous line segment from point to point. The user may select the pixel value to be output to represent the data line by specifying the **-d** option on the program run-line. If the user does not specify a *line_value*, the program defaults to outputting the line with a value of 255. The value specified by the user must be appropriate for the output image bittype. If portions of the data file are to be represented by two or more *line_values*, the file must be split and processed separately for each of the representative values.

The first four records of the input file must contain the map area bounds, in meters, for the user desired map area. Subsequent data records must contain the latitude and longitiude geographic coordinates. The geographic coordinate for each vector contained in the input file must have been previously converted by program **proj** to their meter coordinates for the selected map area. The map coordinates must be in *y x* (latitude longitude) order and the coordinate pairs must be integer values.

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

upper left
upper right
lower right
lower left

For example, if the user desires to create a 1^o x 2^o map at a scale of 1:250 for an area bounded by 41^o to 42^o latitude and -71^o to -69^o longitude for a simple UTM (Universal Transverse Mercator) map with a central longitude value of -69^o, the first four records of the input file would contain the following information:

```
46515   3344
46496   5000
45385   5000
45405   3318
```

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

The remainder of data in the input file must be the vector line coordinates which **linepic** will process. Program **linepic** accepts and calculates the location (the actual image line and sample coordinate) within the map space for the line coordinates. When the program has obtained two consecutive pairs of points, the program will compute the series of points required to *draw* a continous line between the points. The user specified *line_value* is placed in the computed image coordinates, on a pixel-by-pixel basis, creating a line of data. It is important to note that the pixel placement is done on a pixel-by-pixel basis. When multiple dn values are computed for the same location, **linepic** will always place the last pixel input in the output location regardless of the coordinates previous content.

Program **proj** is essential in creating the final map product. The user should be familiar with its usage and various options. Some **proj** options must be specified to create the proper data output for program **linepic**. Those options include: **-r**, **-s** and **-f '%,0f'**.

Many selected projections produce a conic or trapezoid map area while the output image is rectangular. Because the map area is placed in a rectangle defined to fit the largest dimension of the map area, the map corner coordinates will, in most cases, not relate to the image corners. To help the user better identify the map area within the image, the location of the map corners are reported to the program print file. Map corner coordinates are reported as image *line, sample* coordinates.

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

-o *output_file*

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

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

-b *bittyp*

specifies the bit type (*bittyp*) of the output file to be created. Valid selections are 8, 16 or 32. The default *bittyp* is 8.

-d *line_value*

specifies the dn value (*line_value*) of the pixels to be output to represent the line. The default output *line_value* is 255 for all output image bit types.

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

The input meter coordinates must be recorded as integer values.

The maximum number of points which can be computed from any two consecutive coordinates is 10 plus the larger of the number of samples or number of lines contained in the image.

EXAMPLE

The example below shows a simple execution of the program where the input file, *200M_contour,* will be previously processed by **proj**. The first four pairs of coordinates contained in the input file must be the projected map area bounds.

```
% linepic -o map2.cdf <200M_contour -b 16 -d 200
```

The following is a typical example using program **proj** as a filter to assist in the processing and mapping of the user selected data. The data in this short example are coastline data which are to be mapped at 100 meter resolution to a Universal Transverse Mercator projection using a central longitude of -69^{o}. The file, *bounds.dat*, must contain the map corner coordinates and is shown below. The file, *ne_coast*, contains the line points as latitude and longitude coordinates. A small portion of the coastline data file to be processed is listed following *bounds.dat*.

```
% proj +proj=utm +lon_0=-69 -r -s -m 1:100 -f ‘%.0f’ \
  bounds.dat ne_coast | linepic -o ne_coast.pic -d 255

% more bounds.dat
42 -71            # upper left corner
42 -69            # upper right corner
41 -69            # lower right corner
41 -71            # lower left corner

% head -10 ne_coast
# -b
42.000000       -70.689270
41.997201       -70.682274
41.993974       -70.676993
41.990161       -70.672006
41.986641       -70.666433
41.982827       -70.662619
41.978427       -70.659685
41.973440       -70.656459
41.969626       -70.654112
```

The above procedure could have been accomplished by utilizing the **getcoast** program of **MAPGEN**. The coastline vector information for a given area can be extracted directly from a user specified *coastfile* and piped to the various programs for processing. An example of this procedure is shown below. The user may wish to verify the location of the coastline files on their particular system.

```
% getcoast -71 -69 41 42 -r /usr/coast/na/cil | \
  proj +proj=utm +lon_0=-69 -r -s -m 1:100 -f ‘%.0f’ \
  bounds.dat - | linepic -o bathy.pic
```

The last example and procedure shows how **proj** and **linepic** can be used to generate a quick map grid. The map grid will be every 30’ for the map area defined in the previous example. To accomplish this, a file with the coordinates for the lines must be created. The file *grid.dat* contains the coordinates for the 30’ intervals. Each grid line segment is marked at the start by a record containing the **MAPGEN** command *# -b*. Additional coordinates between the map corners have been added to create a smoother, and hopefully, more accurate representation of the grid line. The files, *ne_coast* and *grid.pic*, may later be combined using program **qmos** to create a composite image.

```
% proj +proj=utm +lon_0=-69 -r -s -m 1:100 -f ’%.0f’ \
  bounds.dat grid.dat | linepic -o grid.pic -d 254

% more grid.dat
# -b
42 -71
42 -70.5
```

```
42 -70
42 -69.5
42 -69
# -b
41.5 -71
41.5 -70.5
41.5 -70
41.5 -69.5
41.5 -69
# -b
41 -71
41 -70.5
41 -70
41 -69.5
41 -69
# -b
42 -71
41.5 -71
41 -71
# -b
42 -70.5
41.5 -70.5
41 -70.5
# -b
42 -70
41.5 -70
41 -70
# -b
42 -69.5
41.5 -69.5
41 -69.5
# -b
42 -69
41.5 -69
41 -69
```

SEE ALSO

filter(1), grid_data(1), pointpic(1), proj(1), qmos(1)

WHIPS(5)

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

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

NOTE

The user may process individual line segments as separate files and combine the files with **qmos**.

DIAGNOSTICS

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

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

```
              input           output
           -------------   ------------
   line 1:|  11  |  22  |  |  11 |  22  |
           -------------   ------------
   line 2:|  33  |  44  |  |  28 |  44  |
           -------------   ------------
```

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

```
          input                      output
        -------------------     --------------------
line 1: | 11 | 22 | 33 | 44 |    | 17 | 28 | 39 | .. |
        -------------------     --------------------
line 2: | 33 | 44 | 55 | 66 |    | 28 | 39 | 50 | .. |
        -------------------     --------------------
```

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

```
          input                   output
        --------------          ---------------
line 1: | 11 | 22 | 44 |         | 17 | 33 | 44 |
        --------------          ---------------
line 2: | 33 | 44 | 66 |         | 28 | 88 | 55 |
        --------------          ---------------
line 3: | 55 | 66 | 88 |         | 50 | 66 | 77 |
        --------------          ---------------
line 4: | 77 | 88 | 99 |         | 72 | 85 | 94 |
        --------------          ---------------
```

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

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

-H

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

RESTRICTIONS

The program accepts only 8-bit image files.

The output file to be created must not currently exist.

EXAMPLE

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

SEE ALSO

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

WHIPS(5)

DIAGNOSTICS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

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

-z

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

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

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

NOTES

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

SEE ALSO

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

WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

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

-z

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

-Z

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

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

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

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

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

NOTES

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

SEE ALSO

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

WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

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

-i *input_file*

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

-o *output_file*

specifies the output file to be created.

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

-z

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

-Z

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

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

EXAMPLE

The example below shows a simple execution of the program.

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

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

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

NOTES

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

SEE ALSO

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

WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

pointpic - take a file that contains projected geographic coordinates and an associated data value and place the data value in a WHIPS netCDF image file that represents a defined map space.

SYNOPSIS

pointpic -o *output* < std_in [**-b** *bittyp*] [**-H**]

DESCRIPTION

Pointpic will take a data value (z) from a file which contains it's geographic coordinates projected into a map space and place the data value in an image file that represents a defined *map* for a specific area, projection and scale. The data value associated with the geographic coordinate may be either an 8, 16 or 32-bit value. This program and accompanying procedure provides a simple way to rasterize discrete data values into an image.

The first four records of the input file must contain the map area bounds, in meters, for the user desired map area. Subsequent data records must contain the latitude, longitiude and data value (y,x,z). The geographic coordinate for each data value contained in the input file must have been previously converted by program **proj** to their meter coordinates for the selected map area. The map coordinates must be in *y x* (latitude longitude) order and the coordinate pairs must be integer values.

Program input is through std_in. As stated above, the actual input to program **pointpic** is a file of discrete data values each with an associated geographic coordinate which have been converted to meter values based on a specific map projection and scale. The first four records of the input file **MUST BE** the map corner coordinates, in meters, for the map projection and scale. The map corner coordinates must be specified in the following order:

upper left
upper right
lower right
lower left

For example, if the user desires to create a 2^o x 1 $1/2^o$ map at a scale of 1:250 for an area bounded by 24^o to 26^o latitude and -85^o to -83.5^o longitude for a simple UTM (Universal Transverse Mercator) map with a central longitude value of -86^o, the first four records of the input file would contain the following information:

```
11508   2801
11521   3402
10634   3425
10622   2814
```

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

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

Program **proj** is essential in creating the final map product. The user should be familiar with its usage and various options. Some **proj** options must be specified to create the proper data output for program **pointpic**. Those options include: **-r**, **-s** and **-f '%,0f'**.

Many selected projections produce a conic or trapezoid map area while the output image is rectangular. Because the map area is placed in a rectangle defined to fit the largest dimension of the map area, the map corner coordinates will, in most cases, not relate to the image corners. To help the user better identify the map area within the image, the location of the map corners are reported to the program print file. Map corner coordinates are reported as image *line, sample* coordinates.

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

-o *output_file*

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

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

-b *bittyp*

specifies the bit type (*bittyp*) of the output file to be created. Valid selections are 8, 16 or 32. The default *bittyp* is 8.

-H

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

RESTRICTIONS

The output file to be created must not currently exist.

The input meter coordinates must be recorded as integer values.

EXAMPLE

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

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

The following is a typical example using program **proj** as a filter to assist in the processing and mapping of the user selected data. The data in this short example is bathymetry data which are to be mapped at 250 meter resolution to a Universal Transverse Mercator projection using a central longitude of -86^o. The file, *bounds.dat*, must contain the map corner coordinates and is show below. The file *bathy.dat* contains the latitude and longitude coordinates to be projected along with the bathymetry data values to be placed in the image. A small portion of the bathymetry data file (*bathy.dat*) to be processed is listed following *bounds.-dat*.

```
% proj +proj=utm +lon_0=-86 -r -s -m 1:250 -f '%.0f' \
  bounds.dat bathy.dat | pointpic -o bathy.pic -b 16

% more bounds.dat
26 -85            # upper left corner
26 -83.5          # upper right corner
24 -83.5          # lower right corner
24 -85            # lower left corner
```

```
% head -10 bathy.dat
   26.30870   -84.85630    1318
   26.31240   -84.85710    1317
   26.31620   -84.85800    1326
   26.32000   -84.85890    1335
   26.32390   -84.85980    1345
   26.32770   -84.86060    1348
   26.33150   -84.86150    1352
   26.33540   -84.86250    1356
   26.33920   -84.86340    1393
   26.34300   -84.86440    1430
```

SEE ALSO

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

proj(1), projss(1)

WHIPS(5)

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

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

NOTE

This program is similar to program **projss**. The difference between this program and **projss** is the ability of **pointpic** to handle 8, 16 or 32-bit data.

DIAGNOSTICS

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

BUGS

None known.

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

qmos - quick mosaic of two WHIPS netCDF images

SYNOPSIS

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

DESCRIPTION

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

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

-i *input_file*

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

-o *output_file*

specifies the output file to be updated. The output file **must** currently exist.

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

-I

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

-O

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

-A

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

-H

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

RESTRICTIONS

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

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

EXAMPLE

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

SEE ALSO

WHIPS(5)

DIAGNOSTICS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

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

-i *input_file*

specifies the input file to be processed. The input file may only be 8-bit.

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

-a *sl,ss,nl,ns*

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

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

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

-c *comp*

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

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

-H

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

RESTRICTIONS

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

NOTES

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

EXAMPLE

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

SEE ALSO

WHIPS(5)

DIAGNOSTICS

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

BUGS

none known

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

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

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

-i *input_file*

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

-o *output_file*

specifies the netCDF output file to be created.

-l *nl*

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

-s *ns*

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

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

-b *bittyp*

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

-H

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

RESTRICTIONS

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

The output file to be created must not currently exist.

EXAMPLE

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

SEE ALSO

whips2raw (1), WHIPS(5)

DIAGNOSTICS

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

BUGS

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

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

NAME

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

SYNOPSIS

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

DESCRIPTION

Program **whips2raw** converts a WHIPS netCDF image file to a raw binary stream image file.

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

-i *input_file*
: specifies the netCDF file to be processed.

-o *output_file*
: specifies the binary stream input file to be converted. The input file may be 8, 16 or 32-bit.

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

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

RESTRICTIONS

none known

EXAMPLE

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

SEE ALSO

raw2whips(1), WHIPS(5)

DIAGNOSTICS

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

BUGS

none known

AUTHOR/MAINTENANCE

Valerie Paskevich, USGS, Woods Hole, MA.

APPENDIX B

Cookbook examples

COOKBOOK EXAMPLES

There are two forms of rasterizing data discussed in this report. The first is the rasterization of discrete data, and the second is the rasterization of vector data. This section provides a summary of the programs and procedures used to produce the figures shown in this report. These processing steps may be used as a guide by the user in creating scripts for his or her own applications. A detailed description of the steps and procedures are not discussed here though comments are interspersed among the processing commands for clarity. The commands to convert, extract and display the figure images on an X-11 display are also listed. Programs and procedures utilized to convert the image to an Encapsulated PostScript (EPS) file for printing have not been included.

• Cookbook Example 1 - Rasterizing point data.

The first example illustrates a procedure to rasterize discrete data values and create a map representing the distribution of bathymetry in a small area of the West Florida escarpment. The data used in this example are single point soundings collected during a survey of the area in 1985. The complete data coverage is an area from 29.2106^o to 23.8792^o north and 88.2697^o to 82.0021^o west. The area to be mapped in this example is a subset of the full data set and will be defined by the bounds of 24^o to 26^o north and 85^o to 83.5^o west. This example covers the steps utilized in filtering the data to produce an image with complete bathymetry coverage. It is assumed that the data has been compiled, properly formatted and stored in file *farn3.bt* for processing.

Before beginning the rasterization process, two supplementary files must be created. The files may be created by invoking the editor and will contain the map image geographic bounds & cartographic information. Though the user may select to enter the information entirely on the run-line, creating the supplementary files before beginning the processing is highly recommended. Entering the information into a file will help reduce typographic errors and, if needed, provide access to the redundant information for additional processing.

The first file is the *wfla_bounds.dat* file and contains the map image geographic boundaries. The second file, *utm-proj*, contains the cartographic specifications for program **proj**. The UNIX command to display the contents of those files and the file contents are shown below. Note the inclusion of the *wfla_bounds.dat* file on the **proj** run-line in file *utm-proj*. Also note the specification of the optional **proj** parameters, **-r**, **-s** and **-f**. Though these are optional parameters for the execution of **proj**, they are required specifications to complete the processing procedures discussed here.

- Supplemental files:

```
% more wfla_bounds.dat
26 -85
26 -83.5
24 -83.5
24 -85

% more utm-proj
proj +proj=utm +lon_0=-86 -r -s -m 1:250 -f ‘%.0f’ wfla_bounds.dat farn3.bt
```

Once the **proj** parameters have been selected and the supplementary files have been created, processing of the data may begin. The first command shows the procedure to convert the geographic coordinates to the selected cartesian system and creates a 16-bit WHIPS netCDF image. The following commands contained in the block show the programs and procedures used to convert the 16-bit WHIPS netCDF image to an 8-bit WHIPS image and extract the sub-area used as the figure from the larger image. The command to display the 8-bit image in an X-11 window is also included.

- Figure 2:

```
% utm-proj | datapic -b 16 -o wfla_bt.cdf
#
#   convert the image to 8-bit and select the sub-area
#
% bit2bit -i wfla_bt.cdf -o wfla_bt.8bit -v 500,3500
% dk2dk -i wfla_bt.8bit -o wfla_bt.sub -a 430,200,240,175
#
#   View the 8-bit subarea of the image on the workstation screen
#   with quickview.  If it’s okay, remove the temporary files.
#
% quickview -i wfla_bt.sub
% rm wfla_bt.8bit
% rm wfla_bt.sub
```

A series of filters were applied to Figure 2, *wfla_bt.cdf*, to distribute the bathymetry data more completely throughout the image. The processing procedure applied and shown here is subjective to the wishes of the user. The filters and filter sizes were chosen to fill the image as quickly as possible. The summary of the processing steps are meant to be representative of processing that can be applied to the image and should not to be considered the ultimate processing solution.

- Figure 3:

```
% filter -i wfla_bt.cdf  -o bt.lpfz1 -z -b 3,3
% filter -i bt.lpfz1 -o bt.lpfz2 -z -b 3,3
% filter -i bt.lpfz2 -o bt.lpfz3 -z -b 5,5
% filter -i bt.lpfz3 -o bt.lpfz4 -z -b 23,23
% filter -i bt.lpfz4 -o bt.lpfz5 -l -b 7,7 -v 500,3550
% filter -i bt.lpfz5 -o bt.lpfz6 -z -b 7,7
% filter -i bt.lpfz6 -o bt.lpfz7 -z -b 9,9
% filter -i bt.lpfz7 -o bt.lpfz8 -z -b 15,15
% filter -i bt.lpfz8 -o bt.lpfz9 -z -b 7,7
% filter -i bt.lpfz9 -o bt.lpfz10 -z -b 11,11
#
#   The final smoothing filter
#
% filter -i bt.lpfz10 -o bt_16final.cdf -l -b 7,7
% dk2dk -i bt_16final.cdf -o bt_16final.sub -a 430,200,240,175
% minmaxdn -i bt_16final.sub
% bit2bit -i bt_16final.sub -o figure03.cdf -v 600,3500
% rm bt.lpfz1 bt.lpfz2 bt.lpfz3 bt.lpfz4 bt.lpfz5
% rm bt.lpfz6 bt.lpfz7 bt.lpfz8 bt.lpfz9
% rm bt_16final.sub
```

Once the image has been completely filtered, a first-order derivative was produced from the bathymetry. The derivative, shown as Figure 4, was produced with the following programs and their parameters are listed below. The derivative is computed on the entire 16-bit WHIPS netCDF image. The sub-area, shown as Figure 4, was then extracted and converted from 16-bit to 8-bit and converted to an Encapsulated PostScript (EPS) file for printing purposes.

- Figure 4:

```
% derivative -i bt.lpfz10 -o x04.dder -d
% dk2dk -i x04.dder -o xfinal.sub -a 430,200,240,175
% bit2bit -i xfinal.sub -o figure04.cdf -v 3950,4150
% rm x04.dder
% rm xfinal.sub
```

• Cookbook Example 2 - Rasterizing vector data.

The final set of figures illustrate the ability to rasterize vector data to create several image files that can be combined together. Generally, these rasterized vector data files are utilized as overlays. However, rasterized vector data, such as contour lines, may be processed to produce representative images of the seafloor bathymetry. The format of the coastline and bathymetric contour files utilized in this example are specific to the Branch of Atlantic Marine Geology and were generated for compatibility with the MAPGEN system. The coastline information was created using the World Data BankII coastline vector information. The user should verify the location of the *coast* files prior to following these examples.

For this example, the Branch’s available coastline data files were accessed to extract the required vector information to draw the coastline information and create a WHIPS netCDF image file. The desired map area is 41^o to 43^o latitude and 72^o to 69.5^o longitude. The final image will be created in a UTM projection with a map scale of 500 meters. Utilizing the MAPGEN program **getcoast** and the available coastline file (*/usr/coast/na/bdy*), we are able to extract the vector information that may be used to draw the coastline of eastern Massachusetts.

- Figure 5

```
%  more bounds2.dat
43 -72
43 -69.5
41 -69.5
41 -72

% more utm2-proj
proj +proj=utm +lon_0=-69 -r -s -m 1:250 -f ’%.0f’  bounds2.dat -

% getcoast -72 -69.5 41 43 -r /usr/coast/na/bdy | utm2-proj | \
  linepic -o ne_coast.pic
```

To create the final image, coastline, bathymetry contours and map graticules are combined to create one image. The three images are created individually and combined together using WHIPS program **qmos**. Map boundaries are defined as map 39° to 43° latitude and 72° to 68° longitude.

Once again, before beginning, the supplementary files containing the geographic limits and the projection parameters of the map are created. Initially, the created file *figure06.pic* contains only the coastline information for the area. As the additional overlays are created in file *xx.pic*, the information is combined with *figure06.pic* to produce an image with the combined vector information.

- Figure 6:

```
% more bounds3.dat
43 -72
43 -68
39 -68
39 -72

% more utm3-proj
proj +proj=utm +lon_0=-71 -r -s -m 1:1000 -f ’%.0f’ bounds3.dat - |

#
#   Do the coastlines and state boundaries
#
% getcoast -72 -68 39 43 -r -f 01 /usr/coast/na/cil | utm3-proj | \
  linepic -o figure07.pic -d 254
% getcoast -72 -68 39 43 -r /usr/coast/na/bdy | utm3-proj | \
  linepic -o xx.pic -d 250
% qmos -o figure07.pic -i xx.pic
% rm -f xx.pic
#
#   Do the contour lines
#
% getcoast -72 -68 39 43 -r /usr/coast/bathy/neatl_cnt | utm3-proj | \
  linepic -o xx.pic -d 225
% qmos -o figure07.pic -i xx.pic
% rm -f xx.pic
#
#   Do the grid lines
#
% grid_data <bounds3.dat | utm3-proj | linepic -o xx.pic -d 255
% qmos -o figure07.pic -i xx.pic
% rm -f xx.pic
```

REFERENCES

Evenden, Gerald I. and Botbol, Joseph Moses, 1985, User's Manual for MAPGEN (UNIX version): a method of transforming digital cartographic data to a map, Open-File Report 85-706, 58 pages plus appendixes on font codes and map projections.

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

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

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

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

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